HALCON

a product of MVTec

HALCON/HDevelop

Operator-Reference (de)



HALCON 21.05 *Progress*

MVTec Software GmbH

Building Vision for Business

HALCON/HDevelop 21.05.0.0

Alle Rechte vorbehalten. Kein Teil dieser Publikation darf ohne vorherige schriftliche Genehmigung des Herausgebers in irgendeiner Form, sei es elektronisch, mechanisch, durch Fotokopie, Aufnahme oder andere Verfahren reproduziert, gespeichert oder übertragen werden.

Copyright © 1997-2021 MVTec Software GmbH, München, Deutschland



AMD ist ein Warenzeichen von Advanced Micro Devices, Inc.
Arm ist ein eingetragenes Warenzeichen von Arm Limited.
CUDA, cuBLAS und cuDNN sind (eingetragene) Warenzeichen von NVIDIA Corporation.
Intel ist ein Warenzeichen von Intel Corporation oder ihrer Tochterunternehmen in den U.S. und/oder anderen Ländern.
Linux ist ein Warenzeichen von Linus Torvalds.
macOS und OpenCL sind Warenzeichen von Apple Inc.
Microsoft, Windows und Microsoft .NET sind (eingetragene) Warenzeichen von Microsoft Corporation.
OpenGL ist ein Warenzeichen von Silicon Graphics, Inc.

All weiteren national und international anerkannten Warenzeichen und Handelsnamen sind hiermit anerkannt.

Mehr Information über HALCON findet sich unter: `http://www.mvtec.com`

Inhaltsverzeichnis

1 1D-Vermessung **1**

close_measure . . . 3
deserialize_measure . . . 3
fuzzy_measure_pairing . . . 4
fuzzy_measure_pairs . . . 6
fuzzy_measure_pos . . . 8
gen_measure_arc . . . 10
gen_measure_rectangle2 . . . 13
measure_pairs . . . 15
measure_pos . . . 17
measure_projection . . . 19
measure_thresh . . . 19
read_measure . . . 21
reset_fuzzy_measure . . . 22
serialize_measure . . . 22
set_fuzzy_measure . . . 23
set_fuzzy_measure_norm_pair . . . 25
translate_measure . . . 27
write_measure . . . 28

2 2D-Messtechnik **31**

add_metrology_object_circle_measure . . . 34
add_metrology_object_ellipse_measure . . . 36
add_metrology_object_generic . . . 39
add_metrology_object_line_measure . . . 42
add_metrology_object_rectangle2_measure . . . 44
align_metrology_model . . . 46
apply_metrology_model . . . 49
clear_metrology_model . . . 50
clear_metrology_object . . . 51
copy_metrology_model . . . 52
create_metrology_model . . . 52
deserialize_metrology_model . . . 53
get_metrology_model_param . . . 54
get_metrology_object_fuzzy_param . . . 55
get_metrology_object_indices . . . 56
get_metrology_object_measures . . . 57
get_metrology_object_model_contour . . . 58
get_metrology_object_num_instances . . . 59
get_metrology_object_param . . . 60
get_metrology_object_result . . . 62

get_metrology_object_result_contour . . . 64
read_metrology_model . . . 65
reset_metrology_object_fuzzy_param . . . 66
reset_metrology_object_param . . . 66
serialize_metrology_model . . . 67
set_metrology_model_image_size . . . 68
set_metrology_model_param . . . 69
set_metrology_object_fuzzy_param . . . 71
set_metrology_object_param . . . 73
write_metrology_model . . . 76

3 3D-Matching **77**
3.1 3D-Box . . . 79
find_box_3d . . . 79
3.2 Deformierbar Oberflächenbasiert . . . 82
add_deformable_surface_model_reference_point . . . 82
add_deformable_surface_model_sample . . . 83
clear_deformable_surface_matching_result . . . 84
clear_deformable_surface_model . . . 85
create_deformable_surface_model . . . 85
deserialize_deformable_surface_model . . . 88
find_deformable_surface_model . . . 89
get_deformable_surface_matching_result . . . 92
get_deformable_surface_model_param . . . 93
read_deformable_surface_model . . . 94
refine_deformable_surface_model . . . 95
serialize_deformable_surface_model . . . 96
write_deformable_surface_model . . . 97
3.3 Formbasiert . . . 98
clear_shape_model_3d . . . 98
create_cam_pose_look_at_point . . . 98
create_shape_model_3d . . . 100
deserialize_shape_model_3d . . . 107
find_shape_model_3d . . . 107
get_shape_model_3d_contours . . . 112
get_shape_model_3d_params . . . 113
project_shape_model_3d . . . 115
read_shape_model_3d . . . 117
serialize_shape_model_3d . . . 117
trans_pose_shape_model_3d . . . 118
write_shape_model_3d . . . 119
3.4 Oberflächenbasiert . . . 119
clear_surface_matching_result . . . 119
clear_surface_model . . . 120
create_surface_model . . . 121
deserialize_surface_model . . . 124
find_surface_model . . . 125
find_surface_model_image . . . 134
get_surface_matching_result . . . 135
get_surface_model_param . . . 137
read_surface_model . . . 139
refine_surface_model_pose . . . 140
refine_surface_model_pose_image . . . 142
serialize_surface_model . . . 143
set_surface_model_param . . . 144
write_surface_model . . . 148

4 3D-Objektmodell **149**
4.1 Generierung . . . 149
clear_object_model_3d . . . 149
copy_object_model_3d . . . 149
deserialize_object_model_3d . . . 152
gen_box_object_model_3d . . . 152
gen_cylinder_object_model_3d . . . 153
gen_empty_object_model_3d . . . 154
gen_object_model_3d_from_points . . . 154
gen_plane_object_model_3d . . . 155
gen_sphere_object_model_3d . . . 156
gen_sphere_object_model_3d_center . . . 157
read_object_model_3d . . . 158
remove_object_model_3d_attrib . . . 162
remove_object_model_3d_attrib_mod . . . 163
serialize_object_model_3d . . . 164
set_object_model_3d_attrib . . . 165
set_object_model_3d_attrib_mod . . . 168
union_object_model_3d . . . 169
write_object_model_3d . . . 170
4.2 Merkmale . . . 171
area_object_model_3d . . . 171
distance_object_model_3d . . . 172
get_object_model_3d_params . . . 177
max_diameter_object_model_3d . . . 182
moments_object_model_3d . . . 183
select_object_model_3d . . . 184
smallest_bounding_box_object_model_3d . . . 186
smallest_sphere_object_model_3d . . . 187
volume_object_model_3d_relative_to_plane . . . 188
4.3 Segmentation . . . 190
fit_primitives_object_model_3d . . . 190
reduce_object_model_3d_by_view . . . 192
segment_object_model_3d . . . 194
select_points_object_model_3d . . . 195
4.4 Transformationen . . . 197
affine_trans_object_model_3d . . . 197
connection_object_model_3d . . . 198
convex_hull_object_model_3d . . . 200
edges_object_model_3d . . . 200
fuse_object_model_3d . . . 202
intersect_plane_object_model_3d . . . 207
object_model_3d_to_xyz . . . 208
prepare_object_model_3d . . . 210
project_object_model_3d . . . 213
projective_trans_object_model_3d . . . 215
register_object_model_3d_global . . . 216
register_object_model_3d_pair . . . 218
render_object_model_3d . . . 220
rigid_trans_object_model_3d . . . 221
sample_object_model_3d . . . 222
simplify_object_model_3d . . . 224
smooth_object_model_3d . . . 226
surface_normals_object_model_3d . . . 228
triangulate_object_model_3d . . . 230
xyz_to_object_model_3d . . . 239

5 3D-Rekonstruktion **241**

5.1 Binokulares Stereo . . . 241
binocular_disparity . . . 241
binocular_disparity_mg . . . 244
binocular_disparity_ms . . . 250
binocular_distance . . . 252
binocular_distance_mg . . . 256
binocular_distance_ms . . . 258
disparity_image_to_xyz . . . 260
disparity_to_distance . . . 261
disparity_to_point_3d . . . 263
distance_to_disparity . . . 264
essential_to_fundamental_matrix . . . 265
gen_binocular_proj_rectification . . . 266
gen_binocular_rectification_map . . . 269
intersect_lines_of_sight . . . 272
match_essential_matrix_ransac . . . 273
match_fundamental_matrix_distortion_ransac . . . 277
match_fundamental_matrix_ransac . . . 281
match_rel_pose_ransac . . . 284
reconst3d_from_fundamental_matrix . . . 287
rel_pose_to_fundamental_matrix . . . 288
vector_to_essential_matrix . . . 290
vector_to_fundamental_matrix . . . 292
vector_to_fundamental_matrix_distortion . . . 294
vector_to_rel_pose . . . 297
5.2 Depth From Focus . . . 299
depth_from_focus . . . 299
select_grayvalues_from_channels . . . 301
5.3 Mehrbild-Stereo . . . 302
clear_stereo_model . . . 304
create_stereo_model . . . 305
get_stereo_model_image_pairs . . . 306
get_stereo_model_object . . . 306
get_stereo_model_object_model_3d . . . 307
get_stereo_model_param . . . 308
reconstruct_points_stereo . . . 309
reconstruct_surface_stereo . . . 311
set_stereo_model_image_pairs . . . 318
set_stereo_model_param . . . 319
5.4 Photometrisches Stereo . . . 326
estimate_al_am . . . 326
estimate_sl_al_lr . . . 327
estimate_sl_al_zc . . . 327
estimate_tilt_lr . . . 328
estimate_tilt_zc . . . 328
photometric_stereo . . . 329
reconstruct_height_field_from_gradient . . . 333
sfs_mod_lr . . . 334
sfs_orig_lr . . . 336
sfs_pentland . . . 337
shade_height_field . . . 338
uncalibrated_photometric_stereo . . . 340
5.5 Sheet-of-light . . . 341
apply_sheet_of_light_calibration . . . 341
calibrate_sheet_of_light . . . 342
clear_sheet_of_light_model . . . 345
create_sheet_of_light_calib_object . . . 346
create_sheet_of_light_model . . . 347

deserialize_sheet_of_light_model . . . 350
get_sheet_of_light_param . . . 351
get_sheet_of_light_result . . . 353
get_sheet_of_light_result_object_model_3d . . . 354
measure_profile_sheet_of_light . . . 355
query_sheet_of_light_params . . . 356
read_sheet_of_light_model . . . 357
reset_sheet_of_light_model . . . 358
serialize_sheet_of_light_model . . . 359
set_profile_sheet_of_light . . . 359
set_sheet_of_light_param . . . 360
write_sheet_of_light_model . . . 363

6 Bild **365**

6.1 Bildeinzug . . . 371
close_framegrabber . . . 371
get_framegrabber_callback . . . 372
get_framegrabber_lut . . . 373
get_framegrabber_param . . . 374
grab_data . . . 375
grab_data_async . . . 376
grab_image . . . 377
grab_image_async . . . 378
grab_image_start . . . 380
info_framegrabber . . . 381
open_framegrabber . . . 383
set_framegrabber_callback . . . 386
set_framegrabber_lut . . . 387
set_framegrabber_param . . . 388

6.2 Domain . . . 389
add_channels . . . 389
change_domain . . . 390
full_domain . . . 391
get_domain . . . 391
rectangle1_domain . . . 392
reduce_domain . . . 393

6.3 Format . . . 393
change_format . . . 393
crop_domain . . . 394
crop_domain_rel . . . 395
crop_part . . . 396
crop_rectangle1 . . . 397
crop_rectangle2 . . . 398
tile_channels . . . 399
tile_images . . . 400
tile_images_offset . . . 402

6.4 Generierung . . . 403
copy_image . . . 403
gen_image1 . . . 404
gen_image1_extern . . . 405
gen_image1_rect . . . 407
gen_image3 . . . 409
gen_image3_extern . . . 410
gen_image_const . . . 412
gen_image_gray_ramp . . . 414
gen_image_interleaved . . . 415
gen_image_proto . . . 418
gen_image_surface_first_order . . . 419

gen_image_surface_second_order . . . 420
interleave_channels . . . 422
region_to_bin . . . 424
region_to_label . . . 425
region_to_mean . . . 426
6.5 Kanal . . . 427
access_channel . . . 427
append_channel . . . 428
channels_to_image . . . 428
compose2 . . . 429
compose3 . . . 429
compose4 . . . 430
compose5 . . . 431
compose6 . . . 432
compose7 . . . 433
count_channels . . . 434
decompose2 . . . 434
decompose3 . . . 435
decompose4 . . . 436
decompose5 . . . 437
decompose6 . . . 438
decompose7 . . . 439
image_to_channels . . . 440
6.6 Manipulation . . . 441
overpaint_gray . . . 441
overpaint_region . . . 442
paint_gray . . . 443
paint_region . . . 444
paint_xld . . . 445
set_grayval . . . 447
6.7 Merkmale . . . 448
area_center_gray . . . 448
cooc_feature_image . . . 449
cooc_feature_matrix . . . 450
elliptic_axis_gray . . . 452
entropy_gray . . . 452
estimate_noise . . . 454
fit_surface_first_order . . . 456
fit_surface_second_order . . . 457
fuzzy_entropy . . . 459
fuzzy_perimeter . . . 460
gen_cooc_matrix . . . 461
gray_features . . . 463
gray_histo . . . 464
gray_histo_abs . . . 465
gray_histo_range . . . 466
gray_projections . . . 467
histo_2dim . . . 469
intensity . . . 470
min_max_gray . . . 470
moments_gray_plane . . . 472
plane_deviation . . . 473
select_gray . . . 474
shape_histo_all . . . 476
shape_histo_point . . . 477
6.8 Typanpassung . . . 478
complex_to_real . . . 478
convert_image_type . . . 478

real_to_complex . . . 479
real_to_vector_field . . . 480
vector_field_to_real . . . 480
6.9 Zugriff . . . 481
get_grayval . . . 481
get_grayval_contour_xld . . . 482
get_grayval_interpolated . . . 483
get_image_pointer1 . . . 484
get_image_pointer1_rect . . . 486
get_image_pointer3 . . . 487
get_image_size . . . 488
get_image_time . . . 488
get_image_type . . . 489

7 Datei **491**
7.1 Bilddaten . . . 491
deserialize_image . . . 491
read_image . . . 491
read_image_metadata . . . 493
read_sequence . . . 494
serialize_image . . . 497
write_image . . . 497
write_image_metadata . . . 500
7.2 Objekt . . . 501
deserialize_object . . . 501
read_object . . . 501
serialize_object . . . 502
write_object . . . 502
7.3 Region . . . 503
deserialize_region . . . 503
read_region . . . 504
serialize_region . . . 505
write_region . . . 505
7.4 Sonstiges . . . 506
copy_file . . . 506
delete_file . . . 507
file_exists . . . 507
get_current_dir . . . 508
list_files . . . 508
make_dir . . . 509
read_world_file . . . 509
remove_dir . . . 510
set_current_dir . . . 510
7.5 Tupel . . . 511
deserialize_handle . . . 511
deserialize_tuple . . . 512
read_tuple . . . 512
serialize_handle . . . 513
serialize_tuple . . . 513
tuple_is_serializable . . . 514
tuple_is_serializable_elem . . . 515
write_tuple . . . 515
7.6 XLD . . . 516
deserialize_xld . . . 516
read_contour_xld_arc_info . . . 517
read_contour_xld_dxf . . . 517
read_polygon_xld_arc_info . . . 519
read_polygon_xld_dxf . . . 520

serialize_xld . . . 521
write_contour_xld_arc_info . . . 522
write_contour_xld_dxf . . . 523
write_polygon_xld_arc_info . . . 525
write_polygon_xld_dxf . . . 526
7.7 Zugriff . . . 527
close_file . . . 527
fnew_line . . . 528
fread_bytes . . . 528
fread_char . . . 529
fread_line . . . 530
fread_string . . . 532
fwrite_bytes . . . 533
fwrite_string . . . 534
open_file . . . 534

8 Deep Learning **537**
get_dl_device_param . . . 547
optimize_dl_model_for_inference . . . 548
query_available_dl_devices . . . 549
set_dl_device_param . . . 551
8.1 Anomalieerkennung . . . 551
train_dl_model_anomaly_dataset . . . 556
8.2 Framework . . . 557
create_dl_layer_activation . . . 557
create_dl_layer_batch_normalization . . . 559
create_dl_layer_class_id_conversion . . . 562
create_dl_layer_concat . . . 565
create_dl_layer_convolution . . . 566
create_dl_layer_dense . . . 570
create_dl_layer_depth_max . . . 572
create_dl_layer_dropout . . . 573
create_dl_layer_elementwise . . . 575
create_dl_layer_input . . . 576
create_dl_layer_loss_cross_entropy . . . 579
create_dl_layer_loss_ctc . . . 581
create_dl_layer_loss_distance . . . 585
create_dl_layer_loss_focal . . . 587
create_dl_layer_loss_huber . . . 589
create_dl_layer_lrn . . . 591
create_dl_layer_pooling . . . 592
create_dl_layer_reshape . . . 595
create_dl_layer_softmax . . . 597
create_dl_layer_transposed_convolution . . . 598
create_dl_layer_zoom_factor . . . 601
create_dl_layer_zoom_size . . . 602
create_dl_layer_zoom_to_layer_size . . . 604
create_dl_model . . . 606
get_dl_layer_param . . . 607
get_dl_model_layer . . . 608
get_dl_model_layer_activations . . . 609
get_dl_model_layer_gradients . . . 609
get_dl_model_layer_param . . . 610
get_dl_model_layer_weights . . . 610
load_dl_model_weights . . . 613
set_dl_model_layer_param . . . 613
set_dl_model_layer_weights . . . 614
8.3 Klassifikation . . . 615

8.4 Modell . . . 619
add_dl_pruning_batch . . . 627
apply_dl_model . . . 628
clear_dl_model . . . 630
create_dl_pruning . . . 630
deserialize_dl_model . . . 631
gen_dl_model_heatmap . . . 632
gen_dl_pruned_model . . . 634
get_dl_model_param . . . 634
get_dl_pruning_param . . . 646
read_dl_model . . . 647
serialize_dl_model . . . 652
set_dl_model_param . . . 653
set_dl_pruning_param . . . 656
train_dl_model_batch . . . 657
write_dl_model . . . 659
8.5 Objektdetektion . . . 660
create_dl_model_detection . . . 668
8.6 Semantische Segmentierung . . . 671

9 Develop **681**
dev_clear_obj . . . 681
dev_clear_window . . . 681
dev_close_inspect_ctrl . . . 682
dev_close_tool . . . 683
dev_close_window . . . 683
dev_disp_text . . . 684
dev_display . . . 686
dev_error_var . . . 687
dev_get_exception_data . . . 687
dev_get_preferences . . . 689
dev_get_system . . . 689
dev_get_window . . . 690
dev_inspect_ctrl . . . 690
dev_open_dialog . . . 691
dev_open_file_dialog . . . 692
dev_open_tool . . . 693
dev_open_window . . . 697
dev_set_check . . . 700
dev_set_color . . . 701
dev_set_colored . . . 703
dev_set_contour_style . . . 704
dev_set_draw . . . 705
dev_set_line_width . . . 705
dev_set_lut . . . 706
dev_set_paint . . . 707
dev_set_part . . . 708
dev_set_preferences . . . 709
dev_set_shape . . . 710
dev_set_tool_geometry . . . 711
dev_set_window . . . 712
dev_set_window_extents . . . 712
dev_show_tool . . . 714

dev_update_pc . . . 715
dev_update_time . . . 715
dev_update_var . . . 716
dev_update_window . . . 716

10 Filter **719**

10.1 Arithmetik . . . 721
abs_diff_image . . . 721
abs_image . . . 722
acos_image . . . 723
add_image . . . 723
asin_image . . . 725
atan2_image . . . 725
atan_image . . . 726
cos_image . . . 726
div_image . . . 727
exp_image . . . 728
gamma_image . . . 729
invert_image . . . 730
log_image . . . 731
max_image . . . 732
min_image . . . 733
mult_image . . . 734
pow_image . . . 735
scale_image . . . 736
sin_image . . . 737
sqrt_image . . . 738
sub_image . . . 738
tan_image . . . 740

10.2 Bit . . . 741
bit_and . . . 741
bit_lshift . . . 742
bit_mask . . . 742
bit_not . . . 743
bit_or . . . 744
bit_rshift . . . 745
bit_slice . . . 746
bit_xor . . . 747

10.3 FFT . . . 748
convol_fft . . . 748
convol_gabor . . . 749
correlation_fft . . . 750
deserialize_fft_optimization_data . . . 750
energy_gabor . . . 751
fft_generic . . . 752
fft_image . . . 754
fft_image_inv . . . 755
gen_bandfilter . . . 756
gen_bandpass . . . 757
gen_derivative_filter . . . 758
gen_filter_mask . . . 760
gen_gabor . . . 761
gen_gauss_filter . . . 763
gen_highpass . . . 764
gen_lowpass . . . 765
gen_mean_filter . . . 766
gen_sin_bandpass . . . 768
gen_std_bandpass . . . 769

optimize_fft_speed . . . 770
optimize_rft_speed . . . 771
phase_correlation_fft . . . 772
phase_deg . . . 773
phase_rad . . . 774
power_byte . . . 775
power_ln . . . 776
power_real . . . 777
read_fft_optimization_data . . . 778
rft_generic . . . 779
serialize_fft_optimization_data . . . 780
write_fft_optimization_data . . . 781
10.4 Farbe . . . 781
apply_color_trans_lut . . . 781
cfa_to_rgb . . . 782
clear_color_trans_lut . . . 784
create_color_trans_lut . . . 784
gen_principal_comp_trans . . . 785
linear_trans_color . . . 786
principal_comp . . . 788
rgb1_to_gray . . . 788
rgb3_to_gray . . . 789
trans_from_rgb . . . 790
trans_to_rgb . . . 797
10.5 Geometrische Transformationen . . . 803
affine_trans_image . . . 803
affine_trans_image_size . . . 805
convert_map_type . . . 807
map_image . . . 809
mirror_image . . . 810
polar_trans_image_ext . . . 811
polar_trans_image_inv . . . 813
projective_trans_image . . . 815
projective_trans_image_size . . . 816
rotate_image . . . 818
zoom_image_factor . . . 819
zoom_image_size . . . 820
10.6 Glättung . . . 822
anisotropic_diffusion . . . 827
bilateral_filter . . . 829
binomial_filter . . . 833
eliminate_min_max . . . 834
eliminate_sp . . . 836
fill_interlace . . . 837
gauss_filter . . . 839
guided_filter . . . 840
info_smooth . . . 843
isotropic_diffusion . . . 844
mean_image . . . 845
mean_n . . . 847
mean_sp . . . 847
median_image . . . 849
median_rect . . . 851
median_separate . . . 853
median_weighted . . . 854
midrange_image . . . 855
rank_image . . . 856
rank_n . . . 858

rank_rect . . . 859
sigma_image . . . 861
smooth_image . . . 863
trimmed_mean . . . 864
10.7 Inpainting . . . 866
harmonic_interpolation . . . 866
inpainting_aniso . . . 867
inpainting_ced . . . 869
inpainting_ct . . . 871
inpainting_mcf . . . 874
inpainting_texture . . . 875
10.8 Kanten . . . 877
close_edges . . . 877
close_edges_length . . . 878
derivate_gauss . . . 879
diff_of_gauss . . . 883
edges_color . . . 884
edges_color_sub_pix . . . 887
edges_image . . . 889
edges_sub_pix . . . 892
frei_amp . . . 895
frei_dir . . . 896
highpass_image . . . 898
info_edges . . . 899
kirsch_amp . . . 900
kirsch_dir . . . 902
laplace . . . 903
laplace_of_gauss . . . 905
prewitt_amp . . . 906
prewitt_dir . . . 907
roberts . . . 908
robinson_amp . . . 909
robinson_dir . . . 911
sobel_amp . . . 912
sobel_dir . . . 914
10.9 Linien . . . 916
bandpass_image . . . 916
lines_color . . . 917
lines_facet . . . 919
lines_gauss . . . 921
10.10 Match . . . 925
exhaustive_match . . . 925
exhaustive_match_mg . . . 926
gen_gauss_pyramid . . . 928
monotony . . . 929
10.11 Optischer Fluss . . . 930
derivate_vector_field . . . 930
optical_flow_mg . . . 931
unwarp_image_vector_field . . . 940
vector_field_length . . . 941
10.12 Punkte . . . 941
corner_response . . . 941
dots_image . . . 943
points_foerstner . . . 944
points_harris . . . 947
points_harris_binomial . . . 949
points_lepetit . . . 950
points_sojka . . . 951

10.13 Rauschen . . . 953
add_noise_distribution . . . 953
add_noise_white . . . 954
gauss_distribution . . . 955
noise_distribution_mean . . . 956
sp_distribution . . . 956
10.14 Sonstiges . . . 957
convol_image . . . 957
deviation_n . . . 959
expand_domain_gray . . . 960
gray_inside . . . 962
gray_skeleton . . . 963
lut_trans . . . 964
symmetry . . . 964
topographic_sketch . . . 966
10.15 Szenenfluss . . . 967
scene_flow_calib . . . 967
scene_flow_uncalib . . . 969
10.16 Texturinspektion . . . 974
deviation_image . . . 974
entropy_image . . . 975
texture_laws . . . 976
10.17 Verbesserung . . . 978
coherence_enhancing_diff . . . 978
emphasize . . . 980
equ_histo_image . . . 981
illuminate . . . 982
mean_curvature_flow . . . 984
scale_image_max . . . 986
shock_filter . . . 987
10.18 Wienerfilter . . . 988
gen_psf_defocus . . . 988
gen_psf_motion . . . 989
simulate_defocus . . . 991
simulate_motion . . . 992
wiener_filter . . . 993
wiener_filter_ni . . . 995

11 Graphik **999**

11.1 3D-Szene . . . 999
add_scene_3d_camera . . . 999
add_scene_3d_instance . . . 1000
add_scene_3d_label . . . 1000
add_scene_3d_light . . . 1002
clear_scene_3d . . . 1003
create_scene_3d . . . 1003
display_scene_3d . . . 1005
get_display_scene_3d_info . . . 1006
remove_scene_3d_camera . . . 1007
remove_scene_3d_instance . . . 1007
remove_scene_3d_label . . . 1008
remove_scene_3d_light . . . 1008
render_scene_3d . . . 1009
set_scene_3d_camera_pose . . . 1009
set_scene_3d_instance_param . . . 1010
set_scene_3d_instance_pose . . . 1013
set_scene_3d_label_param . . . 1013
set_scene_3d_light_param . . . 1015

set_scene_3d_param . . . 1015
set_scene_3d_to_world_pose . . . 1017
11.2 Ausgabe . . . 1017
disp_arc . . . 1017
disp_arrow . . . 1019
disp_channel . . . 1020
disp_circle . . . 1021
disp_color . . . 1022
disp_cross . . . 1023
disp_ellipse . . . 1024
disp_image . . . 1026
disp_line . . . 1027
disp_obj . . . 1028
disp_object_model_3d . . . 1029
disp_polygon . . . 1033
disp_rectangle1 . . . 1034
disp_rectangle2 . . . 1036
disp_region . . . 1037
disp_xld . . . 1038
11.3 Farbtabelle . . . 1039
get_lut . . . 1039
query_lut . . . 1039
set_lut . . . 1040
11.4 Fenster . . . 1042
clear_window . . . 1042
close_window . . . 1043
copy_rectangle . . . 1044
dump_window . . . 1045
dump_window_image . . . 1047
flush_buffer . . . 1048
get_disp_object_model_3d_info . . . 1048
get_os_window_handle . . . 1049
get_window_attr . . . 1051
get_window_extents . . . 1052
get_window_pointer3 . . . 1053
get_window_type . . . 1054
new_extern_window . . . 1054
open_window . . . 1056
query_window_type . . . 1060
set_window_attr . . . 1061
set_window_dc . . . 1061
set_window_extents . . . 1062
set_window_type . . . 1063
unproject_coordinates . . . 1064
update_window_pose . . . 1065
11.5 Maus . . . 1067
get_mbutton . . . 1067
get_mbutton_sub_pix . . . 1068
get_mposition . . . 1068
get_mposition_sub_pix . . . 1069
get_mshape . . . 1070
query_mshape . . . 1071
send_mouse_double_click_event . . . 1072
send_mouse_down_event . . . 1073
send_mouse_drag_event . . . 1074
send_mouse_up_event . . . 1074
set_mshape . . . 1075
11.6 Objekt . . . 1076

attach_background_to_window . . . 1076
attach_drawing_object_to_window . . . 1077
clear_drawing_object . . . 1078
create_drawing_object_circle . . . 1079
create_drawing_object_circle_sector . . . 1080
create_drawing_object_ellipse . . . 1081
create_drawing_object_ellipse_sector . . . 1082
create_drawing_object_line . . . 1084
create_drawing_object_rectangle1 . . . 1085
create_drawing_object_rectangle2 . . . 1086
create_drawing_object_text . . . 1087
create_drawing_object_xld . . . 1088
detach_background_from_window . . . 1089
detach_drawing_object_from_window . . . 1090
get_drawing_object_iconic . . . 1091
get_drawing_object_params . . . 1091
get_window_background_image . . . 1092
set_content_update_callback . . . 1093
set_drawing_object_callback . . . 1094
set_drawing_object_params . . . 1095
set_drawing_object_xld . . . 1097
11.7 Parameter . . . 1098
convert_coordinates_image_to_window . . . 1098
convert_coordinates_window_to_image . . . 1099
get_contour_style . . . 1100
get_draw . . . 1100
get_hsi . . . 1101
get_icon . . . 1102
get_line_style . . . 1102
get_line_width . . . 1103
get_paint . . . 1103
get_part . . . 1104
get_part_style . . . 1105
get_rgb . . . 1105
get_rgba . . . 1106
get_shape . . . 1107
get_window_param . . . 1107
query_all_colors . . . 1108
query_color . . . 1109
query_colored . . . 1110
query_gray . . . 1110
query_line_width . . . 1111
query_paint . . . 1112
query_shape . . . 1112
set_color . . . 1113
set_colored . . . 1115
set_contour_style . . . 1116
set_draw . . . 1117
set_gray . . . 1117
set_hsi . . . 1118
set_icon . . . 1120
set_line_style . . . 1120
set_line_width . . . 1121
set_paint . . . 1122
set_part . . . 1125
set_part_style . . . 1126
set_rgb . . . 1127
set_rgba . . . 1128
set_shape . . . 1129

set_window_param . . . 1130
11.8 Text . . . 1132
disp_text . . . 1132
get_font . . . 1134
get_font_extents . . . 1135
get_string_extents . . . 1136
get_tposition . . . 1136
new_line . . . 1137
query_font . . . 1138
read_char . . . 1139
read_string . . . 1139
set_font . . . 1140
set_tposition . . . 1141
write_string . . . 1142
11.9 Zeichnen . . . 1143
drag_region1 . . . 1144
drag_region2 . . . 1145
drag_region3 . . . 1146
draw_circle . . . 1147
draw_circle_mod . . . 1148
draw_ellipse . . . 1149
draw_ellipse_mod . . . 1151
draw_line . . . 1152
draw_line_mod . . . 1153
draw_nurbs . . . 1154
draw_nurbs_interp . . . 1156
draw_nurbs_interp_mod . . . 1158
draw_nurbs_mod . . . 1161
draw_point . . . 1163
draw_point_mod . . . 1164
draw_polygon . . . 1165
draw_rectangle1 . . . 1166
draw_rectangle1_mod . . . 1167
draw_rectangle2 . . . 1168
draw_rectangle2_mod . . . 1169
draw_region . . . 1170
draw_xld . . . 1171
draw_xld_mod . . . 1172

12 Identifizierung **1175**
12.1 Barcode . . . 1175
clear_bar_code_model . . . 1177
create_bar_code_model . . . 1177
decode_bar_code_rectangle2 . . . 1179
deserialize_bar_code_model . . . 1181
find_bar_code . . . 1181
get_bar_code_object . . . 1185
get_bar_code_param . . . 1186
get_bar_code_param_specific . . . 1188
get_bar_code_result . . . 1189
query_bar_code_params . . . 1197
read_bar_code_model . . . 1199
serialize_bar_code_model . . . 1199
set_bar_code_param . . . 1200
set_bar_code_param_specific . . . 1209
write_bar_code_model . . . 1212
12.2 Datacode . . . 1213
clear_data_code_2d_model . . . 1215

create_data_code_2d_model . . . 1216
deserialize_data_code_2d_model . . . 1220
find_data_code_2d . . . 1220
get_data_code_2d_objects . . . 1227
get_data_code_2d_param . . . 1229
get_data_code_2d_results . . . 1233
query_data_code_2d_params . . . 1250
read_data_code_2d_model . . . 1251
serialize_data_code_2d_model . . . 1252
set_data_code_2d_param . . . 1253
write_data_code_2d_model . . . 1262
12.3 Samplebasiert . . . 1263
add_sample_identifier_preparation_data . . . 1266
add_sample_identifier_training_data . . . 1268
apply_sample_identifier . . . 1270
clear_sample_identifier . . . 1272
create_sample_identifier . . . 1273
deserialize_sample_identifier . . . 1274
get_sample_identifier_object_info . . . 1275
get_sample_identifier_param . . . 1277
prepare_sample_identifier . . . 1278
read_sample_identifier . . . 1279
remove_sample_identifier_preparation_data . . . 1280
remove_sample_identifier_training_data . . . 1281
serialize_sample_identifier . . . 1282
set_sample_identifier_object_info . . . 1283
set_sample_identifier_param . . . 1283
train_sample_identifier . . . 1285
write_sample_identifier . . . 1286

13 Inspektion **1289**
13.1 Kleberaupeninspektion . . . 1289
apply_bead_inspection_model . . . 1289
clear_bead_inspection_model . . . 1290
create_bead_inspection_model . . . 1291
get_bead_inspection_param . . . 1293
set_bead_inspection_param . . . 1294
13.2 OCV . . . 1295
close_ocv . . . 1295
create_ocv_proj . . . 1296
deserialize_ocv . . . 1297
do_ocv_simple . . . 1298
read_ocv . . . 1299
serialize_ocv . . . 1300
traind_ocv_proj . . . 1301
write_ocv . . . 1302
13.3 Strukturiertes Licht . . . 1302
clear_structured_light_model . . . 1303
create_structured_light_model . . . 1304
decode_structured_light_pattern . . . 1305
deserialize_structured_light_model . . . 1306
gen_structured_light_pattern . . . 1307
get_structured_light_model_param . . . 1311
get_structured_light_object . . . 1312
read_structured_light_model . . . 1314
serialize_structured_light_model . . . 1315
set_structured_light_model_param . . . 1316
write_structured_light_model . . . 1319

13.4 Texturinspektion . . . 1320
add_texture_inspection_model_image . . . 1324
apply_texture_inspection_model . . . 1325
clear_texture_inspection_model . . . 1327
clear_texture_inspection_result . . . 1328
create_texture_inspection_model . . . 1329
deserialize_texture_inspection_model . . . 1330
get_texture_inspection_model_image . . . 1331
get_texture_inspection_model_param . . . 1332
get_texture_inspection_result_object . . . 1334
read_texture_inspection_model . . . 1335
remove_texture_inspection_model_image . . . 1336
serialize_texture_inspection_model . . . 1338
set_texture_inspection_model_param . . . 1339
train_texture_inspection_model . . . 1343
write_texture_inspection_model . . . 1345
13.5 Variationsmodell . . . 1346
clear_train_data_variation_model . . . 1346
clear_variation_model . . . 1346
compare_ext_variation_model . . . 1347
compare_variation_model . . . 1348
create_variation_model . . . 1350
deserialize_variation_model . . . 1351
get_thresh_images_variation_model . . . 1352
get_variation_model . . . 1353
prepare_direct_variation_model . . . 1353
prepare_variation_model . . . 1355
read_variation_model . . . 1356
serialize_variation_model . . . 1357
train_variation_model . . . 1358
write_variation_model . . . 1359

14 Kalibrierung **1361**

14.1 Binokular . . . 1376
binocular_calibration . . . 1376
14.2 Hand-Auge . . . 1380
calibrate_hand_eye . . . 1380
get_calib_data_observ_pose . . . 1385
hand_eye_calibration . . . 1386
set_calib_data_observ_pose . . . 1391
14.3 Inverse Projektion . . . 1392
get_line_of_sight . . . 1392
14.4 Kalibrierkörper . . . 1393
caltab_points . . . 1393
create_caltab . . . 1395
disp_caltab . . . 1401
find_calib_object . . . 1402
find_caltab . . . 1405
find_marks_and_pose . . . 1406
gen_caltab . . . 1409
sim_caltab . . . 1412
14.5 Kameraparameter . . . 1414
cam_mat_to_cam_par . . . 1414
cam_par_to_cam_mat . . . 1415
deserialize_cam_par . . . 1416
read_cam_par . . . 1417
serialize_cam_par . . . 1418
write_cam_par . . . 1418

14.6 Mehrbild . . . 1419
calibrate_cameras . . . 1423
clear_calib_data . . . 1424
clear_camera_setup_model . . . 1424
create_calib_data . . . 1425
create_camera_setup_model . . . 1426
deserialize_calib_data . . . 1428
deserialize_camera_setup_model . . . 1428
get_calib_data . . . 1429
get_calib_data_observ_contours . . . 1436
get_calib_data_observ_points . . . 1437
get_camera_setup_param . . . 1438
query_calib_data_observ_indices . . . 1439
read_calib_data . . . 1441
read_camera_setup_model . . . 1441
remove_calib_data . . . 1442
remove_calib_data_observ . . . 1442
serialize_calib_data . . . 1443
serialize_camera_setup_model . . . 1444
set_calib_data . . . 1445
set_calib_data_calib_object . . . 1448
set_calib_data_cam_param . . . 1449
set_calib_data_observ_points . . . 1451
set_camera_setup_cam_param . . . 1453
set_camera_setup_param . . . 1454
write_calib_data . . . 1455
write_camera_setup_model . . . 1455
14.7 Monokular . . . 1456
camera_calibration . . . 1456
14.8 Projektion . . . 1460
cam_par_pose_to_hom_mat3d . . . 1460
project_3d_point . . . 1460
project_hom_point_hom_mat3d . . . 1461
project_point_hom_mat3d . . . 1462
14.9 Rektifizierung . . . 1463
change_radial_distortion_cam_par . . . 1463
change_radial_distortion_contours_xld . . . 1465
change_radial_distortion_image . . . 1466
change_radial_distortion_points . . . 1467
contour_to_world_plane_xld . . . 1467
gen_image_to_world_plane_map . . . 1469
gen_radial_distortion_map . . . 1471
image_points_to_world_plane . . . 1472
image_to_world_plane . . . 1474
14.10 Selbst-Kalibrierung . . . 1476
radial_distortion_self_calibration . . . 1476
radiometric_self_calibration . . . 1479
stationary_camera_self_calibration . . . 1482

15 Klassifikation 1489

15.1 Gaußsches Mischverteilungsmodell . . . 1489
add_class_train_data_gmm . . . 1489
add_sample_class_gmm . . . 1490
classify_class_gmm . . . 1491
clear_class_gmm . . . 1492
clear_samples_class_gmm . . . 1492
create_class_gmm . . . 1493
deserialize_class_gmm . . . 1496

evaluate_class_gmm . . . 1496
get_class_train_data_gmm . . . 1498
get_params_class_gmm . . . 1499
get_prep_info_class_gmm . . . 1500
get_sample_class_gmm . . . 1501
get_sample_num_class_gmm . . . 1502
read_class_gmm . . . 1503
read_samples_class_gmm . . . 1504
select_feature_set_gmm . . . 1505
serialize_class_gmm . . . 1507
train_class_gmm . . . 1508
write_class_gmm . . . 1510
write_samples_class_gmm . . . 1511
15.2 K-Nächste-Nachbarn . . . 1511
add_class_train_data_knn . . . 1511
add_sample_class_knn . . . 1512
classify_class_knn . . . 1513
clear_class_knn . . . 1514
create_class_knn . . . 1515
deserialize_class_knn . . . 1516
get_class_train_data_knn . . . 1516
get_params_class_knn . . . 1517
get_sample_class_knn . . . 1518
get_sample_num_class_knn . . . 1518
read_class_knn . . . 1519
select_feature_set_knn . . . 1520
serialize_class_knn . . . 1522
set_params_class_knn . . . 1523
train_class_knn . . . 1524
write_class_knn . . . 1525
15.3 Look-Up-Tabelle . . . 1526
clear_class_lut . . . 1526
create_class_lut_gmm . . . 1526
create_class_lut_knn . . . 1528
create_class_lut_mlp . . . 1530
create_class_lut_svm . . . 1531
15.4 Neuronale Netze . . . 1533
add_class_train_data_mlp . . . 1533
add_sample_class_mlp . . . 1534
classify_class_mlp . . . 1535
clear_class_mlp . . . 1536
clear_samples_class_mlp . . . 1536
create_class_mlp . . . 1537
deserialize_class_mlp . . . 1542
evaluate_class_mlp . . . 1542
get_class_train_data_mlp . . . 1543
get_params_class_mlp . . . 1544
get_prep_info_class_mlp . . . 1545
get_regularization_params_class_mlp . . . 1546
get_rejection_params_class_mlp . . . 1547
get_sample_class_mlp . . . 1548
get_sample_num_class_mlp . . . 1549
read_class_mlp . . . 1549
read_samples_class_mlp . . . 1550
select_feature_set_mlp . . . 1551
serialize_class_mlp . . . 1553
set_regularization_params_class_mlp . . . 1554
set_rejection_params_class_mlp . . . 1559

train_class_mlp . . . 1561
write_class_mlp . . . 1563
write_samples_class_mlp . . . 1564
15.5 Sonstiges . . . 1564
add_sample_class_train_data . . . 1564
clear_class_train_data . . . 1565
create_class_train_data . . . 1566
deserialize_class_train_data . . . 1567
get_sample_class_train_data . . . 1568
get_sample_num_class_train_data . . . 1568
read_class_train_data . . . 1569
select_sub_feature_class_train_data . . . 1570
serialize_class_train_data . . . 1571
set_feature_lengths_class_train_data . . . 1571
write_class_train_data . . . 1573
15.6 Support-Vektor-Maschinen . . . 1573
add_class_train_data_svm . . . 1573
add_sample_class_svm . . . 1574
classify_class_svm . . . 1575
clear_class_svm . . . 1576
clear_samples_class_svm . . . 1577
create_class_svm . . . 1577
deserialize_class_svm . . . 1582
evaluate_class_svm . . . 1582
get_class_train_data_svm . . . 1583
get_params_class_svm . . . 1584
get_prep_info_class_svm . . . 1585
get_sample_class_svm . . . 1586
get_sample_num_class_svm . . . 1587
get_support_vector_class_svm . . . 1588
get_support_vector_num_class_svm . . . 1589
read_class_svm . . . 1590
read_samples_class_svm . . . 1590
reduce_class_svm . . . 1591
select_feature_set_svm . . . 1593
serialize_class_svm . . . 1595
train_class_svm . . . 1595
write_class_svm . . . 1597
write_samples_class_svm . . . 1598

16 Legacy **1599**

16.1 2D-Messtechnik . . . 1599
copy_metrology_object . . . 1599
transform_metrology_object . . . 1600
16.2 DL Klassifikation . . . 1601
apply_dl_classifier . . . 1603
clear_dl_classifier . . . 1605
clear_dl_classifier_result . . . 1605
clear_dl_classifier_train_result . . . 1606
deserialize_dl_classifier . . . 1607
get_dl_classifier_param . . . 1607
get_dl_classifier_result . . . 1608
get_dl_classifier_train_result . . . 1609
read_dl_classifier . . . 1610
serialize_dl_classifier . . . 1612
set_dl_classifier_param . . . 1613
train_dl_classifier_batch . . . 1616
write_dl_classifier . . . 1618

16.3 Develop . . . 1619
dev_map_par . . . 1619
dev_map_prog . . . 1620
dev_map_var . . . 1620
dev_unmap_par . . . 1620
dev_unmap_prog . . . 1621
dev_unmap_var . . . 1621
16.4 Filter . . . 1622
gauss_image . . . 1622
polar_trans_image . . . 1623
16.5 Graphik . . . 1624
clear_rectangle . . . 1624
disp_distribution . . . 1626
disp_lut . . . 1627
get_comprise . . . 1628
get_fix . . . 1628
get_fixed_lut . . . 1629
get_insert . . . 1629
get_line_approx . . . 1630
get_lut_style . . . 1630
get_pixel . . . 1631
get_tshape . . . 1632
gnuplot_close . . . 1632
gnuplot_open_file . . . 1633
gnuplot_open_pipe . . . 1634
gnuplot_plot_ctrl . . . 1634
gnuplot_plot_funct_1d . . . 1635
gnuplot_plot_image . . . 1636
move_rectangle . . . 1637
open_textwindow . . . 1639
query_insert . . . 1642
query_tshape . . . 1643
set_comprise . . . 1644
set_fix . . . 1645
set_fixed_lut . . . 1645
set_insert . . . 1646
set_line_approx . . . 1647
set_lut_style . . . 1648
set_pixel . . . 1649
set_tshape . . . 1650
slide_image . . . 1651
write_lut . . . 1652
16.6 Klassifikation . . . 1652
clear_sampset . . . 1652
close_class_box . . . 1653
create_class_box . . . 1653
descript_class_box . . . 1654
deserialize_class_box . . . 1655
enquire_class_box . . . 1656
enquire_reject_class_box . . . 1657
get_class_box_param . . . 1658
learn_class_box . . . 1658
learn_sampset_box . . . 1659
read_class_box . . . 1660
read_sampset . . . 1661
serialize_class_box . . . 1662
set_class_box_param . . . 1663
test_sampset_box . . . 1664

write_class_box . . . 1665
16.7 Matching . . . 1665
adapt_template . . . 1665
best_match . . . 1666
best_match_mg . . . 1667
best_match_pre_mg . . . 1669
best_match_rot . . . 1670
best_match_rot_mg . . . 1671
clear_template . . . 1673
create_template . . . 1673
create_template_rot . . . 1675
deserialize_template . . . 1677
fast_match . . . 1677
fast_match_mg . . . 1678
read_template . . . 1679
serialize_template . . . 1680
set_offset_template . . . 1681
set_reference_template . . . 1682
write_template . . . 1682
16.8 Morphologie . . . 1683
closing_golay . . . 1683
dilation_golay . . . 1684
dilation_seq . . . 1686
erosion_golay . . . 1687
erosion_seq . . . 1688
fitting . . . 1689
gen_struct_elements . . . 1690
golay_elements . . . 1691
hit_or_miss_golay . . . 1695
hit_or_miss_seq . . . 1696
morph_hat . . . 1697
morph_skeleton . . . 1698
morph_skiz . . . 1699
opening_golay . . . 1700
opening_seg . . . 1702
thickening . . . 1703
thickening_golay . . . 1704
thickening_seq . . . 1706
thinning . . . 1707
thinning_golay . . . 1708
thinning_seq . . . 1709
16.9 OCR . . . 1711
close_ocr . . . 1711
create_ocr_class_box . . . 1712
create_text_model . . . 1714
deserialize_ocr . . . 1715
do_ocr_multi . . . 1715
do_ocr_single . . . 1716
info_ocr_class_box . . . 1717
ocr_change_char . . . 1718
ocr_get_features . . . 1719
read_ocr . . . 1719
serialize_ocr . . . 1720
testd_ocr_class_box . . . 1721
traind_ocr_class_box . . . 1721
trainf_ocr_class_box . . . 1722
write_ocr . . . 1723
16.10 Regionen . . . 1724

get_region_chain . . . 1724
hamming_change_region . . . 1725
interjacent . . . 1726
16.11 Segmentation . . . 1728
bin_threshold . . . 1728
class_ndim_box . . . 1728
expand_line . . . 1730
learn_ndim_box . . . 1731
16.12 Steuerung . . . 1732
ifelse . . . 1732
16.13 Tools . . . 1733
approx_chain . . . 1733
approx_chain_simple . . . 1736
clear_all_bar_code_models . . . 1738
clear_all_barriers . . . 1738
clear_all_calib_data . . . 1739
clear_all_camera_setup_models . . . 1739
clear_all_class_gmm . . . 1740
clear_all_class_knn . . . 1740
clear_all_class_lut . . . 1741
clear_all_class_mlp . . . 1741
clear_all_class_svm . . . 1742
clear_all_class_train_data . . . 1742
clear_all_color_trans_luts . . . 1742
clear_all_component_models . . . 1743
clear_all_conditions . . . 1743
clear_all_data_code_2d_models . . . 1744
clear_all_deformable_models . . . 1744
clear_all_descriptor_models . . . 1745
clear_all_events . . . 1745
clear_all_lexica . . . 1746
clear_all_matrices . . . 1746
clear_all_metrology_models . . . 1747
clear_all_mutexes . . . 1747
clear_all_ncc_models . . . 1747
clear_all_object_model_3d . . . 1748
clear_all_ocr_class_knn . . . 1748
clear_all_ocr_class_mlp . . . 1749
clear_all_ocr_class_svm . . . 1749
clear_all_sample_identifiers . . . 1750
clear_all_scattered_data_interpolators . . . 1750
clear_all_serialized_items . . . 1751
clear_all_shape_model_3d . . . 1751
clear_all_shape_models . . . 1752
clear_all_sheet_of_light_models . . . 1752
clear_all_stereo_models . . . 1753
clear_all_surface_matching_results . . . 1753
clear_all_surface_models . . . 1754
clear_all_templates . . . 1754
clear_all_text_models . . . 1754
clear_all_text_results . . . 1755
clear_all_training_components . . . 1755
clear_all_variation_models . . . 1756
close_all_bg_esti . . . 1756
close_all_class_box . . . 1757
close_all_files . . . 1757
close_all_framegrabbers . . . 1758
close_all_measures . . . 1758

close_all_ocrs . . . 1759
close_all_ocvs . . . 1759
close_all_serials . . . 1760
close_all_sockets . . . 1760
filter_kalman . . . 1760
intersection_ll . . . 1764
partition_lines . . . 1766
read_kalman . . . 1767
select_lines . . . 1770
select_lines_longest . . . 1771
update_kalman . . . 1772
16.14 XLD . . . 1774
union_straight_contours_histo_xld . . . 1774

17 Matching **1777**
17.1 Deformierbar . . . 1777
clear_deformable_model . . . 1777
create_local_deformable_model . . . 1777
create_local_deformable_model_xld . . . 1780
create_planar_calib_deformable_model . . . 1782
create_planar_calib_deformable_model_xld . . . 1784
create_planar_uncalib_deformable_model . . . 1787
create_planar_uncalib_deformable_model_xld . . . 1791
deserialize_deformable_model . . . 1794
determine_deformable_model_params . . . 1795
find_local_deformable_model . . . 1798
find_planar_calib_deformable_model . . . 1801
find_planar_uncalib_deformable_model . . . 1803
get_deformable_model_contours . . . 1807
get_deformable_model_origin . . . 1808
get_deformable_model_params . . . 1809
read_deformable_model . . . 1810
serialize_deformable_model . . . 1811
set_deformable_model_origin . . . 1812
set_deformable_model_param . . . 1813
set_local_deformable_model_metric . . . 1814
set_planar_calib_deformable_model_metric . . . 1815
set_planar_uncalib_deformable_model_metric . . . 1816
write_deformable_model . . . 1818
17.2 Deskriptorbasiert . . . 1818
clear_descriptor_model . . . 1818
create_calib_descriptor_model . . . 1819
create_uncalib_descriptor_model . . . 1820
deserialize_descriptor_model . . . 1824
find_calib_descriptor_model . . . 1824
find_uncalib_descriptor_model . . . 1826
get_descriptor_model_origin . . . 1829
get_descriptor_model_params . . . 1829
get_descriptor_model_points . . . 1830
get_descriptor_model_results . . . 1831
read_descriptor_model . . . 1832
serialize_descriptor_model . . . 1833
set_descriptor_model_origin . . . 1834
write_descriptor_model . . . 1834
17.3 Formbasiert . . . 1835
adapt_shape_model_high_noise . . . 1835
clear_shape_model . . . 1836
create_aniso_shape_model . . . 1837

create_aniso_shape_model_xld . . . 1842
create_generic_shape_model . . . 1847
create_scaled_shape_model . . . 1847
create_scaled_shape_model_xld . . . 1852
create_shape_model . . . 1856
create_shape_model_xld . . . 1861
deserialize_shape_model . . . 1865
determine_shape_model_params . . . 1865
find_aniso_shape_model . . . 1868
find_aniso_shape_models . . . 1874
find_generic_shape_model . . . 1882
find_scaled_shape_model . . . 1883
find_scaled_shape_models . . . 1889
find_shape_model . . . 1897
find_shape_models . . . 1902
get_generic_shape_model_param . . . 1909
get_generic_shape_model_result . . . 1910
get_generic_shape_model_result_object . . . 1912
get_shape_model_clutter . . . 1913
get_shape_model_contours . . . 1914
get_shape_model_origin . . . 1914
get_shape_model_params . . . 1915
inspect_shape_model . . . 1916
read_shape_model . . . 1918
serialize_shape_model . . . 1918
set_generic_shape_model_object . . . 1919
set_generic_shape_model_param . . . 1920
set_shape_model_clutter . . . 1930
set_shape_model_metric . . . 1933
set_shape_model_origin . . . 1935
set_shape_model_param . . . 1936
train_generic_shape_model . . . 1937
write_shape_model . . . 1938

17.4 Komponentenbasiert . . . 1939
clear_component_model . . . 1939
clear_training_components . . . 1939
cluster_model_components . . . 1940
create_component_model . . . 1942
create_trained_component_model . . . 1946
deserialize_component_model . . . 1949
deserialize_training_components . . . 1949
find_component_model . . . 1950
gen_initial_components . . . 1956
get_component_model_params . . . 1959
get_component_model_tree . . . 1960
get_component_relations . . . 1962
get_found_component_model . . . 1964
get_training_components . . . 1966
inspect_clustered_components . . . 1968
modify_component_relations . . . 1970
read_component_model . . . 1972
read_training_components . . . 1972
serialize_component_model . . . 1973
serialize_training_components . . . 1974
train_model_components . . . 1974
write_component_model . . . 1979
write_training_components . . . 1980

17.5 Korrelationsbasiert . . . 1980
clear_ncc_model . . . 1980

create_ncc_model . . . 1981
deserialize_ncc_model . . . 1983
determine_ncc_model_params . . . 1983
find_ncc_model . . . 1985
find_ncc_models . . . 1989
get_ncc_model_origin . . . 1994
get_ncc_model_params . . . 1994
get_ncc_model_region . . . 1995
read_ncc_model . . . 1996
serialize_ncc_model . . . 1996
set_ncc_model_origin . . . 1997
set_ncc_model_param . . . 1997
write_ncc_model . . . 1998

18 Matrix **2001**

18.1 Arithmetik . . . 2001
abs_matrix . . . 2001
abs_matrix_mod . . . 2002
add_matrix . . . 2002
add_matrix_mod . . . 2003
div_element_matrix . . . 2004
div_element_matrix_mod . . . 2005
invert_matrix . . . 2006
invert_matrix_mod . . . 2008
mult_element_matrix . . . 2010
mult_element_matrix_mod . . . 2011
mult_matrix . . . 2012
mult_matrix_mod . . . 2014
pow_element_matrix . . . 2016
pow_element_matrix_mod . . . 2017
pow_matrix . . . 2018
pow_matrix_mod . . . 2019
pow_scalar_element_matrix . . . 2020
pow_scalar_element_matrix_mod . . . 2021
scale_matrix . . . 2022
scale_matrix_mod . . . 2023
solve_matrix . . . 2024
sqrt_matrix . . . 2026
sqrt_matrix_mod . . . 2027
sub_matrix . . . 2028
sub_matrix_mod . . . 2029
transpose_matrix . . . 2030
transpose_matrix_mod . . . 2031
18.2 Datei . . . 2031
deserialize_matrix . . . 2031
read_matrix . . . 2032
serialize_matrix . . . 2033
write_matrix . . . 2033
18.3 Eigenwerte . . . 2034
eigenvalues_general_matrix . . . 2034
eigenvalues_symmetric_matrix . . . 2035
generalized_eigenvalues_general_matrix . . . 2036
generalized_eigenvalues_symmetric_matrix . . . 2038
18.4 Generierung . . . 2039
clear_matrix . . . 2039
copy_matrix . . . 2040
create_matrix . . . 2041
repeat_matrix . . . 2042

18.5 Merkmale . . . 2043
determinant_matrix . . . 2043
get_size_matrix . . . 2044
max_matrix . . . 2045
mean_matrix . . . 2046
min_matrix . . . 2047
norm_matrix . . . 2048
sum_matrix . . . 2050
18.6 Zerlegung . . . 2051
decompose_matrix . . . 2051
orthogonal_decompose_matrix . . . 2053
svd_matrix . . . 2056
18.7 Zugriff . . . 2058
get_diagonal_matrix . . . 2058
get_full_matrix . . . 2059
get_sub_matrix . . . 2060
get_value_matrix . . . 2061
set_diagonal_matrix . . . 2062
set_full_matrix . . . 2064
set_sub_matrix . . . 2065
set_value_matrix . . . 2066

19 Morphologie **2069**
19.1 Grauwerte . . . 2069
dual_rank . . . 2071
gen_disc_se . . . 2073
gray_bothat . . . 2074
gray_closing . . . 2075
gray_closing_rect . . . 2076
gray_closing_shape . . . 2077
gray_dilation . . . 2078
gray_dilation_rect . . . 2079
gray_dilation_shape . . . 2080
gray_erosion . . . 2081
gray_erosion_rect . . . 2082
gray_erosion_shape . . . 2083
gray_opening . . . 2084
gray_opening_rect . . . 2085
gray_opening_shape . . . 2086
gray_range_rect . . . 2088
gray_tophat . . . 2089
read_gray_se . . . 2089
19.2 Region . . . 2091
bottom_hat . . . 2093
boundary . . . 2094
closing . . . 2096
closing_circle . . . 2097
closing_rectangle1 . . . 2099
dilation1 . . . 2100
dilation2 . . . 2101
dilation_circle . . . 2103
dilation_rectangle1 . . . 2104
erosion1 . . . 2106
erosion2 . . . 2107
erosion_circle . . . 2109
erosion_rectangle1 . . . 2110
hit_or_miss . . . 2112
minkowski_add1 . . . 2113

minkowski_add2 . . . 2114
minkowski_sub1 . . . 2116
minkowski_sub2 . . . 2117
opening . . . 2119
opening_circle . . . 2120
opening_rectangle1 . . . 2121
pruning . . . 2122
top_hat . . . 2123

20 OCR **2125**

20.1 Convolutional Neural Networks . . . 2125
clear_ocr_class_cnn . . . 2125
deserialize_ocr_class_cnn . . . 2125
do_ocr_multi_class_cnn . . . 2126
do_ocr_single_class_cnn . . . 2127
do_ocr_word_cnn . . . 2128
get_params_ocr_class_cnn . . . 2130
query_params_ocr_class_cnn . . . 2131
read_ocr_class_cnn . . . 2131
serialize_ocr_class_cnn . . . 2132

20.2 Deep OCR . . . 2133
apply_deep_ocr . . . 2133
create_deep_ocr . . . 2135
get_deep_ocr_param . . . 2136
read_deep_ocr . . . 2139
set_deep_ocr_param . . . 2140
write_deep_ocr . . . 2141

20.3 K-Nächste-Nachbarn . . . 2142
clear_ocr_class_knn . . . 2142
create_ocr_class_knn . . . 2142
deserialize_ocr_class_knn . . . 2146
do_ocr_multi_class_knn . . . 2146
do_ocr_single_class_knn . . . 2147
do_ocr_word_knn . . . 2148
get_features_ocr_class_knn . . . 2150
get_params_ocr_class_knn . . . 2151
read_ocr_class_knn . . . 2152
select_feature_set_trainf_knn . . . 2152
serialize_ocr_class_knn . . . 2154
trainf_ocr_class_knn . . . 2155
write_ocr_class_knn . . . 2156

20.4 Lexika . . . 2157
clear_lexicon . . . 2157
create_lexicon . . . 2157
import_lexicon . . . 2158
inspect_lexicon . . . 2159
lookup_lexicon . . . 2159
suggest_lexicon . . . 2160

20.5 Neuronale Netze . . . 2161
clear_ocr_class_mlp . . . 2161
create_ocr_class_mlp . . . 2161
deserialize_ocr_class_mlp . . . 2165
do_ocr_multi_class_mlp . . . 2166
do_ocr_single_class_mlp . . . 2167
do_ocr_word_mlp . . . 2168
get_features_ocr_class_mlp . . . 2170
get_params_ocr_class_mlp . . . 2171
get_prep_info_ocr_class_mlp . . . 2171

get_regularization_params_ocr_class_mlp . . . 2173
get_rejection_params_ocr_class_mlp . . . 2174
read_ocr_class_mlp . . . 2174
select_feature_set_trainf_mlp . . . 2175
select_feature_set_trainf_mlp_protected . . . 2177
serialize_ocr_class_mlp . . . 2179
set_regularization_params_ocr_class_mlp . . . 2179
set_rejection_params_ocr_class_mlp . . . 2182
trainf_ocr_class_mlp . . . 2182
trainf_ocr_class_mlp_protected . . . 2184
write_ocr_class_mlp . . . 2185
20.6 Segmentation . . . 2186
clear_text_model . . . 2186
clear_text_result . . . 2187
create_text_model_reader . . . 2187
find_text . . . 2189
get_text_model_param . . . 2190
get_text_object . . . 2191
get_text_result . . . 2192
segment_characters . . . 2194
select_characters . . . 2196
set_text_model_param . . . 2200
text_line_orientation . . . 2205
text_line_slant . . . 2206
20.7 Support-Vektor-Maschinen . . . 2207
clear_ocr_class_svm . . . 2207
create_ocr_class_svm . . . 2208
deserialize_ocr_class_svm . . . 2211
do_ocr_multi_class_svm . . . 2212
do_ocr_single_class_svm . . . 2213
do_ocr_word_svm . . . 2213
get_features_ocr_class_svm . . . 2215
get_params_ocr_class_svm . . . 2216
get_prep_info_ocr_class_svm . . . 2217
get_support_vector_num_ocr_class_svm . . . 2219
get_support_vector_ocr_class_svm . . . 2219
read_ocr_class_svm . . . 2220
reduce_ocr_class_svm . . . 2221
select_feature_set_trainf_svm . . . 2222
select_feature_set_trainf_svm_protected . . . 2223
serialize_ocr_class_svm . . . 2225
trainf_ocr_class_svm . . . 2226
trainf_ocr_class_svm_protected . . . 2227
write_ocr_class_svm . . . 2228
20.8 Trainingsdateien . . . 2229
append_ocr_trainf . . . 2229
concat_ocr_trainf . . . 2230
protect_ocr_trainf . . . 2231
read_ocr_trainf . . . 2232
read_ocr_trainf_names . . . 2232
read_ocr_trainf_names_protected . . . 2233
read_ocr_trainf_select . . . 2234
write_ocr_trainf . . . 2235
write_ocr_trainf_image . . . 2235

21 Objekt **2237**
21.1 Information . . . 2238
compare_obj . . . 2238

count_obj . . . 2239
get_channel_info . . . 2239
get_obj_class . . . 2240
test_equal_obj . . . 2241
21.2 Manipulation . . . 2242
clear_obj . . . 2242
concat_obj . . . 2243
copy_obj . . . 2244
gen_empty_obj . . . 2245
insert_obj . . . 2245
integer_to_obj . . . 2246
obj_diff . . . 2247
obj_to_integer . . . 2248
remove_obj . . . 2249
replace_obj . . . 2249
select_obj . . . 2250

22 Regionen **2253**
22.1 Generierung . . . 2253
gen_checker_region . . . 2253
gen_circle . . . 2254
gen_circle_sector . . . 2256
gen_ellipse . . . 2258
gen_ellipse_sector . . . 2259
gen_empty_region . . . 2261
gen_grid_region . . . 2261
gen_random_region . . . 2263
gen_random_regions . . . 2264
gen_rectangle1 . . . 2266
gen_rectangle2 . . . 2267
gen_region_contour_xld . . . 2268
gen_region_histo . . . 2269
gen_region_hline . . . 2270
gen_region_line . . . 2271
gen_region_points . . . 2272
gen_region_polygon . . . 2273
gen_region_polygon_filled . . . 2274
gen_region_polygon_xld . . . 2275
gen_region_runs . . . 2276
label_to_region . . . 2277
22.2 Geometrische Transformationen . . . 2278
affine_trans_region . . . 2278
mirror_region . . . 2280
move_region . . . 2281
polar_trans_region . . . 2281
polar_trans_region_inv . . . 2284
projective_trans_region . . . 2286
transpose_region . . . 2287
zoom_region . . . 2288
22.3 Mengen-Operationen . . . 2289
complement . . . 2289
difference . . . 2290
intersection . . . 2291
symm_difference . . . 2291
union1 . . . 2292
union2 . . . 2293
22.4 Merkmale . . . 2294
area_center . . . 2300

area_holes . . . 2302
circularity . . . 2302
compactness . . . 2304
connect_and_holes . . . 2305
contlength . . . 2305
convexity . . . 2307
diameter_region . . . 2308
eccentricity . . . 2309
elliptic_axis . . . 2310
euler_number . . . 2311
find_neighbors . . . 2312
get_region_index . . . 2313
get_region_thickness . . . 2314
hamming_distance . . . 2315
hamming_distance_norm . . . 2316
height_width_ratio . . . 2317
inner_circle . . . 2318
inner_rectangle1 . . . 2319
moments_region_2nd . . . 2320
moments_region_2nd_invar . . . 2321
moments_region_2nd_rel_invar . . . 2322
moments_region_3rd . . . 2323
moments_region_3rd_invar . . . 2324
moments_region_central . . . 2325
moments_region_central_invar . . . 2326
orientation_region . . . 2327
rectangularity . . . 2328
region_features . . . 2329
roundness . . . 2332
runlength_distribution . . . 2333
runlength_features . . . 2334
select_region_point . . . 2335
select_region_spatial . . . 2336
select_shape . . . 2337
select_shape_proto . . . 2340
select_shape_std . . . 2342
smallest_circle . . . 2343
smallest_rectangle1 . . . 2345
smallest_rectangle2 . . . 2346
spatial_relation . . . 2347
22.5 Tests . . . 2349
test_equal_region . . . 2349
test_region_point . . . 2350
test_subset_region . . . 2350
22.6 Transformationen . . . 2351
background_seg . . . 2351
clip_region . . . 2352
clip_region_rel . . . 2353
closest_point_transform . . . 2355
connection . . . 2357
distance_transform . . . 2358
eliminate_runs . . . 2359
expand_region . . . 2360
fill_up . . . 2362
fill_up_shape . . . 2362
junctions_skeleton . . . 2363
merge_regions_line_scan . . . 2364
partition_dynamic . . . 2366
partition_rectangle . . . 2367

rank_region . . . 2367
remove_noise_region . . . 2369
shape_trans . . . 2370
skeleton . . . 2371
sort_region . . . 2372
split_skeleton_lines . . . 2373
split_skeleton_region . . . 2374
22.7 Zugriff . . . 2375
get_region_contour . . . 2375
get_region_convex . . . 2376
get_region_points . . . 2377
get_region_polygon . . . 2378
get_region_runs . . . 2378

23 Segmentation **2381**
23.1 Flächenwachstum . . . 2381
expand_gray . . . 2381
expand_gray_ref . . . 2383
regiongrowing . . . 2385
regiongrowing_mean . . . 2387
regiongrowing_n . . . 2388
23.2 Kanten . . . 2392
detect_edge_segments . . . 2392
hysteresis_threshold . . . 2394
nonmax_suppression_amp . . . 2395
nonmax_suppression_dir . . . 2396
23.3 Klassifikation . . . 2397
add_samples_image_class_gmm . . . 2397
add_samples_image_class_knn . . . 2398
add_samples_image_class_mlp . . . 2399
add_samples_image_class_svm . . . 2400
class_2dim_sup . . . 2401
class_2dim_unsup . . . 2403
class_ndim_norm . . . 2405
classify_image_class_gmm . . . 2407
classify_image_class_knn . . . 2408
classify_image_class_lut . . . 2409
classify_image_class_mlp . . . 2410
classify_image_class_svm . . . 2412
learn_ndim_norm . . . 2413
23.4 Maximally Stable Extremal Regions . . . 2414
segment_image_mser . . . 2414
23.5 Schwellenwert-Verfahren . . . 2420
auto_threshold . . . 2426
binary_threshold . . . 2427
char_threshold . . . 2429
check_difference . . . 2430
dual_threshold . . . 2431
dyn_threshold . . . 2433
fast_threshold . . . 2435
histo_to_thresh . . . 2436
local_threshold . . . 2437
threshold . . . 2439
threshold_sub_pix . . . 2440
var_threshold . . . 2441
zero_crossing . . . 2446
zero_crossing_sub_pix . . . 2447
23.6 Topographie . . . 2448

critical_points_sub_pix . . . 2448
local_max . . . 2450
local_max_sub_pix . . . 2451
local_min . . . 2452
local_min_sub_pix . . . 2453
lowlands . . . 2454
lowlands_center . . . 2455
plateaus . . . 2456
plateaus_center . . . 2458
pouring . . . 2459
saddle_points_sub_pix . . . 2461
watersheds . . . 2462
watersheds_marker . . . 2463
watersheds_threshold . . . 2465

24 Steuerung **2467**
assign . . . 2467
assign_at . . . 2468
break . . . 2468
case . . . 2469
catch . . . 2469
comment . . . 2471
continue . . . 2471
convert_tuple_to_vector_1d . . . 2471
convert_vector_to_tuple . . . 2472
default . . . 2472
else . . . 2473
elseif . . . 2473
endfor . . . 2473
endif . . . 2474
endswitch . . . 2474
endtry . . . 2474
endwhile . . . 2475
executable_expression . . . 2475
exit . . . 2475
export_def . . . 2476
for . . . 2477
global . . . 2479
if . . . 2480
import . . . 2480
insert . . . 2481
par_join . . . 2482
repeat . . . 2483
return . . . 2483
stop . . . 2483
switch . . . 2484
throw . . . 2485
try . . . 2486
until . . . 2487
while . . . 2488

25 System **2491**
25.1 Betriebssystem . . . 2491

count_seconds . . . 2491
get_system_time . . . 2491
system_call . . . 2492
wait_seconds . . . 2493
25.2 Compute-Devices . . . 2493
activate_compute_device . . . 2493
deactivate_all_compute_devices . . . 2494
deactivate_compute_device . . . 2494
get_compute_device_info . . . 2495
get_compute_device_param . . . 2496
init_compute_device . . . 2497
open_compute_device . . . 2497
query_available_compute_devices . . . 2498
release_all_compute_devices . . . 2499
release_compute_device . . . 2500
set_compute_device_param . . . 2500
25.3 Datenbank . . . 2502
count_relation . . . 2502
get_modules . . . 2504
reset_obj_db . . . 2504
25.4 Fehlerbehandlung . . . 2505
get_check . . . 2505
get_error_text . . . 2506
get_extended_error_info . . . 2506
get_spy . . . 2507
query_spy . . . 2508
set_check . . . 2508
set_spy . . . 2510
25.5 I/O-Geräte . . . 2511
close_io_channel . . . 2511
close_io_device . . . 2512
control_io_channel . . . 2513
control_io_device . . . 2513
control_io_interface . . . 2514
get_io_channel_param . . . 2515
get_io_device_param . . . 2516
open_io_channel . . . 2516
open_io_device . . . 2517
query_io_device . . . 2519
query_io_interface . . . 2520
read_io_channel . . . 2521
set_io_channel_param . . . 2521
set_io_device_param . . . 2522
write_io_channel . . . 2523
25.6 Information . . . 2524
get_chapter_info . . . 2524
get_keywords . . . 2524
get_operator_info . . . 2525
get_operator_name . . . 2527
get_param_info . . . 2527
get_param_names . . . 2529
get_param_num . . . 2530
get_param_types . . . 2531
query_operator_info . . . 2532
query_param_info . . . 2532
search_operator . . . 2533
25.7 Multithreading . . . 2533
broadcast_condition . . . 2533

clear_barrier . . . 2534
clear_condition . . . 2534
clear_event . . . 2535
clear_message . . . 2536
clear_message_queue . . . 2536
clear_mutex . . . 2537
create_barrier . . . 2538
create_condition . . . 2539
create_event . . . 2540
create_message . . . 2541
create_message_queue . . . 2542
create_mutex . . . 2543
dequeue_message . . . 2544
enqueue_message . . . 2545
get_current_hthread_id . . . 2547
get_message_obj . . . 2547
get_message_param . . . 2548
get_message_queue_param . . . 2550
get_message_tuple . . . 2551
get_threading_attrib . . . 2552
interrupt_operator . . . 2552
lock_mutex . . . 2554
read_message . . . 2554
set_message_obj . . . 2555
set_message_param . . . 2556
set_message_queue_param . . . 2557
set_message_tuple . . . 2559
signal_condition . . . 2560
signal_event . . . 2561
timed_wait_condition . . . 2561
try_lock_mutex . . . 2562
try_wait_event . . . 2562
unlock_mutex . . . 2563
wait_barrier . . . 2563
wait_condition . . . 2564
wait_event . . . 2564
write_message . . . 2565
25.8 Parallelisierung . . . 2566
get_aop_info . . . 2566
optimize_aop . . . 2567
query_aop_info . . . 2570
read_aop_knowledge . . . 2571
set_aop_info . . . 2572
write_aop_knowledge . . . 2573
25.9 Parameter . . . 2575
get_system . . . 2575
get_system_info . . . 2579
set_operator_timeout . . . 2579
set_system . . . 2580
25.10 Serialisiertes Element . . . 2595
clear_serialized_item . . . 2595
create_serialized_item_ptr . . . 2596
fread_serialized_item . . . 2597
fwrite_serialized_item . . . 2597
get_serialized_item_ptr . . . 2598
25.11 Seriell . . . 2599
clear_serial . . . 2599
close_serial . . . 2600

get_serial_param . . . 2600
open_serial . . . 2601
read_serial . . . 2602
set_serial_param . . . 2603
write_serial . . . 2604
25.12 Sockets . . . 2605
close_socket . . . 2605
get_next_socket_data_type . . . 2606
get_socket_descriptor . . . 2606
get_socket_param . . . 2607
open_socket_accept . . . 2607
open_socket_connect . . . 2610
receive_data . . . 2611
receive_image . . . 2612
receive_region . . . 2612
receive_serialized_item . . . 2613
receive_tuple . . . 2614
receive_xld . . . 2614
send_data . . . 2615
send_image . . . 2616
send_region . . . 2617
send_serialized_item . . . 2617
send_tuple . . . 2618
send_xld . . . 2619
set_socket_param . . . 2620
socket_accept_connect . . . 2620

26 Tools **2623**

26.1 Funktion . . . 2623
abs_funct_1d . . . 2623
compose_funct_1d . . . 2623
create_funct_1d_array . . . 2624
create_funct_1d_pairs . . . 2625
derivate_funct_1d . . . 2626
distance_funct_1d . . . 2626
funct_1d_to_pairs . . . 2627
get_pair_funct_1d . . . 2627
get_y_value_funct_1d . . . 2628
integrate_funct_1d . . . 2629
invert_funct_1d . . . 2629
local_min_max_funct_1d . . . 2630
match_funct_1d_trans . . . 2630
negate_funct_1d . . . 2632
num_points_funct_1d . . . 2632
read_funct_1d . . . 2632
sample_funct_1d . . . 2633
scale_y_funct_1d . . . 2634
smooth_funct_1d_gauss . . . 2634
smooth_funct_1d_mean . . . 2635
transform_funct_1d . . . 2636
write_funct_1d . . . 2637
x_range_funct_1d . . . 2637
y_range_funct_1d . . . 2638
zero_crossings_funct_1d . . . 2638
26.2 Geometrie . . . 2638
angle_ll . . . 2638
angle_lx . . . 2640
apply_distance_transform_xld . . . 2641
area_intersection_rectangle2 . . . 2642

clear_distance_transform_xld . . . 2643
create_distance_transform_xld . . . 2643
deserialize_distance_transform_xld . . . 2645
distance_cc . . . 2646
distance_cc_min . . . 2647
distance_cc_min_points . . . 2648
distance_contours_xld . . . 2649
distance_lc . . . 2650
distance_lr . . . 2651
distance_pc . . . 2652
distance_pl . . . 2652
distance_pp . . . 2653
distance_pr . . . 2654
distance_ps . . . 2655
distance_rr_min . . . 2656
distance_rr_min_dil . . . 2657
distance_sc . . . 2658
distance_sl . . . 2658
distance_sr . . . 2660
distance_ss . . . 2661
get_distance_transform_xld_contour . . . 2662
get_distance_transform_xld_param . . . 2662
get_points_ellipse . . . 2663
intersection_circle_contour_xld . . . 2665
intersection_circles . . . 2666
intersection_contours_xld . . . 2667
intersection_line_circle . . . 2668
intersection_line_contour_xld . . . 2669
intersection_lines . . . 2670
intersection_segment_circle . . . 2671
intersection_segment_contour_xld . . . 2672
intersection_segment_line . . . 2673
intersection_segments . . . 2674
projection_pl . . . 2674
read_distance_transform_xld . . . 2676
serialize_distance_transform_xld . . . 2676
set_distance_transform_xld_param . . . 2677
write_distance_transform_xld . . . 2678
26.3 Gitter-Entzerrung . . . 2679
connect_grid_points . . . 2679
create_rectification_grid . . . 2680
find_rectification_grid . . . 2681
gen_arbitrary_distortion_map . . . 2682
gen_grid_rectification_map . . . 2683
26.4 Hintergrundschätzer . . . 2685
close_bg_esti . . . 2685
create_bg_esti . . . 2686
get_bg_esti_params . . . 2688
give_bg_esti . . . 2690
run_bg_esti . . . 2691
set_bg_esti_params . . . 2692
update_bg_esti . . . 2694
26.5 Hough . . . 2695
hough_circle_trans . . . 2695
hough_circles . . . 2696
hough_line_trans . . . 2697
hough_line_trans_dir . . . 2698
hough_lines . . . 2699

hough_lines_dir . . . 2700
select_matching_lines . . . 2702
26.6 Interpolation . . . 2703
clear_scattered_data_interpolator . . . 2703
create_scattered_data_interpolator . . . 2704
interpolate_scattered_data . . . 2705
interpolate_scattered_data_image . . . 2706
interpolate_scattered_data_points_to_image . . . 2707
26.7 Linien . . . 2708
line_orientation . . . 2708
line_position . . . 2709
26.8 Mosaicking . . . 2710
adjust_mosaic_images . . . 2710
bundle_adjust_mosaic . . . 2713
gen_bundle_adjusted_mosaic . . . 2716
gen_cube_map_mosaic . . . 2717
gen_projective_mosaic . . . 2719
gen_spherical_mosaic . . . 2721
proj_match_points_distortion_ransac . . . 2724
proj_match_points_distortion_ransac_guided . . . 2727
proj_match_points_ransac . . . 2731
proj_match_points_ransac_guided . . . 2734

27 Transformationen **2737**
27.1 2D-Transformationen . . . 2737
affine_trans_pixel . . . 2742
affine_trans_point_2d . . . 2743
deserialize_hom_mat2d . . . 2744
hom_mat2d_compose . . . 2745
hom_mat2d_determinant . . . 2746
hom_mat2d_identity . . . 2746
hom_mat2d_invert . . . 2747
hom_mat2d_reflect . . . 2748
hom_mat2d_reflect_local . . . 2749
hom_mat2d_rotate . . . 2751
hom_mat2d_rotate_local . . . 2752
hom_mat2d_scale . . . 2753
hom_mat2d_scale_local . . . 2755
hom_mat2d_slant . . . 2756
hom_mat2d_slant_local . . . 2758
hom_mat2d_to_affine_par . . . 2759
hom_mat2d_translate . . . 2760
hom_mat2d_translate_local . . . 2761
hom_mat2d_transpose . . . 2763
hom_mat3d_project . . . 2763
hom_vector_to_proj_hom_mat2d . . . 2765
point_line_to_hom_mat2d . . . 2767
projective_trans_pixel . . . 2770
projective_trans_point_2d . . . 2771
serialize_hom_mat2d . . . 2772
vector_angle_to_rigid . . . 2773
vector_field_to_hom_mat2d . . . 2774
vector_to_aniso . . . 2775
vector_to_hom_mat2d . . . 2776
vector_to_proj_hom_mat2d . . . 2778
vector_to_proj_hom_mat2d_distortion . . . 2780
vector_to_rigid . . . 2782
vector_to_similarity . . . 2783

27.2 3D-Transformationen . . . 2785
affine_trans_point_3d . . . 2785
deserialize_hom_mat3d . . . 2786
hom_mat3d_compose . . . 2786
hom_mat3d_determinant . . . 2788
hom_mat3d_identity . . . 2788
hom_mat3d_invert . . . 2789
hom_mat3d_rotate . . . 2790
hom_mat3d_rotate_local . . . 2792
hom_mat3d_scale . . . 2793
hom_mat3d_scale_local . . . 2795
hom_mat3d_to_pose . . . 2796
hom_mat3d_translate . . . 2797
hom_mat3d_translate_local . . . 2798
hom_mat3d_transpose . . . 2799
pose_to_hom_mat3d . . . 2800
projective_trans_hom_point_3d . . . 2801
projective_trans_point_3d . . . 2802
serialize_hom_mat3d . . . 2803
vector_to_hom_mat3d . . . 2804
27.3 Duale Quaternionen . . . 2805
deserialize_dual_quat . . . 2807
dual_quat_compose . . . 2807
dual_quat_conjugate . . . 2808
dual_quat_interpolate . . . 2809
dual_quat_normalize . . . 2810
dual_quat_to_hom_mat3d . . . 2811
dual_quat_to_screw . . . 2812
dual_quat_trans_line_3d . . . 2813
screw_to_dual_quat . . . 2814
serialize_dual_quat . . . 2816
27.4 Posen . . . 2816
convert_pose_type . . . 2817
create_pose . . . 2818
deserialize_pose . . . 2822
dual_quat_to_pose . . . 2822
get_circle_pose . . . 2823
get_pose_type . . . 2824
get_rectangle_pose . . . 2825
pose_average . . . 2828
pose_compose . . . 2829
pose_invert . . . 2830
pose_to_dual_quat . . . 2831
pose_to_quat . . . 2832
proj_hom_mat2d_to_pose . . . 2832
quat_to_pose . . . 2833
read_pose . . . 2834
serialize_pose . . . 2835
set_origin_pose . . . 2835
vector_to_pose . . . 2836
write_pose . . . 2839
27.5 Quaternionen . . . 2840
axis_angle_to_quat . . . 2840
deserialize_quat . . . 2841
quat_compose . . . 2841
quat_conjugate . . . 2842
quat_interpolate . . . 2843
quat_normalize . . . 2843

quat_rotate_point_3d . . . 2844
quat_to_hom_mat3d . . . 2845
serialize_quat . . . 2845
27.6 Sonstiges . . . 2846
convert_point_3d_cart_to_spher . . . 2846
convert_point_3d_spher_to_cart . . . 2848

28 Tupel **2851**
28.1 Arithmetik . . . 2851
tuple_abs . . . 2851
tuple_acos . . . 2851
tuple_acosh . . . 2852
tuple_add . . . 2853
tuple_asin . . . 2853
tuple_asinh . . . 2854
tuple_atan . . . 2855
tuple_atan2 . . . 2855
tuple_atanh . . . 2856
tuple_cbrt . . . 2857
tuple_ceil . . . 2857
tuple_cos . . . 2858
tuple_cosh . . . 2858
tuple_cumul . . . 2859
tuple_deg . . . 2860
tuple_div . . . 2860
tuple_erf . . . 2861
tuple_erfc . . . 2861
tuple_exp . . . 2862
tuple_exp10 . . . 2863
tuple_exp2 . . . 2863
tuple_fabs . . . 2864
tuple_floor . . . 2865
tuple_fmod . . . 2865
tuple_hypot . . . 2866
tuple_ldexp . . . 2867
tuple_lgamma . . . 2867
tuple_log . . . 2868
tuple_log10 . . . 2868
tuple_log2 . . . 2869
tuple_max2 . . . 2870
tuple_min2 . . . 2870
tuple_mod . . . 2871
tuple_mult . . . 2872
tuple_neg . . . 2872
tuple_pow . . . 2873
tuple_rad . . . 2874
tuple_sgn . . . 2874
tuple_sin . . . 2875
tuple_sinh . . . 2875
tuple_sqrt . . . 2876
tuple_sub . . . 2877
tuple_tan . . . 2877
tuple_tanh . . . 2878
tuple_tgamma . . . 2879
28.2 Bitoperationen . . . 2879
tuple_band . . . 2879
tuple_bnot . . . 2880
tuple_bor . . . 2881
tuple_bxor . . . 2881

tuple_lsh . . . 2882
tuple_rsh . . . 2883
28.3 Datencontainer . . . 2883
copy_dict . . . 2883
create_dict . . . 2884
get_dict_object . . . 2885
get_dict_param . . . 2886
get_dict_tuple . . . 2888
read_dict . . . 2889
remove_dict_key . . . 2890
set_dict_object . . . 2891
set_dict_tuple . . . 2892
write_dict . . . 2893
28.4 Elementauswahl . . . 2894
tuple_find . . . 2894
tuple_find_first . . . 2895
tuple_find_last . . . 2895
tuple_first_n . . . 2896
tuple_last_n . . . 2897
tuple_select . . . 2897
tuple_select_mask . . . 2898
tuple_select_range . . . 2899
tuple_select_rank . . . 2900
tuple_str_bit_select . . . 2900
tuple_uniq . . . 2901
28.5 Elementreihenfolge . . . 2902
tuple_inverse . . . 2902
tuple_sort . . . 2903
tuple_sort_index . . . 2903
28.6 Generierung . . . 2904
clear_handle . . . 2904
tuple_concat . . . 2905
tuple_gen_const . . . 2905
tuple_gen_sequence . . . 2906
tuple_rand . . . 2907
28.7 Konversion . . . 2907
handle_to_integer . . . 2907
integer_to_handle . . . 2908
tuple_chr . . . 2909
tuple_chrt . . . 2910
tuple_int . . . 2910
tuple_is_number . . . 2911
tuple_number . . . 2912
tuple_ord . . . 2912
tuple_ords . . . 2913
tuple_real . . . 2914
tuple_round . . . 2914
tuple_string . . . 2915
28.8 Logische Operationen . . . 2918
tuple_and . . . 2918
tuple_not . . . 2919
tuple_or . . . 2919
tuple_xor . . . 2920
28.9 Manipulation . . . 2921
tuple_insert . . . 2921
tuple_remove . . . 2921
tuple_replace . . . 2922
28.10 Mengen-Operationen . . . 2923

tuple_difference . . . 2923
tuple_intersection . . . 2924
tuple_symmdiff . . . 2924
tuple_union . . . 2925
28.11 Merkmale . . . 2926
get_handle_object . . . 2926
get_handle_param . . . 2927
get_handle_tuple . . . 2928
tuple_deviation . . . 2929
tuple_histo_range . . . 2929
tuple_length . . . 2930
tuple_max . . . 2931
tuple_mean . . . 2931
tuple_median . . . 2932
tuple_min . . . 2933
tuple_sum . . . 2933
28.12 String-Operationen . . . 2934
tuple_environment . . . 2935
tuple_regexp_match . . . 2935
tuple_regexp_replace . . . 2938
tuple_regexp_select . . . 2939
tuple_regexp_test . . . 2940
tuple_split . . . 2941
tuple_str_first_n . . . 2942
tuple_str_last_n . . . 2943
tuple_strchr . . . 2944
tuple_strlen . . . 2945
tuple_strrchr . . . 2946
tuple_strrstr . . . 2947
tuple_strstr . . . 2948
tuple_substr . . . 2949
28.13 Typ . . . 2950
tuple_is_handle . . . 2950
tuple_is_handle_elem . . . 2950
tuple_is_int . . . 2951
tuple_is_int_elem . . . 2952
tuple_is_mixed . . . 2953
tuple_is_real . . . 2954
tuple_is_real_elem . . . 2955
tuple_is_string . . . 2955
tuple_is_string_elem . . . 2956
tuple_is_valid_handle . . . 2957
tuple_sem_type . . . 2958
tuple_sem_type_elem . . . 2958
tuple_type . . . 2959
tuple_type_elem . . . 2961
28.14 Vergleich . . . 2961
tuple_equal . . . 2961
tuple_equal_elem . . . 2962
tuple_greater . . . 2963
tuple_greater_elem . . . 2964
tuple_greater_equal . . . 2964
tuple_greater_equal_elem . . . 2965
tuple_less . . . 2966
tuple_less_elem . . . 2967
tuple_less_equal . . . 2967
tuple_less_equal_elem . . . 2968
tuple_not_equal . . . 2969

tuple_not_equal_elem . . . 2970

29 XLD **2971**

29.1 Generierung . . . 2971
gen_circle_contour_xld . . . 2971
gen_contour_nurbs_xld . . . 2972
gen_contour_polygon_rounded_xld . . . 2974
gen_contour_polygon_xld . . . 2975
gen_contour_region_xld . . . 2975
gen_contours_skeleton_xld . . . 2977
gen_cross_contour_xld . . . 2978
gen_ellipse_contour_xld . . . 2978
gen_nurbs_interp . . . 2980
gen_parallels_xld . . . 2981
gen_polygons_xld . . . 2982
gen_rectangle2_contour_xld . . . 2983
mod_parallels_xld . . . 2984

29.2 Geometrische Transformationen . . . 2985
affine_trans_contour_xld . . . 2985
affine_trans_polygon_xld . . . 2986
gen_parallel_contour_xld . . . 2987
polar_trans_contour_xld . . . 2988
polar_trans_contour_xld_inv . . . 2990
projective_trans_contour_xld . . . 2992

29.3 Mengen-Operationen . . . 2993
difference_closed_contours_xld . . . 2993
difference_closed_polygons_xld . . . 2994
intersection_closed_contours_xld . . . 2995
intersection_closed_polygons_xld . . . 2996
symm_difference_closed_contours_xld . . . 2997
symm_difference_closed_polygons_xld . . . 2999
union2_closed_contours_xld . . . 3000
union2_closed_polygons_xld . . . 3001

29.4 Merkmale . . . 3002
area_center_points_xld . . . 3002
area_center_xld . . . 3003
circularity_xld . . . 3004
compactness_xld . . . 3005
contour_point_num_xld . . . 3005
convexity_xld . . . 3006
diameter_xld . . . 3007
dist_ellipse_contour_points_xld . . . 3008
dist_ellipse_contour_xld . . . 3009
dist_rectangle2_contour_points_xld . . . 3011
eccentricity_points_xld . . . 3012
eccentricity_xld . . . 3013
elliptic_axis_points_xld . . . 3014
elliptic_axis_xld . . . 3015
fit_circle_contour_xld . . . 3016
fit_ellipse_contour_xld . . . 3018
fit_line_contour_xld . . . 3021
fit_rectangle2_contour_xld . . . 3023
get_contour_angle_xld . . . 3026
get_contour_attrib_xld . . . 3027
get_contour_global_attrib_xld . . . 3030
get_regress_params_xld . . . 3033
height_width_ratio_xld . . . 3035
info_parallels_xld . . . 3036

length_xld . . . 3036
local_max_contours_xld . . . 3037
max_parallels_xld . . . 3038
moments_any_points_xld . . . 3039
moments_any_xld . . . 3040
moments_points_xld . . . 3042
moments_xld . . . 3043
orientation_points_xld . . . 3044
orientation_xld . . . 3044
query_contour_attribs_xld . . . 3045
query_contour_global_attribs_xld . . . 3046
rectangularity_xld . . . 3047
select_contours_xld . . . 3047
select_shape_xld . . . 3049
select_xld_point . . . 3051
smallest_circle_xld . . . 3052
smallest_rectangle1_xld . . . 3053
smallest_rectangle2_xld . . . 3054
test_closed_xld . . . 3055
test_self_intersection_xld . . . 3055
test_xld_point . . . 3056
29.5 Transformationen . . . 3057
add_noise_white_contour_xld . . . 3057
clip_contours_xld . . . 3058
clip_end_points_contours_xld . . . 3059
close_contours_xld . . . 3059
combine_roads_xld . . . 3060
crop_contours_xld . . . 3061
merge_cont_line_scan_xld . . . 3062
regress_contours_xld . . . 3063
segment_contour_attrib_xld . . . 3065
segment_contours_xld . . . 3066
shape_trans_xld . . . 3068
smooth_contours_xld . . . 3069
sort_contours_xld . . . 3070
split_contours_xld . . . 3071
union_adjacent_contours_xld . . . 3071
union_cocircular_contours_xld . . . 3073
union_collinear_contours_ext_xld . . . 3076
union_collinear_contours_xld . . . 3078
union_cotangential_contours_xld . . . 3082
union_straight_contours_xld . . . 3086
29.6 Zugriff . . . 3088
get_contour_xld . . . 3088
get_lines_xld . . . 3089
get_parallels_xld . . . 3090
get_polygon_xld . . . 3091

Index **3093**

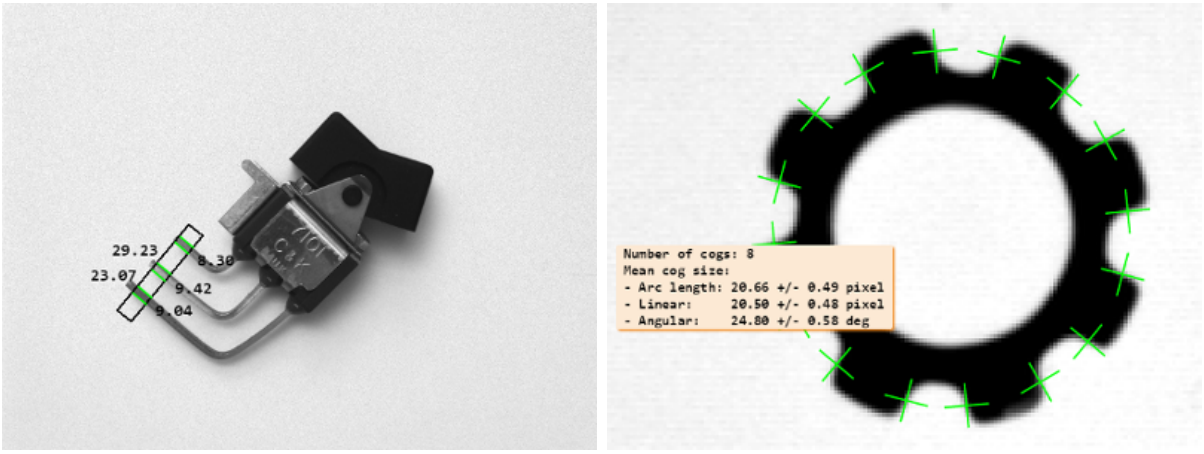
Kapitel 1

1D-Vermessung

Dieses Kapitel beschreibt Operatoren für die 1D-Vermessung.

Konzept der 1D-Vermessung

Mit der 1D-Vermessung können Kanten, d.h., Übergänge von hell nach dunkel oder von dunkel nach hell entlang einer definierten Linie oder eines Kreisbogens gefunden werden. Dies erlaubt die einfache und schnelle Vermessung der Ausmaße von Teilen mit hoher Genauigkeit. Sollen die Ausmaße von geometrischen Primitiven wie Kreise, Ellipsen, Rechtecke oder Linien, für deren Positionen, Ausrichtungen und geometrische Formen Näherungswerte vorhanden sind, vermessen werden, kann 2D-Messtechnik eine sinnvolle Alternative darstellen.



(1) (2)

Vermessung von Kanten und den Distanzen zwischen ihnen entlang einer Linie (1) oder einem Kreisringsegment (2). Diese Bilder stammen aus den Beispielprogrammen fuzzy_measure_pin.hdev und measure_ring.hdev.

Im Folgenden werden die einzelnen Schritte, die zur Benutzung der 1D-Vermessung benötigt werden, kurz beschrieben.

Generieren eines Messobjekts: Zuerst muss ein Messobjekt generiert werden, welches die Region für die Vermessung beschreibt. Soll entlang einer Linie gemessen werden, ist die Region ein Rechteck. Soll entlang eines Kreisbogens gemessen werden, ist die Region ein Kreisringsegment. Die Generierung des Messobjekts erfolgt über einen der Operatoren

- `gen_measure_rectangle2` oder
- `gen_measure_arc`.

Für eine automatische Ausrichtung des Messobjekts kann formbasiertes Matching (siehe Kapitel Matching / Formbasiert) hilfreich sein.

Ausführen der Messung: Für die eigentliche Vermessung wird dann typischerweise einer der folgenden Operatoren verwendet:

- measure_pos findet geradlinige Kanten, die senkrecht zur Hauptachse des Messobjekts liegen und liefert die Positionen der Kantenmittelpunkte, die Kantenamplituden und die Distanzen zwischen aufeinanderfolgenden Kanten zurück.
- measure_pairs findet geradlinige Kanten, die senkrecht zur Hauptachse des Messobjekts liegen und liefert die Positionen der Kantenmittelpunkte von Kantenpaaren, die Kantenamplituden von Kantenpaaren, die Distanzen zwischen den Kanten der Kantenpaare und die Distanzen zwischen aufeinanderfolgenden Kantenpaaren zurück.
- measure_thresh findet Punkte mit einem bestimmten Grauwert entlang der Hauptachse des Messobjekts und liefert ihre Positionen und die Distanzen zwischen aufeinanderfolgenden Punkten zurück.

Alternativ, wenn zusätzliche Kanten vorhanden sind, die nicht zur Messung gehören, kann auch eine Fuzzy-Vermessung erfolgen. Hier müssen sogenannte Fuzzy-Funktionen definiert werden, die die Merkmale guter Kanten beschreiben. Mögliche Merkmale sind z.B. die Positionen, die Distanzen, die Grauwerte oder die Amplituden der Kanten. Die Funktionen werden mit create_funct_1d_pairs erstellt und an das Messtool mit set_fuzzy_measure oder set_fuzzy_measure_norm_pair übergeben. Basierend auf diesen Funktionen kann dann einer der folgenden Operatoren die am besten geeigneten Kanten auswählen:

- fuzzy_measure_pos findet geradlinige Kanten, die senkrecht zur Hauptachse des Messobjekts liegen und liefert die Positionen der Kantenmittelpunkte, die Kantenamplituden, die Fuzzy-Bewertungen und die Distanzen zwischen aufeinanderfolgenden Kanten zurück.
- fuzzy_measure_pairs findet geradlinige Kanten, die senkrecht zur Hauptachse des Messobjekts liegen und liefert die Positionen der Kanten von Kantenpaaren, die Kantenamplituden von Kantenpaaren, die Positionen der Mittelpunkte der Kantenpaare, die Fuzzy-Bewertungen, die Distanzen zwischen den Kanten der Kantenpaare und die Distanzen zwischen aufeinanderfolgenden Kantenpaaren zurück.
- fuzzy_measure_pairing ist wie fuzzy_measure_pairs mit der Ausnahme, dass es über den Parameter Pairing zusätzlich möglich ist, sich überschneidende und ineinanderliegende Kantenpaare zu extrahieren.

Alternativ zur automatischen Extraktion von Kanten oder Punkten innerhalb des Messobjekts kann auch das eindimensionale Grauwertprofil senkrecht zur Hauptachse des Messobjekts extrahiert werden und daraus individuell die benötigte Information abgeleitet werden. Das Extrahieren des Grauwertprofils erfolgt mit dem Operator

- measure_projection.

Freigeben des Handles für das Messobjekt: Wird das Messobjekt nicht mehr gebraucht, erfolgt die Freigabe seines Handles mit

- close_measure.

Weitere Operatoren

Zusätzlich zu den oben genannten Operatoren kann mit reset_fuzzy_measure eine Fuzzy-Funktion, die mit set_fuzzy_measure oder set_fuzzy_measure_norm_pair vorher gesetzt wurde, gelöscht werden, mit translate_measure der Referenzpunkt des Messobjekts an eine festgelegte Position verschoben werden, mit write_measure und read_measure das Messobjekt in eine Datei geschrieben und daraus wieder ausgelesen werden, und mit serialize_measure und deserialize_measure kann das Messobjekt serialisiert bzw. deserialisiert werden.

Glossar

Im Folgenden werden die wichtigsten Begriffe, die im Umfeld der 1D-Vermessung verwendet werden, beschrieben:

Messobjekt Datenstruktur für eine rechteckige oder kreisringförmige Region, die für die Extraktion geradliniger Kanten senkrecht zu ihrer Hauptachse vorbereitet ist.

Kreisring Ein Kreisbogen mit zugehöriger Dicke.

Weiterführende Information

Weitere Details zur 1D-Vermessung finden sich im finden sich im „Solution Guide Basics“ und „Solution Guide on 1D Measuring“.

close_measure (: : MeasureHandle :)

Löschen eines Messobjekts.

close_measure löscht das Messobjekt, das in MeasureHandle übergeben wird. Der für das Messobjekt verwendete Speicher wird freigegeben.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Parameter

▷ **MeasureHandle** (input_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert close_measure den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MeasureHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

gen_measure_rectangle2, gen_measure_arc, measure_pos, measure_pairs

Modul

1D Metrology

deserialize_measure (: : SerializedItemHandle : MeasureHandle)

Deserialisieren eines serialisierten Messobjekts.

deserialize_measure deserialisiert ein Messobjekt, welches mit serialize_measure serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Messobjekt wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Messobjekt mit dem Handle MeasureHandle gespeichert.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **MeasureHandle** (output_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_measure` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_measure`

Nachfolger

`measure_pos`, `measure_pairs`

Siehe auch

`read_measure`, `write_measure`

Modul

1D Metrology

fuzzy_measure_pairing (Image : : MeasureHandle, Sigma, AmpThresh, FuzzyThresh, Transition, Pairing, NumPairs : RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, RowPairCenter, ColumnPairCenter, FuzzyScore, IntraDistance)

Extraktion geradliniger Kantenpaare senkrecht zu einem Rechteck oder Kreisringsegment.

`fuzzy_measure_pairing` dient zur Extraktion von *geradlinigen* Kantenpaaren, die *senkrecht* zur Hauptachse eines Rechtecks oder zu einem Kreisbogensegment liegen.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Der Algorithmus von `fuzzy_measure_pairing` arbeitet analog zu `fuzzy_measure_pairs` (siehe dort für Details) mit der Ausnahme, dass es über den Parameter `Pairing` zusätzlich möglich ist, sich überschneidende und ineinanderliegende Kantenpaare zu extrahieren. Derzeit steht nur der Wert *'no_restriction'* zu Verfügung, der alle möglichen Kantenpaare zurückliefert, die sich sowohl gegenseitig überlagern als auch einschließen können.

Es werden nur die `NumPairs` am besten bewerteten Kantenpaare zurückgeliefert, wobei der Wert 0 bedeutet, dass alle gefundenen Kantenkombinationen zurückgeliefert werden.

Die extrahierten Kanten werden als einzelne Punkte, die auf der Hauptachse des Rechtecks oder dem mittleren Bogen des Kreisringsegments liegen, zurückgegeben. Die zugehörigen Kantenamplituden werden in `AmplitudeFirst` und `AmplitudeSecond`, die Fuzzy-Bewertungen in `FuzzyScore` zurückgeliefert. Zusätzlich wird noch der Abstand innerhalb eines Kantenpaares in `IntraDistance` zurückgeliefert.

Achtung

`fuzzy_measure_pairing` liefert nur dann brauchbare Ergebnisse, falls die Annahme, dass die Kanten gerade und senkrecht zur Hauptachse des Rechtecks oder zum Kreisringsegment sind, erfüllt ist. `fuzzy_measure_pairing` sollte daher z.B. nicht dazu verwendet werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, dass das Rechteck / das Kreisringsegment möglichst senkrecht zu den Bildkanten liegt. Außerdem darf `Sigma` nicht größer als ca. 0.5 * Length1 (siehe `gen_measure_rectangle2` für Length1) sein.

Es ist zu beachten, dass `fuzzy_measure_pairing` den Definitionsbereich von `Image` aus Geschwindigkeitsgründen ignoriert. Falls bestimmte Bildbereiche von der Messung ausgeschlossen werden sollen, sollte ein neues Messobjekt mit entsprechend angepassten Parametern erzeugt werden.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **MeasureHandle** (input_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

▷ **Sigma** (input_control) . . . . . number ⇝ *real*
Sigma der Gaußglättung.
Defaultwert: 1.0
Wertevorschläge: `Sigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ `Sigma` ≤ 100 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Sigma >= 0.4

▷ **AmpThresh** (input_control) . . . . . number ⇝ *real*
Minimale Amplitude einer Kante.
Defaultwert: 30.0
Wertevorschläge: `AmpThresh` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich: 1 ≤ `AmpThresh` ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 2

▷ **FuzzyThresh** (input_control) . . . . . number ⇝ *real*
Minimale Fuzzy-Bewertung.
Defaultwert: 0.5
Wertevorschläge: `FuzzyThresh` ∈ {0.1, 0.3, 0.5, 0.7, 0.9}
Typischer Wertebereich: 0.0 ≤ `FuzzyThresh` ≤ 1.0 (lin)
Empfohlene Schrittweite: 0.1

▷ **Transition** (input_control) . . . . . string ⇝ *string*
Auswahl des ersten Grauwertüberganges der Kantenpaare.
Defaultwert: 'all'
Werteliste: `Transition` ∈ {'all', 'positive', 'negative'}

▷ **Pairing** (input_control) . . . . . string ⇝ *string*
Beschränkung der Paarbildung.
Defaultwert: 'no_restriction'
Werteliste: `Pairing` ∈ {'no_restriction'}

▷ **NumPairs** (input_control) . . . . . number ⇝ *integer*
Maximale Anzahl der Kantenpaare.
Defaultwert: 10
Wertevorschläge: `NumPairs` ∈ {0, 1, 10, 20, 50}
Typischer Wertebereich: 0 ≤ NumPairs
Empfohlene Schrittweite: 1

▷ **RowEdgeFirst** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate der ersten Kante.

▷ **ColumnEdgeFirst** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate der ersten Kante.

▷ **AmplitudeFirst** (output_control) . . . . . real-array ⇝ *real*
Kantenamplitude der ersten Kante (mit Vorzeichen).

▷ **RowEdgeSecond** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate der zweiten Kante.

▷ **ColumnEdgeSecond** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate der zweiten Kante.

▷ **AmplitudeSecond** (output_control) . . . . . real-array ⇝ *real*
Kantenamplitude der zweiten Kante (mit Vorzeichen).

▷ **RowPairCenter** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate des Mittelpunktes des Kantenpaares.

▷ **ColumnPairCenter** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate des Mittelpunktes des Kantenpaares.

▷ **FuzzyScore** (output_control) ..... real-array ⇝ *real*
Fuzzy-Bewertung des Kantenpaares.

▷ **IntraDistance** (output_control) ..... real-array ⇝ *real*
Abstand zwischen Kanten eines Kantenpaars.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `fuzzy_measure_pairing` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_measure_rectangle2`, `gen_measure_arc`, `set_fuzzy_measure`

Nachfolger

`close_measure`

Alternativen

`edges_sub_pix`, `fuzzy_measure_pairs`, `measure_pairs`

Siehe auch

`fuzzy_measure_pos`, `measure_pos`

Modul

1D Metrology

fuzzy_measure_pairs (Image : : MeasureHandle, Sigma, AmpThresh, FuzzyThresh, Transition : RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, RowEdgeCenter, ColumnEdgeCenter, FuzzyScore, IntraDistance, InterDistance)

Extraktion geradliniger Kantenpaare senkrecht zu einem Rechteck oder Kreisringsegment.

`fuzzy_measure_pairs` dient zur Extraktion von *geradlinigen* Kantenpaaren, die *senkrecht* zur Hauptachse eines Rechtecks oder zu einem Kreisringsegment liegen. Im Unterschied zu `measure_pairs` können die Kantenpaare zusätzlich mit Hilfe von Fuzzy-Funktionen bewertet und ausgewählt werden.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Der Algorithmus von `fuzzy_measure_pairs` arbeitet analog zu `fuzzy_measure_pos`. Benachbarte Kanten werden jedoch zu Paaren mit unterschiedlichen Kantenübergängen (Vorzeichen) zusammengefasst. Um sich überlappende oder einschließende Kantenpaare zu extrahieren, verwenden Sie bitte `fuzzy_measure_pairing`.

Falls `Transition` = *'positive'* werden in `RowEdgeFirst` und `ColumnEdgeFirst` Kantenpunkte mit Übergang von Dunkel nach Hell und in `RowEdgeSecond` und `ColumnEdgeSecond` mit Übergang von Hell nach Dunkel übergeben. Im Modus *'negative'* ist dies genau umgekehrt. Im Modus *'all'* bestimmt die erste gefundene Kante die Art der Paarbildung. D.h., je nach Lage des Messobjekts werden entweder hell-dunkel-hell Übergänge oder dunkel-hell-dunkel Übergänge gefunden. Dieses Vorgehen ist dann nützlich, wenn z.B. Objekte vermessen werden, die unterschiedliche Helligkeiten in Relation zum Hintergrund aufweisen.

Sind die subpixelgenauen Positionen der Kanten ermittelt, werden sie zu Paaren gruppiert. Dabei werden die Kanten derart zu Paaren sortiert, dass Überschneidungen und Einschlüsse der Paaren untereinander verhindert werden. Die Merkmale der Kantenpaare werden mit Hilfe einer zugehörigen Fuzzy-Funktion bewertet, welche mit `set_fuzzy_measure` b.z.w. `set_fuzzy_measure_norm_pair` gesetzt werden kann. Welche Kantenpaare ausgewählt werden wird mit dem Parameter `FuzzyThresh` beeinflusst, welcher eine Schwelle über die Bewertung aller Fuzzy-Sets angibt, d.h. über den geometrischen Mittelwert der einzelnen Gewichtungen der Sets.

Die extrahierten Kanten werden als einzelne Punkte, die auf der Hauptachse des Rechtecks oder dem mittleren Bogen des Kreisringsegments liegen, zurückgegeben. Die zugehörigen Kantenamplituden werden in

AmplitudeFirst und AmplitudeSecond, die Fuzzy-Bewertungen in FuzzyScore zurückgeliefert. Zusätzlich wird noch der Abstand innerhalb eines Kantenpaares in IntraDistance und der Abstand von aufeinanderfolgenden Kantenpaaren in InterDistance zurückgeliefert. Dabei entspricht IntraDistance[i] dem Abstand von EdgeFirst[i] und EdgeSecond[i], während InterDistance[i] dem Abstand von EdgeSecond[i] und EdgeFirst[i+1] entspricht. D.h., das Tupel InterDistance enthält ein Element weniger als die Tupel der Kantenpaare.

Achtung

fuzzy_measure_pairs liefert nur dann brauchbare Ergebnisse, falls die Annahme, dass die Kanten gerade und senkrecht zur Hauptachse des Rechtecks oder zum Kreisringsegment sind, erfüllt ist. fuzzy_measure_pairs sollte daher z.B. nicht dazu verwendet werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, dass das Rechteck / das Kreisringsegment möglichst senkrecht zu den Bildkanten liegt. Außerdem darf Sigma nicht größer als ca. 0.5 * Length1 (siehe gen_measure_rectangle2 für Length1) sein.

Es ist zu beachten, dass fuzzy_measure_pairs den Definitionsbereich von Image aus Geschwindigkeitsgründen ignoriert. Falls bestimmte Bildbereiche von der Messung ausgeschlossen werden sollen, sollte ein neues Messobjekt mit entsprechend angepassten Parametern erzeugt werden.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **MeasureHandle** (input_control) ..... measure ⇝ *handle*
  Handle des Messobjekts.
- ▷ **Sigma** (input_control) ..... number ⇝ *real*
  Sigma der Gaußglättung.
  Defaultwert: 1.0
  Wertevorschläge: Sigma ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
  Typischer Wertebereich: 0.4 ≤ Sigma ≤ 100 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Sigma >= 0.4
- ▷ **AmpThresh** (input_control) ..... number ⇝ *real*
  Minimale Amplitude einer Kante.
  Defaultwert: 30.0
  Wertevorschläge: AmpThresh ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
  Typischer Wertebereich: 1 ≤ AmpThresh ≤ 255 (lin)
  Minimale Schrittweite: 0.5
  Empfohlene Schrittweite: 2
- ▷ **FuzzyThresh** (input_control) ..... number ⇝ *real*
  Minimale Fuzzy-Bewertung.
  Defaultwert: 0.5
  Wertevorschläge: FuzzyThresh ∈ {0.1, 0.3, 0.5, 0.7, 0.9}
  Typischer Wertebereich: 0.0 ≤ FuzzyThresh ≤ 1.0 (lin)
  Empfohlene Schrittweite: 0.1
- ▷ **Transition** (input_control) ..... string ⇝ *string*
  Auswahl des ersten Grauwertüberganges der Kantenpaare.
  Defaultwert: 'all'
  Werteliste: Transition ∈ {'all', 'positive', 'negative'}
- ▷ **RowEdgeFirst** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinate der ersten Kante.
- ▷ **ColumnEdgeFirst** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinate der ersten Kante.
- ▷ **AmplitudeFirst** (output_control) ..... real-array ⇝ *real*
  Kantenamplitude der ersten Kante (mit Vorzeichen).
- ▷ **RowEdgeSecond** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinate der zweiten Kante.
- ▷ **ColumnEdgeSecond** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Kante.

▷ **AmplitudeSecond** (output_control) ..... real-array ⇝ *real*
Kantenamplitude der zweiten Kante (mit Vorzeichen).

▷ **RowEdgeCenter** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinate des Mittelpunktes des Kantenpaares.

▷ **ColumnEdgeCenter** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinate des Mittelpunktes des Kantenpaares.

▷ **FuzzyScore** (output_control) ..... real-array ⇝ *real*
Fuzzy-Bewertung des Kantenpaares.

▷ **IntraDistance** (output_control) ..... real-array ⇝ *real*
Abstand zwischen Kanten eines Kantenpaars.

▷ **InterDistance** (output_control) ..... real-array ⇝ *real*
Abstand zwischen aufeinanderfolgenden Kantenpaaren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `fuzzy_measure_pairs` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_measure_rectangle2`, `gen_measure_arc`, `set_fuzzy_measure`

Nachfolger

`close_measure`

Alternativen

`edges_sub_pix`, `fuzzy_measure_pairing`, `measure_pairs`

Siehe auch

`fuzzy_measure_pos`, `measure_pos`

Modul

1D Metrology

fuzzy_measure_pos (Image : : MeasureHandle, Sigma, AmpThresh, FuzzyThresh, Transition : RowEdge, ColumnEdge, Amplitude, FuzzyScore, Distance)

Extraktion geradliniger Kanten senkrecht zu einem Rechteck oder Kreisringsegment.

`fuzzy_measure_pos` dient zur Extraktion von *geradlinigen* Kanten, die *senkrecht* zur Hauptachse eines Rechtecks oder zu einem Kreisringsegment liegen. Im Unterschied zu `measure_pos` können die Kanten zusätzlich mit Hilfe von Fuzzy-Funktionen bewertet und ausgewählt werden.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Der Algorithmus von `fuzzy_measure_pos` erzeugt zunächst ein eindimensionales Grauwertprofil durch Mittelung der Grauwerte entlang von Geraden senkrecht zur Hauptachse des Rechtecks oder zum Kreisringsegment. Dabei wird das Eingabebild `Image` an Subpixel-Positionen abgetastet, die einen ganzzahligen Zeilen- und Spalten-Abstand (im Koordinatensystem des Rechtecks) zum Mittelpunkt des Rechtecks haben. Weil die Abtastung des Bildes einige Berechnungen erfordert, die in mehreren Messungen verwendet werden können, wird der Operator `gen_measure_rectangle2` (bzw. `gen_measure_arc`) verwendet, um diese Berechnungen nur einmal ausführen zu müssen, und somit die Geschwindigkeit von `fuzzy_measure_pos` signifikant zu erhöhen. Aufgrund der Tatsache, dass eine bessere Interpolation bei der Subpixel-Abtastung der Grauwerte zu einer besseren Genauigkeit der extrahierten Kanten führt, aber auch die Laufzeit des Operators erhöht, können verschiedene Interpolationsverfahren in `gen_measure_rectangle2` (bzw. `gen_measure_arc`) gewählt werden. (Die Interpolation beeinflusst nur Rechtecke, die nicht parallel zu den Koordinatenachsen des Bildes sind, und Kreisringsegmente.) Das Messobjekt, das mit `gen_measure_rectangle2` (bzw. `gen_measure_arc`) erzeugt wurde, wird in `MeasureHandle` übergeben.

Nach der Bestimmung des eindimensionalen Kantenprofils werden die subpixelgenauen Kantenpositionen durch Faltung des Profils mit den Ableitungen einer Gauß-Maske mit Standardabweichung `Sigma` berechnet. Auffällige Kanten können mit dem Parameter `AmpThresh` selektiert werden, der einen Schwellenwert für die Kantenamplitude, d.h. den Absolutbetrag der ersten Ableitung, festlegt. Außerdem ist es möglich, nur positive Kanten, also solche, die einen Übergang von Dunkel nach Hell in der Richtung der Hauptachse des Rechtecks oder des Kreisringsegments darstellen (`Transition` = *'positive'*), nur negative Kanten, also Hell-Dunkel-Übergänge (`Transition` = *'negative'*) oder alle Kanten zu selektieren (`Transition` = *'all'*).

Sind die subpixelgenauen Kanten extrahiert, werden ausgewählte Merkmale der Kanten mit Hilfe einer zugehörigen Fuzzy-Funktion bewertet, welche mit `set_fuzzy_measure` gesetzt werden kann. Welche Kanten ausgewählt werden wird mit dem Parameter `FuzzyThresh` beeinflusst, welcher eine Schwelle über die Bewertung aller Fuzzy-Sets angibt, d.h. über den geometrischen Mittelwert der einzelnen Gewichtungen der Sets.

Die extrahierten Kanten werden als einzelne Punkte, die auf der Hauptachse des Rechtecks oder dem mittleren Bogen des Kreisringsegments liegen, in (`RowEdge`, `ColumnEdge`) zurückgegeben. Die zugehörigen Kantenamplituden werden in `Amplitude`, die Fuzzy-Bewertungen der Kanten in `FuzzyScore` zurückgeliefert. Zusätzlich wird noch der Abstand von aufeinanderfolgenden Kantenpunkten in `Distance` zurückgeliefert. Dabei entspricht Distance[i] dem Abstand von Edge[i] und Edge[i+1]. D.h., das Tupel `Distance` enthält ein Element weniger als die Tupel `RowEdge` und `ColumnEdge`.

Achtung

`fuzzy_measure_pos` liefert nur dann brauchbare Ergebnisse, falls die Annahme, dass die Kanten gerade und senkrecht zur Hauptachse des Rechtecks oder zum Kreisringsegment sind, erfüllt ist. `fuzzy_measure_pos` sollte daher z.B. nicht dazu verwendet werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, dass das Rechteck / das Kreisringsegment möglichst senkrecht zu den Bildkanten liegt. Außerdem darf `Sigma` nicht größer als ca. 0.5 * Length1 (siehe `gen_measure_rectangle2` für Length1) sein.

Es ist zu beachten, dass `fuzzy_measure_pos` den Definitionsbereich von `Image` aus Geschwindigkeitsgründen ignoriert. Falls bestimmte Bildbereiche von der Messung ausgeschlossen werden sollen, sollte ein neues Messobjekt mit entsprechend angepassten Parametern erzeugt werden.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **`MeasureHandle`** (input_control) ..... measure ⇝ *handle*
  Handle des Messobjekts.
- ▷ **`Sigma`** (input_control) ..... number ⇝ *real*
  Sigma der Gaußglättung.
  Defaultwert: 1.0
  Wertevorschläge: `Sigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
  Typischer Wertebereich: 0.4 ≤ `Sigma` ≤ 100 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Sigma >= 0.4
- ▷ **`AmpThresh`** (input_control) ..... number ⇝ *real*
  Minimale Amplitude einer Kante.
  Defaultwert: 30.0
  Wertevorschläge: `AmpThresh` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
  Typischer Wertebereich: 1 ≤ `AmpThresh` ≤ 255 (lin)
  Minimale Schrittweite: 0.5
  Empfohlene Schrittweite: 2
- ▷ **`FuzzyThresh`** (input_control) ..... number ⇝ *real*
  Minimale Fuzzy Bewertung.
  Defaultwert: 0.5
  Wertevorschläge: `FuzzyThresh` ∈ {0.1, 0.3, 0.5, 0.6, 0.7, 0.9}
  Typischer Wertebereich: 0.0 ≤ `FuzzyThresh` ≤ 1.0 (lin)
  Empfohlene Schrittweite: 0.1
- ▷ **`Transition`** (input_control) ..... string ⇝ *string*
  Hell/Dunkel oder Dunkel/Hell Kante.
  Defaultwert: 'all'
  Werteliste: `Transition` ∈ {'all', 'positive', 'negative'}

▷ **RowEdge** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinate des Mittelpunktes der Kante.

▷ **ColumnEdge** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinate des Mittelpunktes der Kante.

▷ **Amplitude** (output_control) ..... real-array ⇝ *real*
Kantenamplitude der Kanten (mit Vorzeichen).

▷ **FuzzyScore** (output_control) ..... real-array ⇝ *real*
Fuzzy-Bewertung der Kante.

▷ **Distance** (output_control) ..... real-array ⇝ *real*
Abstand zwischen aufeinanderfolgenden Kante.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `fuzzy_measure_pos` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_measure_rectangle2`, `gen_measure_arc`, `set_fuzzy_measure`

Nachfolger

`close_measure`

Alternativen

`edges_sub_pix`, `measure_pos`

Siehe auch

`fuzzy_measure_pairing`, `fuzzy_measure_pairs`, `measure_pairs`

Modul

1D Metrology

gen_measure_arc (: : CenterRow, CenterCol, Radius, AngleStart, AngleExtent, AnnulusRadius, Width, Height, Interpolation : MeasureHandle)

Vorbereitung der Extraktion geradliniger Kanten senkrecht zu einem Kreisringsegment.

`gen_measure_arc` bereitet die Extraktion *geradliniger* Kanten, die *senkrecht* zu einem Kreisringsegment liegen, vor. Mit Kreisring ist hier ein Kreisbogen mit zugehöriger Dicke gemeint. Der Mittelpunkt des Kreisbogens wird in den Parametern `CenterRow` und `CenterCol` übergeben, sein Radius in `Radius`, der Startwinkel in `AngleStart` und der Winkelbereich des Bogens relativ zum Startwinkel in `AngleExtent`. Falls `AngleExtent` > 0 wird ein Bogen entgegen dem Uhrzeigersinn erzeugt, ansonsten ein Bogen im Uhrzeigersinn. Der Radius des Rings, d.h. seine halbe Dicke, wird mit `AnnulusRadius` bestimmt.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Der Algorithmus zur Kantenextraktion ist bei `measure_pos` beschrieben. Wie dort dargestellt, können verschiedene Arten der Interpolation zur Berechnung des Grauwertprofils verwendet werden. Mit `Interpolation` = *'nearest_neighbor'* werden die Grauwerte bei der Messung durch den Grauwert des nächstgelegenen Pixels, d.h. durch konstante Interpolation, bestimmt. Für `Interpolation` = *'bilinear'* wird bilineare Interpolation verwendet und für `Interpolation` = *'bicubic'* bikubische Interpolation.

Um die eigentlichen Messungen möglichst schnell ausführen zu können, werden alle Berechnungen, die für mehrere Messungen verwendet werden können, schon im Operator `gen_measure_arc` ausgeführt. Hierzu wird eine optimierte Datenstruktur, ein sogenanntes Messobjekt, aufgebaut, das in `MeasureHandle` zurückgegeben wird. Die später bei der Messung verwendete Bildgröße muss in den Parametern `Width` und `Height` angegeben werden.

Der Systemparameter *'int_zooming'* (siehe `set_system`) beeinflusst die Genauigkeit und Geschwindigkeit der Berechnung des Messobjekts. Falls *'int_zooming'* auf *'true'* gesetzt wird, werden die internen Berechnungen in Festkommaarithmetik durchgeführt, was zu wesentlich kürzeren Laufzeiten führt. Allerdings ist hier die geometrische Genauigkeit etwas geringer. Falls *'int_zooming'* auf *'false'* gesetzt wird, werden die internen Berechnungen in Gleitkommaarithmetik durchgeführt, was zur bestmöglichen geometrischen Genauigkeit führt, die Laufzeit allerdings signifikant erhöht.

Achtung

Es ist zu beachten, dass bei Verwendung von bilinearer oder bikubischer Interpolation nicht nur das Messrechteck ins Bild passen muss, sondern auch ein Rand um das Messrechteck. Die Breite des Randes (in alle vier Richtungen) muss mindestens ein Pixel für bilineare Interpolation und mindestens zwei Pixel für bikubische Interpolation sein. Für Projektionslinien, die diese Bedingung nicht erfüllen, wird kein Grauwert berechnet. Somit kann an diesen Stellen auch keine Kante extrahiert werden.

Es ist außerdem zu beachten, dass die Mittelpunktskoordinaten des Kreisringsegments intern gerundet werden, so dass der Mittelpunkt auf dem Pixelraster liegt. Dadurch kann Konsistenz sichergestellt werden.

Parameter

▷ **`CenterRow`** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des Mittelpunktes des Bogens.
Defaultwert: 100.0
Wertevorschläge: `CenterRow` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich: 0.0 ≤ `CenterRow` ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **`CenterCol`** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des Mittelpunktes des Bogens.
Defaultwert: 100.0
Wertevorschläge: `CenterCol` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich: 0.0 ≤ `CenterCol` ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **`Radius`** (input_control) . . . . . number ⇝ *real* / integer
Radius des Bogens.
Defaultwert: 50.0
Wertevorschläge: `Radius` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich: 0.0 ≤ `Radius` ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **`AngleStart`** (input_control) . . . . . angle.rad ⇝ *real* / integer
Start des Bogens im Bogenmaß.
Defaultwert: 0.0
Wertevorschläge: `AngleStart` ∈ {-3.14159, -2.35619, -1.57080, -0.78540, 0.0, 0.78540, 1.57080, 2.35619, 3.14159}
Typischer Wertebereich: -3.14159 ≤ `AngleStart` ≤ 3.14159 (lin)
Minimale Schrittweite: 0.03142
Empfohlene Schrittweite: 0.31416

▷ **`AngleExtent`** (input_control) . . . . . angle.rad ⇝ *real* / integer
Winkelausdehnung des Bogens im Bogenmaß.
Defaultwert: 6.28318
Wertevorschläge: `AngleExtent` ∈ {-6.28318, -5.49779, -4.71239, -3.92699, -3.14159, -2.35619, -1.57080, -0.78540, 0.78540, 1.57080, 2.35619, 3.14159, 3.92699, 4.71239, 5.49779, 6.28318}
Typischer Wertebereich: -6.28318 ≤ `AngleExtent` ≤ 6.28318 (lin)
Minimale Schrittweite: 0.03142
Empfohlene Schrittweite: 0.31416
Restriktion: AngleExtent != 0.0

▷ **AnnulusRadius** (input_control) ..... number ⇝ *real / integer*
Radius (halbe Breite) des Rings.
Defaultwert: 10.0
Wertevorschläge: AnnulusRadius ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich: 0.0 ≤ AnnulusRadius ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: AnnulusRadius <= Radius

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des später zu verarbeitenden Bildes.
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768}
Typischer Wertebereich: 0 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 16

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des später zu verarbeitenden Bildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576}
Typischer Wertebereich: 0 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 16

▷ **Interpolation** (input_control) ..... string ⇝ *string*
Art der zu verwendenden Interpolation.
Defaultwert: 'nearest_neighbor'
Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **MeasureHandle** (output_control) ..... measure ⇝ *handle*
Handle des Messobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_measure_arc den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

draw_circle

Nachfolger

measure_pos, measure_pairs, fuzzy_measure_pos, fuzzy_measure_pairs, fuzzy_measure_pairing

Alternativen

edges_sub_pix

Siehe auch

gen_measure_rectangle2

Modul

1D Metrology

gen_measure_rectangle2 (: : Row, Column, Phi, Length1, Length2, Width, Height, Interpolation : MeasureHandle)

Vorbereitung der Extraktion geradliniger Kanten senkrecht zu einem Rechteck.

gen_measure_rectangle2 bereitet die Extraktion *geradliniger* Kanten, die *senkrecht* zur Hauptachse eines Rechtecks liegen, vor. Der Mittelpunkt des Rechtecks wird in den Parametern Row und Column übergeben, die Richtung der Hauptachse in Phi und die Längen der beiden Achsen, d.h. die halben Durchmesser des Rechtecks, in Length1 und Length2.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Der Algorithmus zur Kantenextraktion ist bei measure_pos beschrieben. Wie dort beschrieben, können verschiedene Arten der Interpolation zur Berechnung des Grauwertprofils verwendet werden. Mit Interpolation = *'nearest_neighbor'* werden die Grauwerte bei der Messung durch den Grauwert des nächstgelegenen Pixels, d.h. durch konstante Interpolation, bestimmt. Für Interpolation = *'bilinear'* wird bilineare Interpolation verwendet und für Interpolation = *'bicubic'* bikubische Interpolation.

Um die eigentlichen Messungen möglichst schnell ausführen zu können, werden alle Berechnungen, die für mehrere Messungen verwendet werden können, schon im Operator gen_measure_rectangle2 ausgeführt. Hierzu wird eine optimierte Datenstruktur, ein sogenanntes Messobjekt, aufgebaut, das in MeasureHandle zurückgegeben wird. Die später bei der Messung verwendete Bildgröße muss in den Parametern Width und Height angegeben werden.

Der Systemparameter *'int_zooming'* (siehe set_system) beeinflusst die Genauigkeit und Geschwindigkeit der Berechnung des Messobjekts. Falls *'int_zooming'* auf *'true'* gesetzt wird, werden die internen Berechnungen in Festkommaarithmetik durchgeführt, was zu wesentlich kürzeren Laufzeiten führt. Allerdings ist hier die geometrische Genauigkeit etwas geringer. Falls *'int_zooming'* auf *'false'* gesetzt wird, werden die internen Berechnungen in Gleitkommaarithmetik durchgeführt, was zur bestmöglichen geometrischen Genauigkeit führt, die Laufzeit allerdings signifikant erhöht.

Achtung

Es ist zu beachten, dass bei Verwendung von bilinearer oder bikubischer Interpolation nicht nur das Messrechteck ins Bild passen muss, sondern auch ein Rand um das Messrechteck. Die Breite des Randes (in alle vier Richtungen) muss mindestens ein Pixel für bilineare Interpolation und mindestens zwei Pixel für bikubische Interpolation sein. Für Projektionslinien, die diese Bedingung nicht erfüllen, wird kein Grauwert berechnet. Somit kann an diesen Stellen auch keine Kante extrahiert werden.

Es ist außerdem zu beachten, dass die Mittelpunktskoordinaten des Rechtecks intern gerundet werden, so dass der Mittelpunkt auf dem Pixelraster liegt. Dadurch kann Konsistenz sichergestellt werden.

Parameter

▷ **Row** (input_control) . . . . . rectangle2.center.y ⇝ *real* / integer
Zeilenkoordinate des Mittelpunktes des Rechtecks.
Defaultwert: 300.0
Wertevorschläge: Row ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich: 0.0 ≤ Row ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Column** (input_control) . . . . . rectangle2.center.x ⇝ *real* / integer
Spaltenkoordinate des Mittelpunktes des Rechtecks.
Defaultwert: 200.0
Wertevorschläge: Column ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich: 0.0 ≤ Column ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Phi** (input_control) . . . . . rectangle2.angle.rad ⇝ *real* / integer
Winkel der Längsachse des Rechtecks zur Horizontalen (Bogenmaß).
Defaultwert: 0.0
Wertevorschläge: Phi ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
Typischer Wertebereich: -1.178097 ≤ Phi ≤ 1.178097 (lin)
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1
Restriktion: - pi < Phi && Phi <= pi

▷ **Length1** (input_control) . . . . . rectangle2.hwidth ⇝ *real* / integer
Halbe Breite des Rechtecks.
Defaultwert: 100.0
Wertevorschläge: Length1 ∈ {3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0}
Typischer Wertebereich: 1.0 ≤ Length1 ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Length1 >= 1.0

▷ **Length2** (input_control) . . . . . rectangle2.hheight ⇝ *real* / integer
Halbe Höhe des Rechtecks.
Defaultwert: 20.0
Wertevorschläge: Length2 ∈ {1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0}
Typischer Wertebereich: 0.0 ≤ Length2 ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Length2 >= 0.0

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des später zu verarbeitenden Bildes.
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768}
Typischer Wertebereich: 0 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 16

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des später zu verarbeitenden Bildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576}
Typischer Wertebereich: 0 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 16

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Art der zu verwendenden Interpolation.
Defaultwert: 'nearest_neighbor'
Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **MeasureHandle** (output_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_measure_rectangle2 den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

draw_rectangle2

Nachfolger

measure_pos, measure_pairs, fuzzy_measure_pos, fuzzy_measure_pairs, fuzzy_measure_pairing, measure_thresh

Alternativen

edges_sub_pix

Siehe auch

gen_measure_arc

Modul

1D Metrology

measure_pairs (Image : : MeasureHandle, Sigma, Threshold, Transition, Select : RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, IntraDistance, InterDistance)

Extraktion geradliniger Kantenpaare senkrecht zu einem Rechteck oder Kreisringsegment.

measure_pairs dient zur Extraktion von *geradlinigen* Kantenpaaren, die *senkrecht* zur Hauptachse eines Rechtecks oder zu einem Kreisringsegment liegen.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Der Algorithmus von measure_pairs arbeitet analog zu measure_pos. Die Kanten werden jedoch zu Paaren mit unterschiedlichen Kantenübergängen (Vorzeichen) zusammengefasst. Falls Transition = *'positive'* werden in RowEdgeFirst und ColumnEdgeFirst Kantenpunkte mit Übergang von Dunkel nach Hell und in RowEdgeSecond und ColumnEdgeSecond mit Übergang von Hell nach Dunkel übergeben. Im Modus *'negative'* ist dies genau umgekehrt. Im Modus *'all'* bestimmt die jeweils erste gefundene Kante die Art der Paarbildung. D.h., je nach Lage des Messobjekts werden entweder hell-dunkel-hell Übergänge oder dunkel-hell-dunkel Übergänge gefunden. Dieses Vorgehen ist dann nützlich, wenn z.B. Objekte vermessen werden, die unterschiedliche Helligkeiten in Relation zum Hintergrund aufweisen.

Treten Kanten mit gleichen Vorzeichen mehrfach hintereinander auf, so wird jeweils die erste Kante der Folge zur Paarbildung verwendet. Dies kann in Anwendungen problematisch sein, in denen der Schwellenwert Threshold nicht so groß gewählt werden kann, dass aufeinanderfolgende Kanten gleichen Vorzeichens über den Schwellenwert eliminiert werden. Daher gibt es zu jedem der oben genannten Modi einen zweiten Modus, der nur die jeweils stärksten aufeinanderfolgenden aufsteigenden und absteigenden Kanten selektiert. Dieser Modus wird selektiert, indem an den jeweiligen Modus in Transition der String *'_strongest'* angehängt wird, also z.B. *'negative_strongest'*. Schließlich kann gewählt werden, welche der extrahierten Kantenpaare zurückgeliefert werden sollen. Wenn Select auf *'all'* gesetzt wird, werden alle gefundenen Paare zurückgegeben, falls Select auf *'first'* gesetzt wird nur das erste Kantenpaar und falls Select auf *'last'* gesetzt wird nur das letzte.

Die extrahierten Kanten werden als einzelne Punkte, die auf der Hauptachse des Rechtecks oder auf dem mittleren Bogen des Kreisringsegments liegen, zurückgegeben. Die zugehörigen Kantenamplituden werden in AmplitudeFirst und AmplitudeSecond zurückgeliefert. Zusätzlich wird noch der Abstand innerhalb eines Kantenpaares in IntraDistance und der Abstand von aufeinanderfolgenden Kantenpaaren in InterDistance zurückgeliefert. Dabei entspricht IntraDistance[i] dem Abstand von EdgeFirst[i] und EdgeSecond[i], während InterDistance[i] dem Abstand von EdgeSecond[i] und EdgeFirst[i+1] entspricht. D.h., das Tupel InterDistance enthält ein Element weniger als die Tupel der Kantenpaare.

Achtung

measure_pairs liefert nur dann brauchbare Ergebnisse, falls die Annahme, dass die Kanten gerade und senkrecht zur Hauptachse des Rechtecks oder zum Kreisringsegment sind, erfüllt ist. measure_pairs sollte daher z.B. nicht dazu verwendet werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, dass das Rechteck / das Kreisringsegment möglichst senkrecht zu den Bildkanten liegt. Außerdem darf Sigma nicht größer als ca. 0.5 * Length1 (siehe gen_measure_rectangle2 für Length1) sein.

Es ist zu beachten, dass measure_pairs den Definitionsbereich von Image aus Geschwindigkeitsgründen ignoriert. Falls bestimmte Bildbereiche von der Messung ausgeschlossen werden sollen, sollte ein neues Messobjekt mit entsprechend angepassten Parametern erzeugt werden.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **MeasureHandle** (input_control) . . . . . measure ⇝ *handle*
  Handle des Messobjekts.

▷ **Sigma** (input_control) ..... number ⇝ *real*
Sigma der Gaußglättung.
Defaultwert: 1.0
Wertevorschläge: `Sigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ `Sigma` ≤ 100 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Sigma >= 0.4

▷ **Threshold** (input_control) ..... number ⇝ *real*
Minimale Amplitude einer Kante.
Defaultwert: 30.0
Wertevorschläge: `Threshold` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich: 1 ≤ `Threshold` ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 2

▷ **Transition** (input_control) ..... string ⇝ *string*
Auswahl des Grauwertüberganges, der zur Paarbildung verwendet wird.
Defaultwert: 'all'
Werteliste: `Transition` ∈ {'all', 'positive', 'negative', 'all_strongest', 'positive_strongest', 'negative_strongest'}

▷ **Select** (input_control) ..... string ⇝ *string*
Auswahl der Kantenpaare.
Defaultwert: 'all'
Werteliste: `Select` ∈ {'all', 'first', 'last'}

▷ **RowEdgeFirst** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinate des Mittelpunktes der ersten Kante.

▷ **ColumnEdgeFirst** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinate des Mittelpunktes der ersten Kante.

▷ **AmplitudeFirst** (output_control) ..... real-array ⇝ *real*
Kantenamplitude der ersten Kante (mit Vorzeichen).

▷ **RowEdgeSecond** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinate des Mittelpunktes der zweiten Kante.

▷ **ColumnEdgeSecond** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinate des Mittelpunktes der zweiten Kante.

▷ **AmplitudeSecond** (output_control) ..... real-array ⇝ *real*
Kantenamplitude der zweiten Kante (mit Vorzeichen).

▷ **IntraDistance** (output_control) ..... real-array ⇝ *real*
Abstand zwischen Kanten eines Kantenpaars.

▷ **InterDistance** (output_control) ..... real-array ⇝ *real*
Abstand zwischen aufeinanderfolgenden Kantenpaaren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `measure_pairs` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_measure_rectangle2`

Nachfolger

`close_measure`

Alternativen

`edges_sub_pix`, `fuzzy_measure_pairs`, `fuzzy_measure_pairing`

Siehe auch

measure_pos, fuzzy_measure_pos

Modul

1D Metrology

measure_pos (Image : : MeasureHandle, Sigma, Threshold, Transition, Select : RowEdge, ColumnEdge, Amplitude, Distance)

Extraktion geradliniger Kanten senkrecht zu einem Rechteck oder Kreisringsegment.

measure_pos dient zur Extraktion von *geradlinigen* Kanten, die *senkrecht* zur Hauptachse eines Rechtecks oder zu einem Kreisringsegment liegen.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Der Algorithmus von measure_pos erzeugt zunächst ein eindimensionales Grauwertprofil durch Mittelung der Grauwerte entlang von Geraden senkrecht zur Hauptachse des Rechtecks oder zum Kreisringsegment. Dabei wird das Eingabebild Image an Subpixel-Positionen abgetastet, die einen ganzzahligen Zeilen- und Spalten-Abstand (im Koordinatensystem des Rechtecks) zum Mittelpunkt des Rechtecks haben. Weil die Abtastung des Bildes einige Berechnungen erfordert, die in mehreren Messungen verwendet werden können, wird der Operator gen_measure_rectangle2 b.z.w. gen_measure_arc verwendet, um diese Berechnungen nur einmal ausführen zu müssen, und somit die Geschwindigkeit von measure_pos signifikant zu erhöhen. Aufgrund der Tatsache, dass eine bessere Interpolation bei der Subpixel-Abtastung der Grauwerte zu einer besseren Genauigkeit der extrahierten Kanten führt, aber auch die Laufzeit des Operators erhöht, können verschiedene Interpolationsverfahren in gen_measure_rectangle2 und gen_measure_arc gewählt werden. (Die Interpolation beeinflusst nur Rechtecke, die nicht parallel zu den Koordinatenachsen des Bildes sind, und Kreisringsegmente.) Das Messobjekt, das mit gen_measure_rectangle2 / gen_measure_arc erzeugt wurde, wird in MeasureHandle übergeben.

Nach der Bestimmung des eindimensionalen Kantenprofils werden die subpixelgenauen Kantenpositionen durch Faltung des Profils mit den Ableitungen einer Gauß-Maske mit Standardabweichung Sigma berechnet. Auffällige Kanten können mit dem Parameter Threshold, der einen Schwellenwert für die Kantenamplitude, d.h. den Absolutbetrag der ersten Ableitung, angibt, selektiert werden. Es ist zu beachten, dass die Amplitudenwerte mit dem Faktor Sigma $\cdot \sqrt{2\pi}$ skaliert werden. Außerdem ist es möglich, nur positive Kanten, also solche, die einen Übergang von Dunkel nach Hell in der Richtung der Hauptachse des Rechtecks oder des Kreisringsegments darstellen, (Transition = *'positive'*), nur negative Kanten, also Hell-Dunkel-Übergänge, (Transition = *'negative'*) oder alle Kanten zu selektieren (Transition = *'all'*). Schließlich kann gewählt werden, welche der extrahierten Kanten zurückgeliefert werden sollen. Wenn Select auf *'all'* gesetzt wird, werden alle gefundenen Kanten zurückgegeben, falls Select auf *'first'* gesetzt wird, nur die erste Kante, und falls Select auf *'last'* gesetzt wird, nur die letzte.

Die extrahierten Kanten werden als einzelne Punkte, die auf der Hauptachse des Rechtecks oder dem mittleren Bogen des Kreisringsegments liegen, in (RowEdge, ColumnEdge) zurückgegeben. Die zugehörigen Kantenamplituden werden in Amplitude zurückgeliefert. Zusätzlich wird noch der Abstand von aufeinanderfolgenden Kantenpunkten in Distance zurückgeliefert. Dabei entspricht Distance[i] dem Abstand von Edge[i] und Edge[i+1]. D.h., das Tupel Distance enthält ein Element weniger als die Tupel RowEdge und ColumnEdge.

Achtung

measure_pos liefert nur dann brauchbare Ergebnisse, falls die Annahme, dass die Kanten gerade und senkrecht zur Hauptachse des Rechtecks oder dem Kreisringsegments sind, erfüllt ist. measure_pos sollte daher z.B. nicht dazu verwendet werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, dass das Rechteck / das Kreisringsegment möglichst senkrecht zu den Bildkanten liegt. Außerdem darf Sigma nicht größer als ca. 0.5 * Length1 (siehe gen_measure_rectangle2 für Length1) sein.

Es ist zu beachten, dass measure_pos den Definitionsbereich von Image aus Geschwindigkeitsgründen ignoriert. Falls bestimmte Bildbereiche von der Messung ausgeschlossen werden sollen, sollte ein neues Messobjekt mit entsprechend angepassten Parametern erzeugt werden.

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **MeasureHandle** (input_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

▷ **Sigma** (input_control) . . . . . number ⇝ *real*
Sigma der Gaußglättung.
Defaultwert: 1.0
Wertevorschläge: Sigma ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ Sigma ≤ 100 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Sigma >= 0.4

▷ **Threshold** (input_control) . . . . . number ⇝ *real*
Minimale Amplitude einer Kante.
Defaultwert: 30.0
Wertevorschläge: Threshold ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich: 1 ≤ Threshold ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 2

▷ **Transition** (input_control) . . . . . string ⇝ *string*
Hell/dunkel oder dunkel/hell Kante.
Defaultwert: 'all'
Werteliste: Transition ∈ {'all', 'positive', 'negative'}

▷ **Select** (input_control) . . . . . string ⇝ *string*
Auswahl der Endpunkte.
Defaultwert: 'all'
Werteliste: Select ∈ {'all', 'first', 'last'}

▷ **RowEdge** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate des Mittelpunktes der Kante.

▷ **ColumnEdge** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate des Mittelpunktes der Kante.

▷ **Amplitude** (output_control) . . . . . real-array ⇝ *real*
Kantenamplitude der Kante (mit Vorzeichen).

▷ **Distance** (output_control) . . . . . real-array ⇝ *real*
Abstand zwischen aufeinanderfolgenden Kanten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert measure_pos den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_measure_rectangle2

Nachfolger

close_measure

Alternativen

edges_sub_pix, fuzzy_measure_pos

Siehe auch

measure_pairs, fuzzy_measure_pairs, fuzzy_measure_pairing

Modul

1D Metrology

measure_projection (Image : : MeasureHandle : GrayValues)

Extraktion eines Grauwertprofils senkrecht zu einem Rechteck oder einem Kreisringsegment.

measure_projection extrahiert ein eindimensionales Grauwertprofil durch Mittelung der Grauwerte entlang von Geraden senkrecht zur Hauptachse eines Rechtecks oder zu einem Kreisringsegment. Dabei wird das Eingabebild Image an Subpixel-Positionen abgetastet, die einen ganzzahligen Zeilen- und Spalten-Abstand (im Koordinatensystem des Rechtecks) zum Mittelpunkt des Rechtecks haben. Weil die Abtastung des Bildes einige Berechnungen erfordert, die in mehreren Projektionen verwendet werden können, wird der Operator gen_measure_rectangle2 verwendet, um diese Berechnungen nur einmal ausführen zu müssen, und somit die Geschwindigkeit von measure_projection signifikant zu erhöhen. Aufgrund der Tatsache, dass eine bessere Interpolation bei der Subpixel-Abtastung der Grauwerte zu einer besseren Genauigkeit des extrahierten Grauwertprofils führt, aber auch die Laufzeit des Operators erhöht, können verschiedene Interpolationsverfahren in gen_measure_rectangle2 gewählt werden (die Interpolation beeinflusst nur Rechtecke, die nicht parallel zu den Koordinatenachsen des Bildes sind, und Kreisringsegmente). Das Messobjekt, das mit gen_measure_rectangle2 erzeugt wurde, wird in MeasureHandle übergeben.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Achtung

Es ist zu beachten, dass measure_projection den Definitionsbereich von Image aus Geschwindigkeitsgründen ignoriert. Falls bestimmte Bildbereiche von der Messung ausgeschlossen werden sollen, sollte ein neues Messobjekt mit entsprechend angepassten Parametern erzeugt werden.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **MeasureHandle** (input_control) ..... measure ⇝ *handle*
  Handle des Messobjekts.
- ▷ **GrayValues** (output_control) ..... number-array ⇝ *real*
  Grauwertprofil.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert measure_projection den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_measure_rectangle2

Nachfolger

close_measure

Alternativen

gray_projections

Modul

1D Metrology

measure_thresh (Image : : MeasureHandle, Sigma, Threshold, Select : RowThresh, ColumnThresh, Distance)

Extraktion von Punkten mit einem bestimmten Grauwert entlang eines Rechtecks oder einem Kreisringsegment.

measure_thresh bestimmt in einem eindimensionalen Grauwertprofil die Punkte mit dem Grauwert Threshold. Da das Grauwertprofil auf die Hauptachse (respektive den Bogen) des mit MeasureHandle übergebenen Messobjekts projiziert wird, entsprechen die im Grauwertprofil ermittelten Punkte bestimmten Bildkoordinaten; diese werden mit RowThresh und ColumnThresh zurückgeliefert.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Durchstößt das Grauwertprofil an mehreren Stellen den übergebenen Schwellenwert, kann mit dem Parameter `Select` ausgewählt werden, ob nur der erste (*'first'*), nur der letzte (*'last'*), der erste und der letzte (*'first_last'*) oder alle (*'all'*) Punkte zurückgeliefert werden sollen. In den letzten beiden Fällen enthält `Distance` die Abstände zwischen benachbarten Punkten.

Das zugrundegelegte Grauwertprofil wird durch Mittelung der Grauwerte entlang all der Liniensegmente gebildet, die wie folgt durch das Measure-Rechteck definiert werden:

1. Die Liniensegmente stehen senkrecht zur Hauptachse des Rechtecks,
2. sie haben einen ganzzahligen Abstand zum Rechteckmittelpunkt,
3. sie werden durch das Rechteck begrenzt.

Für jedes Liniensegment werden die Grauwerte aller Punkte, die einen ganzzahligen Abstand zur Hauptachse des Rechtecks besitzen, gemittelt. Durch Verschiebung und Drehung des Measure-Rechtecks gegenüber den Bildkoordinaten wird das Eingabebild `Image` im Allgemeinen an Subpixel-Positionen abgetastet.

Da für die Bildabtastung einige Berechnungen erforderlich sind, die in verschiedenen Operatoren Verwendung finden können und darüber hinaus mit einem gewissen Rechenaufwand verbunden sind, erfolgen sie losgelöst von den übrigen Berechnungen mit Hilfe des Operators `gen_measure_rectangle2` b.z.w. `gen_measure_arc`. Dort kann auch eine Interpolationsmethode, die für die Subpixel-Abtastung verwendet werden soll, angegeben werden.

Achtung

`measure_thresh` liefert nur dann brauchbare Ergebnisse, falls die Annahme, dass die Kanten gerade und senkrecht zur Hauptachse des Rechtecks sind, erfüllt ist. `measure_thresh` sollte daher z.B. nicht dazu verwendet werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, dass das Rechteck möglichst senkrecht zu den Bildkanten liegt. Außerdem darf `Sigma` nicht größer als ca. 0.5 * Length1 (siehe `gen_measure_rectangle2` für Length1) sein.

Es ist zu beachten, dass `measure_thresh` den Definitionsbereich von `Image` aus Geschwindigkeitsgründen ignoriert. Falls bestimmte Bildbereiche von der Messung ausgeschlossen werden sollen, sollte ein neues Messobjekt mit entsprechend angepassten Parametern erzeugt werden.

Parameter

▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **`MeasureHandle`** (input_control) ..... measure ⇝ *handle*
Handle des Messobjekts.

▷ **`Sigma`** (input_control) ..... number ⇝ *real*
Sigma der Gaußglättung.
Defaultwert: 1.0
Wertevorschläge: `Sigma` ∈ {0.0, 0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ `Sigma` ≤ 100 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Sigma >= 0.0

▷ **`Threshold`** (input_control) ..... number ⇝ *real*
Schwellenwert.
Defaultwert: 128.0
Typischer Wertebereich: 0 ≤ `Threshold` ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 1

▷ **`Select`** (input_control) ..... string ⇝ *string*
Auswahl der Ergebnispunkte.
Defaultwert: 'all'
Werteliste: `Select` ∈ {'all', 'first', 'last', 'first_last'}

▷ **`RowThresh`** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinaten der Punkte mit Schwellenwert.

▷ **`ColumnThresh`** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinaten der Punkte mit Schwellenwert.

▷ **Distance** (output_control) . . . . . real-array ⇝ *real*
Abstand zwischen aufeinanderfolgenden Punkten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `measure_thresh` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_measure_rectangle2`

Nachfolger

`close_measure`

Alternativen

`measure_pos`, `edges_sub_pix`, `measure_pairs`

Modul

1D Metrology

read_measure (: : FileName : MeasureHandle)

Einlesen eines Messobjekts aus einer Datei.

`read_measure` liest ein Messobjekt, welches mit `write_measure` geschrieben wurde, aus der Datei `FileName` ein. Die Dateiendung für ein Messobjekt in HALCON ist 'mtr'. Die im gelesenen Messobjekt vorhanden Werte werden in einem Messobjekt mit dem Handle `MeasureHandle` gespeichert.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.msr`

▷ **MeasureHandle** (output_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_measure` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`measure_pos`, `measure_pairs`

Siehe auch

`write_measure`

Modul

1D Metrology

reset_fuzzy_measure (: : MeasureHandle, SetType :)

Lösche eine Fuzzy-Funktion.

reset_fuzzy_measure löscht eine Fuzzy-Funktion des Fuzzy-Sets SetType. Diese Funktion wurde vorher mit set_fuzzy_measure gesetzt.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Parameter

▷ **MeasureHandle** (input_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

▷ **SetType** (input_control) . . . . . string ⇝ *string*
Auswahl des Fuzzy-Sets.
Defaultwert: 'contrast'
Werteliste: SetType ∈ {'position', 'position_pair', 'size', 'gray', 'contrast'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MeasureHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

set_fuzzy_measure

Nachfolger

fuzzy_measure_pos, fuzzy_measure_pairs

Siehe auch

set_fuzzy_measure, set_fuzzy_measure_norm_pair

Modul

1D Metrology

serialize_measure (: : MeasureHandle : SerializedItemHandle)

Serialisieren eines Messobjekts.

serialize_measure serialisiert die Daten eines Messobjekts (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_measure in eine Datei geschrieben werden. Das Messobjekt wird in dem Handle MeasureHandle übergeben. Das serialisierte Messobjekt wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_measure wieder deserialisiert werden.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Parameter

▷ **MeasureHandle** (input_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_measure den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MeasureHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`gen_measure_rectangle2`, `gen_measure_arc`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_measure`

Siehe auch

`read_measure`, `write_measure`

Modul

1D Metrology

set_fuzzy_measure (: : MeasureHandle, SetType, Function :)

Spezifikation einer Fuzzy-Funktion.

`set_fuzzy_measure` legt eine Fuzzy-Funktion fest, welche dem Parameter `Function` übergeben wird und in den Operatoren `fuzzy_measure_pos`, `fuzzy_measure_pairs` und `fuzzy_measure_pairing` die gefundenen Kanten bewertet. Zu diesem Zweck können Bewertungskriterien für verschiedene Merkmale von Kanten und Kantenpaaren mit je einer Funktion beschrieben werden. Ein auf diese Weise charakterisiertes Merkmal wird Fuzzy-Set genannt. Wird keine Funktion für ein Fuzzy-Set definiert, so wird dieses Merkmal nicht für die Bewertung der gefundenen Kantenpaare herangezogen. Durch erneutes Setzen eines Fuzzy-Sets lässt sich hingegen die zuvor definierte Fuzzy-Funktion überschrieben. Gelöscht wird eine für ein Fuzzy-Set definierte Funktion mit Hilfe des `reset_fuzzy_measure` Operators.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Es können Funktionen für fünf unterschiedliche Typen von Fuzzy-Sets definiert werden, wobei die Unterarten eines Sets nur gegeneinander exklusiv gesetzt werden können:

- *'contrast'* verwendet eine Fuzzy-Funktion, welche die Amplituden der extrahierten Kanten bewertet. Im Falle von Kantenpaaren ermittelt sich die Fuzzy-Bewertung durch das geometrische Mittel der Einzelbewertungen beider Kanten.
- Die Fuzzy-Funktion des Sets *'position'* bewertet den gerichteten Abstand p von jeder gefundenen Kante zum Referenzpunkt des Messobjekts, welches mit `gen_measure_arc` oder `gen_measure_rectangle2` erzeugt wird. Im Falle der Extraktion von Kantenpaare mit `fuzzy_measure_pairs`/`fuzzy_measure_pairing` setzt das Ergebnis der Bewertung aus dem geometrischen Mittel der Bewertungen der zugehörigen beiden Einzelkanten zusammen. *'position'* legt den Referenzpunkt an den Anfang, während *'position_center'* b.z.w. *'position_end'* den Referenzpunkt auf die Mitte b.z.w. das Ende des eindimensionalen Grauwertprofils setzten. Wenn die Bewertung von der Position des zu messenden Objektes entlang des Grauwertprofils abhängt, lässt sich der Referenzpunkt mit Hilfe von *'position_first_edge'* / *'position_last_edge'* auf die erste/letzte extrahierte Kante setzten.
- Ähnlich zu *'position'* bewertet *'position_pair'* den Abstand jeden Kantenpaares zu dem Referenzpunkt. Die Position eines Paares wird dabei durch die Mitte der beiden Einzelkanten definiert. Der Bezugspunkt des Messobjekts kann dabei entsprechend mit *'position_pair_center'*, *'position_pair_end'* und *'position_first_pair'*, *'position_last_pair'* beeinflusst werden. Im Gegensatz zu *'position'* wird dieses Fuzzy-Set nur von den Operatoren `fuzzy_measure_pairs`/ `fuzzy_measure_pairing` verwendet.

- *'size'* bezeichnet ein Fuzzy-Set, das den Abstand zwischen den beiden Kanten eines Paares in Pixeln bewertet. Die Verarbeitungsgeschwindigkeit von fuzzy_measure_pairs / fuzzy_measure_pairing lässt sich erhöhen, indem eine obere Grenze x_max für die Breite festgelegt und die korrespondierende Ordinate auf den Wert 0.0 gesetzt wird. Dadurch wird erreicht, dass nicht mehr alle möglichen Paare betrachtet werden müssen.
- Die Fuzzy-Funktion des Sets *'gray'* gewichtet den Mittelwert des projizierten Grauwertprofils zwischen den beiden Kanten eines Paares. Dieses Fuzzy-Set wird daher nur von den Operatoren fuzzy_measure_pairs und/ fuzzy_measure_pairing unterstützt.

Eine normalisierte Fuzzy-Funktion wird durch eine stückweise lineare Funktion beschrieben, welche aus mindestens zwei Stützstellen besteht, die in aufsteigender Reihenfolge ihres Abszissenwertes x sortiert sind. Die Ordinate y beschreibt die Gewichtung zum korrespondierenden Merkmalwert x und muss in dem Bereich von 0.0 $\leq$ y $\leq$ 1.0 liegen. Außerhalb des Funktionsintervalls, welches durch den größten und kleinsten Abszissenwert x festgelegt ist, wird die Fuzzy-Funktion konstant mit den Ordinatenwerten der Intervallgrenzen fortgeführt. Zwischen den Stützstellen wird der Funktionswert linear interpoliert. Solche Fuzzy-Funktionen können mit Hilfe des create_funct_1d_pairs Operators erzeugt werden.

Wird mehr als ein Fuzzy-Set definiert setzt sich für fuzzy_measure_pos / fuzzy_measure_pairs / fuzzy_measure_pairing die resultierende Fuzzy-Bewertung aus dem geometrischen Mittel der Bewertungen aus den einzelnen Sets zusammen.

Parameter

▷ **MeasureHandle** (input_control) ..... measure ⇝ *handle*
Handle des Messobjekts.

▷ **SetType** (input_control) ..... string ⇝ *string*
Auswahl des Fuzzy-Sets.
Defaultwert: 'contrast'
Werteliste: SetType ∈ {'position', 'position_center', 'position_end', 'position_first_edge', 'position_last_edge', 'position_pair_center', 'position_pair_end', 'position_first_pair', 'position_last_pair', 'size', 'gray', 'contrast'}

▷ **Function** (input_control) ..... function_1d ⇝ *real / integer*
Fuzzy-Funktion.

Beispiel

```
* how to use a fuzzy function
* ...
gen_measure_rectangle2 (50, 100, 0, 200, 100, 512, 512, 'nearest_neighbor', \
                        MeasureHandle)
* create a generalized fuzzy function to evaluate edge pairs
* * (30% uncertainty).
create_funct_1d_pairs ([0.7,1.0,1.3], [0.0,1.0,0.0], SizeFunction)
* and transform it to expected size of 13.45 pixels
transform_funct_1d (SizeFunction, [1.0,0.0,13.45,0.0], TransformedFunction)
set_fuzzy_measure (MeasureHandle, 'size', TransformedFunction)

fuzzy_measure_pairs (Image, MeasureHandle, 1, 30, 0.5, 'all', RowEdgeFirst, \
                     ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond, \
                     ColumnEdgeSecond, AmplitudeSecond, RowEdgeCenter, \
                     ColumnEdgeCenter, FuzzyScore, IntraDistance, \
                     InterDistance)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MeasureHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`gen_measure_arc`, `gen_measure_rectangle2`, `create_funct_1d_pairs`, `transform_funct_1d`

Nachfolger

`fuzzy_measure_pos`, `fuzzy_measure_pairs`

Alternativen

`set_fuzzy_measure_norm_pair`

Siehe auch

`reset_fuzzy_measure`

Modul

1D Metrology

set_fuzzy_measure_norm_pair (: : MeasureHandle, PairSize, SetType, Function :)

Spezifikation einer normierte Fuzzy-Funktion für Kantenpaare.

`set_fuzzy_measure_norm_pair` legt eine normierte Fuzzy-Funktion fest, welche dem Parameter `Function` übergeben wird und in den Operatoren `fuzzy_measure_pos`, `fuzzy_measure_pairs` und `fuzzy_measure_pairing` die gefundenen Kantenpaare bewertet. Zu diesem Zweck können Bewertungskriterien für verschiedene Merkmale von Kantenpaaren mit je einer Funktion beschrieben werden. Ein auf diese Weise charakterisiertes Merkmal wird Fuzzy-Set genannt. Wird keine Funktion für ein Fuzzy-Set definiert, so wird dieses Merkmal nicht für die Bewertung der gefundenen Kantenpaare herangezogen. Durch erneutes Setzen eines Fuzzy-Sets lässt sich hingegen die zuvor definierte Fuzzy-Funktion überschrieben. Gelöscht wird eine für ein Fuzzy-Set definierte Funktion mit Hilfe des `reset_fuzzy_measure` Operators. Im Unterschied zu `set_fuzzy_measure`, muss die Abszisse x der Funktion relativ zur beabsichtigten Breite s (gegeben durch `PairSize`) der Kantenpaare festgelegt werden. Damit erreicht man eine allgemeine Verwendung der definierten Funktionen.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Es können Funktionen für drei unterschiedliche Typen von Fuzzy-Sets definiert werden, wobei die Unterarten eines Sets nur gegeneinander exklusiv gesetzt werden können:

- *'size'* bezeichnet ein Fuzzy-Set, das den absoluten Abstand d zwischen den beiden Kanten eines Paares in Pixeln bewertet.

  $$x = \frac{d}{s}(x \geq 0) \ .$$

  .

  Die Verarbeitungsgeschwindigkeit von `fuzzy_measure_pairs` / `fuzzy_measure_pairing` lässt sich erhöhen, indem eine obere Grenze x_max für die Breite festgelegt und die korrespondierende Ordinate auf den Wert 0.0 gesetzt wird. Dadurch wird erreicht, dass nicht mehr alle möglichen Paare betrachtet werden müssen. Dieses Fuzzy-Set kann aber auch als Differenz der Breiten (*'size_diff'*)

  $$x = \frac{s - d}{s}(x \leq 1)$$

  oder ihren Absolutbetrag (*'size_abs_diff'*)

  $$x = \frac{|s - d|}{s}(0 \leq x \leq 1) \ .$$

  .

  definiert werden. Dieses Fuzzy-Set wird nur von den Operatoren `fuzzy_measure_pairs` und `fuzzy_measure_pairing` verwendet.

- Die Fuzzy-Funktion des Sets *'position'* bewertet den vorzeichenbehafteten Abstand p von jeder gefundenen Kante zum Referenzpunkt des Messobjekts, welches mit gen_measure_arc oder gen_measure_rectangle2 erzeugt wird:

$$x = \frac{p}{s} \ .$$

.

Im Falle der Extraktion von Kantenpaaren mit fuzzy_measure_pairs/ fuzzy_measure_pairing setzt das Ergebnis der Bewertung aus dem geometrischen Mittel der Bewertungen der zugehörigen beiden Einzelkanten zusammen. *'position'* legt den Referenzpunkt an den Anfang, während *'position_center'* b.z.w. *'position_end'* den Referenzpunkt auf die Mitte b.z.w. das Ende des eindimensionalen Grauwertprofils setzten. Wenn die Bewertung von der Position des zu messenden Objektes entlang des Grauwertprofils abhängt, lässt sich der Referenzpunkt mit Hilfe von *'position_first_edge'* / *'position_last_edge'* auf die erste/letzte extrahierte Kante setzten.

- Ähnlich zu *'position'* bewertet *'position_pair'* den vorzeichenbehafteten Abstand jeden Kantenpaares zu dem Referenzpunkt. Die Position eines Paares wird dabei durch die Mitte der beiden Einzelkanten definiert. Der Bezugspunkt des Messobjekts kann dabei entsprechend mit *'position_pair_center'*, *'position_pair_end'* und *'position_first_pair'*, *'position_last_pair'* beeinflusst werden. Im Gegensatz zu *'position'* wird dieses Fuzzy-Set nur von den Operatoren fuzzy_measure_pairs/fuzzy_measure_pairing verwendet.

Eine normalisierte Fuzzy-Funktion wird durch eine stückweise lineare Funktion beschrieben, welche aus mindestens zwei Stützstellen besteht, die in aufsteigender Reihenfolge ihres Abszissenwertes x sortiert sind. Die Ordinate y beschreibt die Gewichtung zum korrespondierenden Merkmalwert x und muss in dem Bereich von 0.0 $\leq$ y $\leq$ 1.0 liegen. Außerhalb des Funktionsintervalls, welches durch den größten und kleinsten Abszissenwert x festgelegt ist, wird die Fuzzy-Funktion konstant mit den Ordinatenwerten der Intervallgrenzen fortgeführt. Solche Fuzzy-Funktionen können mit Hilfe des create_funct_1d_pairs Operators erzeugt werden.

Wird mehr als ein Fuzzy-Set definiert setzt sich für fuzzy_measure_pos / fuzzy_measure_pairs / fuzzy_measure_pairing die resultierende Fuzzy-Bewertung aus dem geometrischen Mittel der Bewertungen aus den einzelnen Sets zusammen.

Parameter

▷ **MeasureHandle** (input_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

▷ **PairSize** (input_control) . . . . . number ⇝ *real* / integer
Gesuchte Breite der Kantenpaare.
Defaultwert: 10.0
Wertevorschläge: PairSize ∈ {4.0, 6.0, 8.0, 10.0, 15.0, 20.0, 30.0}
Typischer Wertebereich: 0.0 ≤ PairSize
Minimale Schrittweite: 0.1
Empfohlene Schrittweite: 1.0

▷ **SetType** (input_control) . . . . . string ⇝ *string*
Auswahl des Fuzzy-Sets.
Defaultwert: 'size_abs_diff'
Werteliste: SetType ∈ {'size', 'size_diff', 'size_abs_diff', 'position', 'position_center', 'position_end', 'position_first_edge', 'position_last_edge', 'position_pair_center', 'position_pair_end', 'position_first_pair', 'position_last_pair'}

▷ **Function** (input_control) . . . . . function_1d ⇝ *real* / integer
Fuzzy-Funktion.

Beispiel

```
* how to use a fuzzy function
* ...
gen_measure_rectangle2 (50, 100, 0, 200, 100, 512, 512, 'nearest_neighbor', \
                        MeasureHandle)
* create a generalized fuzzy function to evaluate edge pairs
* * (30% uncertainty).
create_funct_1d_pairs ([0.7,1.0,1.3], [0.0,1.0,0.0], SizeFunction)
* and set it for an expected pair size of 13.45 pixels
```

```
set_fuzzy_measure_norm_pair (MeasureHandle, 13.45, 'size', SizeFunction)

fuzzy_measure_pairs (Image, MeasureHandle, 1, 30, 0.5, 'all', RowEdgeFirst, \
                     ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond, \
                     ColumnEdgeSecond, AmplitudeSecond, RowEdgeCenter, \
                     ColumnEdgeCenter, FuzzyScore, IntraDistance, \
                     InterDistance)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MeasureHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`gen_measure_arc`, `gen_measure_rectangle2`, `create_funct_1d_pairs`

Nachfolger

`fuzzy_measure_pairs`, `fuzzy_measure_pairing`

Alternativen

`transform_funct_1d`, `set_fuzzy_measure`

Siehe auch

`reset_fuzzy_measure`

Modul

1D Metrology

`translate_measure ( : : MeasureHandle, Row, Column : )`

Translation eines Messobjekts.

`translate_measure` verschiebt den Referenzpunkt des durch `MeasureHandle` angegebenen Messobjekts zum Punkt (`Row`, `Column`). Falls das Messobjekt und das verschobene Messobjekt vollständig im Bild liegen, wird das Messobjekt in effizienter Weise an den neuen Referenzpunkt angepasst. Ansonsten wird das Messobjekt mit den bei der Generierung angegebenen Daten am neuen Referenzpunkt mit `gen_measure_rectangle2` oder `gen_measure_arc` neu generiert.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Parameter

▷ **`MeasureHandle`** (input_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

▷ **`Row`** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des neuen Referenzpunktes.
Defaultwert: 50.0
Wertevorschläge: `Row` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich: 0.0 ≤ `Row` ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Column** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des neuen Referenzpunktes.
Defaultwert: 100.0
Wertevorschläge: Column ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich: 0.0 ≤ Column ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert translate_measure den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MeasureHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

gen_measure_rectangle2, gen_measure_arc

Nachfolger

measure_pos, measure_pairs, fuzzy_measure_pos, fuzzy_measure_pairs, fuzzy_measure_pairing, measure_thresh

Alternativen

gen_measure_rectangle2, gen_measure_arc

Siehe auch

close_measure

Modul

1D Metrology

write_measure (: : MeasureHandle, FileName :)

Schreiben eines Messobjekts in eine Datei.

write_measure schreibt ein Messobjekt, welches z.B. von gen_measure_rectangle2 erzeugt wurde, in die Datei FileName. Das Messobjekt ist durch das Handle MeasureHandle definiert. Das Messobjekt kann mit read_measure wieder eingelesen werden. Die Dateiendung für ein Messobjekt in HALCON ist 'mtr'.

Das Konzept der 1D-Vermessung ist bei der Einleitung zu Kapitel 1D-Vermessung beschrieben.

Parameter

▷ **MeasureHandle** (input_control) . . . . . measure ⇝ *handle*
Handle des Messobjekts.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: .msr

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_measure den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_measure_rectangle2`, `gen_measure_arc`

Siehe auch

`read_measure`

Modul

1D Metrology

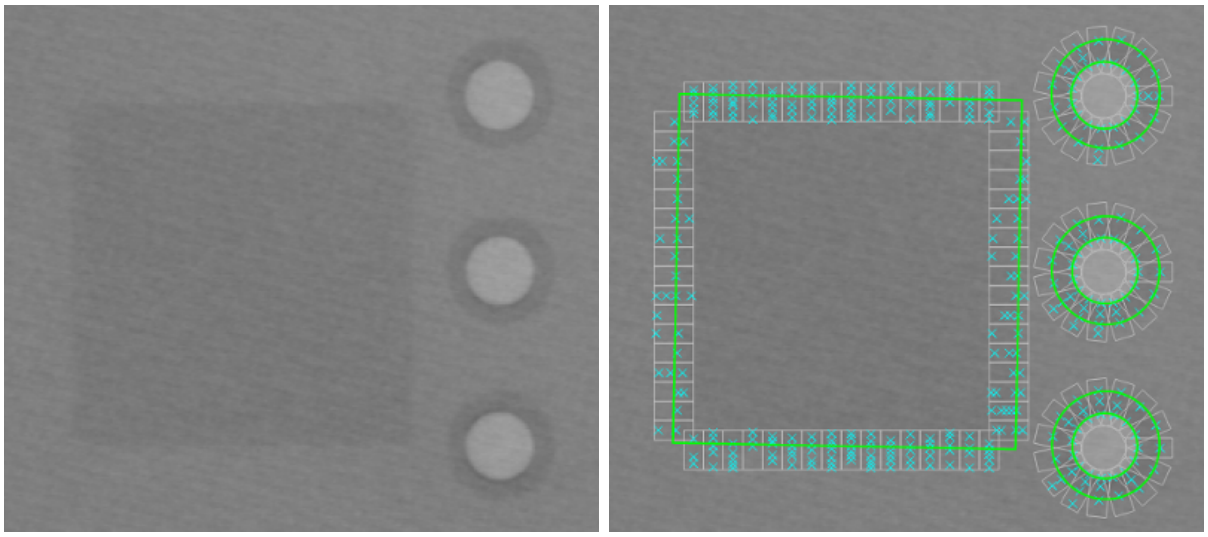
Kapitel 2

2D-Messtechnik

Dieses Kapitel beschreibt Operatoren für die 2D-Messtechnik.

Konzept der 2D-Messtechnik

Mit 2D-Messtechnik können Objekte einer bestimmten geometrischen Form vermessen werden. Die dafür verfügbaren geometrischen Formen umfassen Kreise, Ellipsen, Rechtecke und Linien. Es müssen genäherte Werte für die Positionen, Orientierungen und geometrischen Formen der zu vermessenden Objekte vorhanden sein. Dann werden in der Nähe der Umrandungen der genäherten Objekte im Bild die tatsächlichen Kanten der Objekte lokalisiert. Die Parameter der geometrischen Formen werden dann so angepasst, dass sie optimal zu den Bilddaten passen. Die Ergebnisse der Messungen sind die optimierten Parameter.



(1) (2)

Die geometrischen Formen in (1) werden mit 2D-Messtechnik vermessen (2): Es wird ein Messmodel mit 4 Metrology-Messobjekten erstellt (blaue Konturen). Unter Benutzung der Kantenpunkte (Kreuze in Cyan), die innerhalb der Messregionen (graue Rechtecke) gefunden wurden, werden die geometrischen Formen (grüne Konturen) angepasst und ihre Parameter können abgefragt werden. Wie man für die Metrology-Messobjekte vom Typ Kreis sieht, können auch mehrere Instanzen pro Objekt gefunden werden. Dieses Bild stammt aus dem Beispielprogramm apply_metrology_model.hdev.

Die genäherten Werte für die Formparameter eines Objekts sowie Parameter zur Steuerung der Messung werden in einer Datenstruktur gespeichert, die als Metrology-Messobjekt bezeichnet wird. Die Kanten des Objekts im Bild werden in den sogenannten Messregionen gesucht. Diese sind rechteckige Regionen, deren Hauptachsen senkrecht zum Rand der genäherten Objekte liegen. Parameter, die die Größe und Verteilung der Messregionen beeinflussen, werden zusammen mit den genäherten Formparametern für jedes Objekt festgelegt. Bei der Messung wird die Anpassung der geometrischen Form an die gefundenen Kantenpositionen mittels eines RANSAC-Algorithmus

vorgenommen. Alle Messobjekte, alle weiteren für die Messung notwendigen Informationen und die Ergebnisse der Messungen werden im sogenannten Messmodel gespeichert.

Im Folgenden werden die einzelnen Schritte, die zur Benutzung der 2D-Messtechnik benötigt werden, kurz beschrieben.

Erstellen der Datenstruktur des Messmodells: Als erstes muss das Messmodell erstellt werden. Dies geschieht mit dem Operator

- `create_metrology_model`.

Das Messmodell dient als Container für ein oder mehrere Metrology-Messobjekte. Für eine effiziente Messung sollte dann die Größe des Bildes, auf dem die Messungen durchgeführt werden, festgelegt werden. Dies erfolgt mit dem Operator

- `set_metrology_model_image_size`

Bereitstellen der genäherten Werte: Anschließend werden die Metrology-Messobjekte zum Messmodell hinzugefügt. Jedes Metrology-Messobjekt besteht aus genäherten Parametern für die Form des zugehörigen Objekts im Bild und aus Parametern, die die Messungen steuern. Die Parameter, die die Messungen steuern enthalten z.B. Parameter, die die Dimensionen und die Verteilung der Messregionen festlegen. Des weiteren können verschiedene generische Parameter für jedes Metrology-Messobjekt festgelegt werden. Die Metrology-Messobjekte werden mit den folgenden Operatoren definiert:

- `add_metrology_object_circle_measure` für Kreise,
- `add_metrology_object_ellipse_measure` für Ellipsen,
- `add_metrology_object_rectangle2_measure` für Rechtecke und
- `add_metrology_object_line_measure` für Linien.
- `add_metrology_object_generic` erlaubt es, Metrology-Messobjekte verschiedener Formen (Kreis, Ellipse, etc.) mit einem einzigen Operator zu definieren.

Um die definierten Metrology-Messobjekte visuell zu prüfen, können die XLD-Konturen ihrer Umrandungen mit dem Operator `get_metrology_object_model_contour` abgerufen werden. Um die Messregionen visuell zu überprüfen, kann auf die XLD-Konturen ihrer Umrandungen mit dem Operator `get_metrology_object_measures` zugegriffen werden.

Ändern der Modellparameter: Wenn eine Kamerakalibrierung ausgeführt wurde, werden die Kameraparameter und die Pose des Weltkoordinatensystems zum Messmodel hinzugefügt. Dies erfolgt über den Operator

- `set_metrology_model_param`.

In diesem Fall werden die Ergebnisse der Messungen, die mit `get_metrology_object_result` abgerufen werden, im Weltkoordinatensystem zurückgegeben. Weiterhin kann das Referenzsystem, in dem die Metrology-Messobjekte vorliegen, mit `set_metrology_model_param` geändert werden.

Ändern der Objektparameter: Viele Parameter können beim Hinzufügen der Metrology-Messobjekte gesetzt werden. Einige davon können anschließend noch mithilfe des Operators

- `set_metrology_object_param`.

geändert werden.

Ausrichten des Messmodells (Alignment): Der Operator

- `align_metrology_model`

verschiebt und rotiert das Messmodell vor einer Messung, so dass es über dem zu messenden Objekt im aktuellen Bild liegt. Diese Modellausrichtung ist temporär und wird durch die nächste ersetzt. Die Parameter der einzelnen Objekte werden nicht verändert. Typischerweise werden die für die Modellausrichtung benötigten Parameter über formbasiertes Matching ermittelt.

Durchführen der Messungen: Die eigentlichen Messungen im Bild erfolgen mit dem Operator

- apply_metrology_model.

Dieser findet die Kanten innerhalb der Messregionen und passt die festgelegten geometrischen Formen an die Kantenpositionen unter Verwendung eines RANSAC-Algorithmus an. Die Kanten werden intern mit den Operatoren measure_pos oder fuzzy_measure_pos (siehe auch Kapitel 1D-Vermessung) ermittelt. Der letztere verwendet Fuzzy-Funktionen und wird nur verwendet, wenn vor der Messung mindestens eine Fuzzy-Funktion mithilfe von set_metrology_object_fuzzy_param gesetzt wurde. Wenn mehr als eine Instanz der zurückgegebenen Objektform benötigt wird (siehe Bild oben), muss der generische Parameter *'num_instances'* auf die Anzahl benötigter Instanzen gesetzt werden. Der Parameter kann beim Setzen der einzelnen Metrology-Messobjekte oder später mit dem Operator set_metrology_object_param gesetzt werden.

Zugriff auf die Ergebnisse: Nach der Messung kann auf die Ergebnisse zugegriffen werden. Die angepassten geometrischen Formen der Objekte werden mit dem Operator

- get_metrology_object_result

abgefragt. Auf die Zeilen- und Spaltenkoordinaten der gefundenen Kanten kann mit dem Operator

- get_metrology_object_measures

zugegriffen werden. Die Zeilen- und Spaltenkoordinaten der Kanten, die tatsächlich für die Berechnung der Ergebnisse benutzt wurden, können ebenfalls mit get_metrology_object_result abgefragt werden. Eine Visualisierung der gemessenen Formen als XLD Konturen erhält man mit

- get_metrology_object_result_contour.

Weitere Operatoren

Zusätzlich zu den oben genannten Operatoren kann das Messmodell mit copy_metrology_model kopiert werden. Um ein Messmodell in eine Datei zu schreiben ist der Operator write_metrology_model zu verwenden. Das Model kann unter Verwendung des Operators read_metrology_model wieder eingelesen werden. Mit serialize_metrology_model bzw. deserialize_metrology_model kann das Messmodell serialisiert oder deserialisiert werden.

Die im Messmodell vorhandenen Informationen können auf verschiedene Arten abgefragt werden. Zum Beispiel erhält man die Indizes der Metrology-Messobjekte mit get_metrology_object_indices. Parameter, die für das ganze Messmodell zutreffen, können mit get_metrology_model_param abgefragt werden. Zum Abfragen der Fuzzy-Parameter des Messmodells (wenn vorhanden) steht get_metrology_object_fuzzy_param zur Verfügung. Die Anzahl der Instanzen der Metrology-Messobjekte eines Messmodells erhält man mit get_metrology_object_num_instances. Und die aktuelle Konfiguration des Messmodells kann mit get_metrology_object_param abgefragt werden.

Außerdem können alle Parameter eines Messmodells mit reset_metrology_object_param zurückgesetzt werden. Sollen nur eventuell vorhandene Fuzzy-Parameter und Fuzzy-Funktionen zurückgesetzt werden, kann reset_metrology_object_fuzzy_param benutzt werden.

Glossar

Im Folgenden werden die wichtigsten Begriffe, die im Umfeld der 2D-Messtechnik verwendet werden, beschrieben:

Messmodell Datenstruktur, die alle Metrology-Messobjekte, alle Informationen, die für die Messung gebraucht werden, sowie die Messergebnisse enthält.

Metrology-Messobjekt Datenstruktur für das Objekt, das mit 2D-Messtechnik vermessen werden soll. Das Metrology Messobjekt ist durch eine bestimmte geometrische Form repräsentiert für deren Formparameter Näherungswerte bekannt sind. Außerdem enthält es Parameter zur Steuerung der Messung, z.B. Parameter, die die Dimensionen und die Verteilung der Messregionen bestimmen.

Messregionen Rechteckige Regionen, die senkrecht zum Rand der genäherten Objekte angeordnet sind. Innerhalb dieser Regionen werden Kanten extrahiert, die benutzt werden, um die exakten Formparameter der Metrology-Messobjekte zu ermitteln.

Zurückgegebene Instanz eines Metrology-Messobjekts Für jedes Metrology-Messobjekt können unterschiedliche Instanzen des Objekts durch die Messung zurückgegeben werden, z.B. wenn in der Nähe der Umrandung der genäherten Form verschiedene parallele Strukturen der gleichen Form vorhanden sind. Die Reihenfolge der zurückgegebenen Instanzen ist zufällig, d.h. die Reihenfolge stellt kein Maß für die Qualität der Anpassung dar.

Weiterführende Information

Weitere Details zur 2D-Messtechnik finden sich im „Solution Guide on 2D Measuring".

add_metrology_object_circle_measure (: : MetrologyHandle, Row, Column, Radius, MeasureLength1, MeasureLength2, MeasureSigma, MeasureThreshold, GenParamName, GenParamValue : Index)

Hinzufügen eines Kreises oder Kreisbogens zu einem Messmodell.

add_metrology_object_circle_measure fügt ein Metrology-Messobjekt vom Typ Kreis oder Kreisbogen zu einem Messmodell hinzu und bereitet die rechteckigen Messregionen vor. Das Handle des Messmodells muss in MetrologyHandle übergeben werden.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Die genäherte geometrische Form des Metrology-Messobjekts vom Typ Kreis wird durch seinen Mittelpunkt (Row, Column) und den Radius beschrieben. Die Messregionen liegen *senkrecht* zum Kreisrand. Die halben Kantenlängen der Messregionen senkrecht und tangential zum Kreis werden jeweils in MeasureLength1 und in MeasureLength2 gesetzt. Die Mittelpunkte der Messregionen liegen auf dem Rand des Kreises. Der Parameter MeasureSigma legt eine Standardabweichung für die Glättung der Grauwerte im Bild durch den Operator apply_metrology_model fest. Auffällige Kanten können mit dem Parameter MeasureThreshold, der einen Schwellenwert für die Kantenamplitude, d.h. den Absolutbetrag der ersten Ableitung, angibt, selektiert werden. Der Operator add_metrology_object_circle_measure gibt den Index des hinzugefügten Metrology-Messobjekts in dem Parameter Index zurück.

Des weiteren können in GenParamName und GenParamValue generische Parameter angepasst werden. Die folgenden generischen Parameter GenParamName und die dazugehörigen Werte GenParamValue zur Festlegung eines Kreisbogens sind verfügbar:

'start_phi': Der Parameter legt den Anfangswinkel des Kreisbogens fest. Um einen geschlossenen Kreis zu erhalten, müssen der Wert des Parameters *'start_phi'* auf 0 und der Wert des Parameters *'end_phi'* auf 2π gesetzt sein (bei positivem Umlaufsinn). Die Eingabewerte werden automatisch auf das Intervall $[0, 2\pi]$ abgebildet.

Wertevorschläge: 0.0, 0.78, 6.28318

Defaultwert: 0.0

'end_phi': Der Parameter legt den Endwinkel des Kreisbogens fest. Um einen geschlossenen Kreis zu erhalten, müssen der Wert des Parameters *'start_phi'* auf 0 und der Wert des Parameters *'end_phi'* auf 2π gesetzt sein (bei positivem Umlaufsinn). Die Eingabewerte werden automatisch auf das Intervall $[0, 2\pi]$ abgebildet.

Wertevorschläge: 0.0, 0.78, 6.28318

Defaultwert: 6.28318

'point_order': Der Parameter legt die Richtung des Kreisbogens fest. Wird der Wert auf *'positive'* gesetzt, ist der Kreisbogen zwischen *'start_phi'* und *'end_phi'* im mathematisch positiven Sinn (gegen den Uhrzeigersinn) definiert. Wird der Wert auf *'negative'* gesetzt, ist der Kreisbogen zwischen *'start_phi'* und *'end_phi'* im mathematisch negativen Sinn (im Uhrzeigersinn) definiert.

Wertevorschläge: 'positive', 'negative'

Defaultwert: 'positive'

Weitere generische Parameter sind alle generischen Parameter, die im Operator set_metrology_object_param zur Verfügung stehen. In den meisten Fällen reichen die Standardwerte aus, so dass keine Anpassung nötig ist.

Parameter

▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **Row** (input_control) . . . . . circle.center.y(-array) ⇝ *real* / integer
Zeilenkoordinate des Mittelpunkts des Kreises oder des Kreisbogens.

▷ **Column** (input_control) . . . . . circle.center.x(-array) ⇝ *real* / integer
Spaltenkoordinate des Mittelpunkts des Kreises oder des Kreisbogens.

▷ **Radius** (input_control) . . . . . circle.radius(-array) ⇝ *real* / integer
Radius des Kreises oder Kreisbogens.

▷ **MeasureLength1** (input_control) . . . . . number ⇝ *real* / integer
Halbe Länge der Messregionen senkrecht zum Rand.
Defaultwert: 20.0
Wertevorschläge: `MeasureLength1` ∈ {10.0, 20.0, 30.0}
Typischer Wertebereich: 1.0 ≤ MeasureLength1
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: MeasureLength1 < Radius

▷ **MeasureLength2** (input_control) . . . . . number ⇝ *real* / integer
Halbe Länge der Messregionen tangential zum Rand.
Defaultwert: 5.0
Wertevorschläge: `MeasureLength2` ∈ {3.0, 5.0, 10.0}
Typischer Wertebereich: 1.0 ≤ MeasureLength2
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **MeasureSigma** (input_control) . . . . . number ⇝ *real* / integer
Sigma der Gaußfunktion für die Glättung.
Defaultwert: 1.0
Wertevorschläge: `MeasureSigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ MeasureSigma
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: 0.4 <= MeasureSigma && MeasureSigma <= 100

▷ **MeasureThreshold** (input_control) . . . . . number ⇝ *real* / integer
Minimale Amplitude einer Kante.
Defaultwert: 30.0
Wertevorschläge: `MeasureThreshold` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich: 1 ≤ `MeasureThreshold` ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 2

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'distance_threshold', 'end_phi', 'instances_outside_measure_regions', 'max_num_iterations', 'measure_distance', 'measure_interpolation', 'measure_select', 'measure_transition', 'min_score', 'num_instances', 'num_measures', 'point_order', 'rand_seed', 'start_phi'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {1, 2, 3, 4, 5, 10, 20, 'all', 'true', 'false', 'first', 'last', 'positive', 'negative', 'uniform', 'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **Index** (output_control) . . . . . integer ⇝ *integer*
Index des erstellten Metrology-Messobjekts.

Beispiel

```
read_image (Image, 'rings_and_nuts')
create_metrology_model (MetrologyHandle)
```

```
get_image_size (Image, Width, Height)
set_metrology_model_image_size (MetrologyHandle, Width, Height)
add_metrology_object_circle_measure (MetrologyHandle, 120, 130, 35, 10, 2, \
                                     1, 30, ['measure_distance'], [40], Index)
apply_metrology_model (Image, MetrologyHandle)
get_metrology_object_result (MetrologyHandle, Index, 'all', 'result_type', \
                             'all_param', Circle)
get_metrology_object_result_contour (Contour, MetrologyHandle, Index, \
                                     'all', 1.5)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_metrology_object_circle_measure` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`set_metrology_model_image_size`

Nachfolger

`align_metrology_model`, `apply_metrology_model`

Alternativen

`add_metrology_object_generic`

Siehe auch

`get_metrology_object_model_contour`, `set_metrology_model_param`, `add_metrology_object_ellipse_measure`, `add_metrology_object_line_measure`, `add_metrology_object_rectangle2_measure`

Modul

2D Metrology

add_metrology_object_ellipse_measure (: : MetrologyHandle, Row, Column, Phi, Radius1, Radius2, MeasureLength1, MeasureLength2, MeasureSigma, MeasureThreshold, GenParamName, GenParamValue : Index)

Hinzufügen einer Ellipse oder eines Ellipsenbogens zu einem Messmodell.

`add_metrology_object_ellipse_measure` fügt ein Metrology-Messobjekt vom Typ Ellipse oder einen Ellipsenbogen zu einem Messmodell hinzu und bereitet die rechteckigen Messregionen vor. Das Handle des Messmodells muss in `MetrologyHandle` übergeben werden.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Die genäherte geometrische Form des Metrology-Messobjekts vom Typ Ellipse wird durch ihren Mittelpunkt (`Row`, `Column`), die Orientierung der Hauptachse `Phi` und die Länge der großen Halbachse `Radius1` bzw. der kleinen Halbachse `Radius2` beschrieben. Der Eingabewert `Phi` wird automatisch auf das Intervall $]-\pi,\pi]$ abgebildet. Die Messregionen liegen *senkrecht* zum Ellipsenrand. Die halbe Kantenlänge der Messregionen senkrecht und tangential zur Ellipse werden jeweils in `MeasureLength1` und in `MeasureLength2` gesetzt. Die Mittelpunkte der Messregionen liegen auf dem Rand der Ellipse. Der Parameter `MeasureSigma` legt eine Standardabweichung für die Glättung der Grauwerte im Bild durch den Operator `apply_metrology_model`

fest. Auffällige Kanten können mit dem Parameter `MeasureThreshold`, der einen Schwellenwert für die Kantenamplitude, d.h. den Absolutbetrag der ersten Ableitung, angibt, selektiert werden. Der Operator `add_metrology_object_ellipse_measure` gibt den Index des hinzugefügten Metrology-Messobjekts in dem Parameter `Index` zurück.

Des weiteren können in `GenParamName` und `GenParamValue` generische Parameter angepasst werden. Die folgenden generischen Parameter `GenParamName` und die dazugehörigen Werte `GenParamValue` zur Festlegung eines Ellipsenbogens sind verfügbar:

'start_phi': Der Parameter legt den Anfangswinkel des Ellipsenbogens fest. Der Anfangswinkel wird relativ zur positiven großen Halbachse gemessen, deren Orientierung durch `Phi` gegeben ist und bezieht sich auf den kleinsten die Ellipse umgebenden Kreis. Der tatsächliche Anfangspunkt der Ellipse ergibt sich aus dem Schnittpunkt der Ellipse mit der Orthogonalprojektion des entsprechenden Kreispunktes auf die große Halbachse. Der Winkel wird dabei im Ellipsenkoordinatensystem bezogen auf die Hauptachse der Ellipse mathematisch positiv angegeben. Die beiden Hauptpole der Ellipse haben demnach die Winkel 0 bzw. π, die Nebenpole die Winkel $\pi/2$ bzw. $3\pi/2$. Um eine geschlossene Ellipse zu erhalten, müssen der Wert des Parameters *'start_phi'* auf 0 und der Wert des Parameters *'end_phi'* auf 2π gesetzt sein (bei positivem Umlaufsinn). Die Eingabewerte werden automatisch auf das Intervall $[0, 2\pi]$ abgebildet.

Wertevorschläge: 0.0, 0.78, 6.28318

Defaultwert: 0.0

'end_phi': Der Parameter legt den Endwinkel des Ellipsenbogens fest. Der Endwinkel wird relativ zur positiven großen Halbachse gemessen, deren Orientierung durch `Phi` gegeben ist und bezieht sich auf den kleinsten die Ellipse umgebenden Kreis. Der tatsächliche Endpunkt der Ellipse ergibt sich aus dem Schnittpunkt der Ellipse mit der Orthogonalprojektion des entsprechenden Kreispunktes auf die große Halbachse. Der Winkel wird dabei im Ellipsenkoordinatensystem bezogen auf die Hauptachse der Ellipse mathematisch positiv angegeben. Die beiden Hauptpole der Ellipse haben demnach die Winkel 0 bzw. π, die Nebenpole die Winkel $\pi/2$ bzw. $3\pi/2$. Um eine geschlossene Ellipse zu erhalten, müssen der Wert des Parameters *'start_phi'* auf 0 und der Wert des Parameters *'end_phi'* auf 2π gesetzt sein (bei positivem Umlaufsinn). Die Eingabewerte werden automatisch auf das Intervall $[0, 2\pi]$ abgebildet.

Wertevorschläge: 0.0, 0.78, 6.28318

Defaultwert: 6.28318

'point_order': Der Parameter legt die Richtung des Ellipsenbogens fest. Wird der Wert auf *'positive'* gesetzt, ist der Ellipsenbogen zwischen *'start_phi'* und *'end_phi'* im mathematisch positiven Sinn (gegen den Uhrzeigersinn) definiert. Wird der Wert auf *'negative'* gesetzt, ist der Ellipsenbogen zwischen *'start_phi'* und *'end_phi'* im mathematisch negativen Sinn (im Uhrzeigersinn) definiert.

Wertevorschläge: 'positive', 'negative'

Defaultwert: 'positive'

Weitere generische Parameter sind alle generischen Parameter, die im Operator `set_metrology_object_param` zur Verfügung stehen. In den meisten Fällen reichen die Standardwerte aus, so dass keine Anpassung nötig ist.

Parameter

- ▷ **`MetrologyHandle`** (input_control) . . . . . metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **`Row`** (input_control) . . . . . ellipse.center.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Mittelpunkts der Ellipse.
- ▷ **`Column`** (input_control) . . . . . ellipse.center.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Mittelpunkts der Ellipse.
- ▷ **`Phi`** (input_control) . . . . . ellipse.angle.rad(-array) ⇝ *real* / integer
  Orientierung der Hauptachse [rad].
- ▷ **`Radius1`** (input_control) . . . . . ellipse.radius1(-array) ⇝ *real* / integer
  Länge der großen Halbachse.
- ▷ **`Radius2`** (input_control) . . . . . ellipse.radius2(-array) ⇝ *real* / integer
  Länge der kleinen Halbachse.

▷ **MeasureLength1** (input_control) . . . . . number ⇝ *real* / integer
Halbe Länge der Messregionen senkrecht zum Rand.
Defaultwert: 20.0
Wertevorschläge: `MeasureLength1` ∈ {10.0, 20.0, 30.0}
Typischer Wertebereich: 1.0 ≤ MeasureLength1
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: MeasureLength1 < Radius1 && MeasureLength1 < Radius2

▷ **MeasureLength2** (input_control) . . . . . number ⇝ *real* / integer
Halbe Länge der Messregionen tangential zum Rand.
Defaultwert: 5.0
Wertevorschläge: `MeasureLength2` ∈ {3.0, 5.0, 10.0}
Typischer Wertebereich: 1.0 ≤ MeasureLength2
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **MeasureSigma** (input_control) . . . . . number ⇝ *real* / integer
Sigma der Gaußfunktion für die Glättung.
Defaultwert: 1.0
Wertevorschläge: `MeasureSigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ MeasureSigma
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: 0.4 <= MeasureSigma && MeasureSigma <= 100

▷ **MeasureThreshold** (input_control) . . . . . number ⇝ *real* / integer
Minimale Amplitude einer Kante.
Defaultwert: 30.0
Wertevorschläge: `MeasureThreshold` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich: 1 ≤ `MeasureThreshold` ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 2

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'distance_threshold', 'end_phi', 'instances_outside_measure_regions', 'max_num_iterations', 'measure_distance', 'measure_interpolation', 'measure_select', 'measure_transition', 'min_score', 'num_instances', 'num_measures', 'point_order', 'rand_seed', 'start_phi'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {1, 2, 3, 4, 5, 10, 20, 'all', 'true', 'false', 'first', 'last', 'positive', 'negative', 'uniform', 'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **Index** (output_control) . . . . . integer ⇝ *integer*
Index des erstellten Metrology-Messobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_metrology_object_ellipse_measure` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

set_metrology_model_image_size

Nachfolger

align_metrology_model, apply_metrology_model

Alternativen

add_metrology_object_generic

Siehe auch

get_metrology_object_model_contour, set_metrology_model_param, add_metrology_object_circle_measure, add_metrology_object_line_measure, add_metrology_object_rectangle2_measure

Modul

2D Metrology

```
add_metrology_object_generic ( : : MetrologyHandle, Shape,
    ShapeParam, MeasureLength1, MeasureLength2, MeasureSigma,
    MeasureThreshold, GenParamName, GenParamValue : Index )
```

Hinzufügen eines Metrology-Messobjekts zu einem Messmodell.

add_metrology_object_generic fügt ein Metrology-Messobjekt vom Typ Shape zu einem Messmodell hinzu und bereitet die rechteckigen Messregionen vor.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Handle des Messmodells wird in MetrologyHandle übergeben. Shape definiert den Typ des Objekts der zum Messmodell hinzugefügt werden soll. add_metrology_object_generic gibt den Index des hinzugefügten Metrology-Messobjekts im Parameter Index zurück. Beachten Sie, dass add_metrology_object_generic die Funktionalität der Operatoren add_metrology_object_circle_measure, add_metrology_object_ellipse_measure, add_metrology_object_rectangle2_measure und add_metrology_object_line_measure in einem Operator zusammenfasst.

Liste der geometrischen Formen

Mittels Shape wird spezifiziert, welche Art von Metrology-Messobjekt dem Messmodell hinzugefügt wird. Abhängig vom Wert von Shape werden folgende Werte erwartet:

'circle': Die geometrische Form des Metrology-Messobjekts vom Typ Kreis oder Kreisbogen wird durch seinen Mittelpunkt (Row, Column) und den Radius beschrieben.

ShapeParam=*[Row, Column, Radius]*

'rectangle2': Die geometrische Form des Metrology-Messobjekts vom Typ Rechteck wird durch seinen Mittelpunkt (Row, Column), die Orientierung der Hauptachse Phi und die halben Kantenlängen Length1 und Length2 beschrieben. Der Eingabewert für Phi wird automatisch auf das Intervall]-PI,PI] abgebildet.

ShapeParam=*[Row, Column, Phi, Length1, Length2]*

'ellipse': Die geometrische Form des Metrology-Messobjekts vom Typ Ellipse wird durch ihren Mittelpunkt (Row, Column), die Orientierung der Hauptachse Phi und die Länge der großen und kleinen Halbachsen Radius1 und Radius2 beschrieben. Der Eingabewert für Phi wird automatisch auf das Intervall]-PI,PI] abgebildet.

ShapeParam=*[Row, Column, Phi, Radius1, Radius2]*

'line': Die geometrische Form des Metrology-Messobjekts vom Typ Linie wird durch die Koordinaten des Startpunkts (RowBegin, ColumnBegin) und die Koordinaten des Endpunkts (RowEnd, ColumnEnd) beschrieben.

ShapeParam=*[RowBegin, ColumnBegin, RowEnd, ColumnEnd]*

Definition der Messregionen

add_metrology_object_generic bereitet die rechteckigen Messregionen vor. Die Messregionen liegen *senkrecht* zur Objektkontur. Die halben Kantenlängen der Messregionen senkrecht und tangential zur Objektkontur werden jeweils in MeasureLength1 und in MeasureLength2 gesetzt. Die Mittelpunkte der Messregionen liegen auf dem Rand der Objektkontur. Der Parameter MeasureSigma legt eine Standardabweichung für

die Glättung der Grauwerte im Bild durch den Operator apply_metrology_model fest. Auffällige Kanten können mit dem Parameter MeasureThreshold, der einen Schwellenwert für die Kantenamplitude, d.h. den Absolutbetrag der ersten Ableitung, angibt, selektiert werden.

Generische Parameter

In GenParamName und GenParamValue können generische Parameter übergeben werden. Setzbar sind alle generischen Parameter, die auch im Operator set_metrology_object_param zur Verfügung stehen. In den meisten Fällen reichen die Standardwerte aus, so dass keine Anpassung nötig ist. Zusätzlich sind die folgenden generischen Parameter GenParamName und die dazugehörigen Werte GenParamValue zur Festlegung eines Kreis- oder Ellipsenbogens nur für Shape = 'circle' oder 'ellipse' verfügbar:

'start_phi': Der Parameter legt den Anfangswinkel des Kreis- bzw. Ellipsenbogens fest. Für eine Ellipse wird der Anfangswinkel relativ zur positiven großen Halbachse gemessen und bezieht sich auf den kleinsten die Ellipse umgebenden Kreis. Der tatsächliche Anfangspunkt der Ellipse ergibt sich aus dem Schnittpunkt der Ellipse mit der Orthogonalprojektion des entsprechenden Kreispunktes auf die große Halbachse. Um einen geschlossenen Kreis bzw. eine geschlossene Ellipse zu erhalten, müssen der Wert des Parameters *'start_phi'* auf 0 und der Wert des Parameters *'end_phi'* auf 2π gesetzt sein (bei positivem Umlaufsinn). Die Eingabewerte werden automatisch auf das Intervall $[0, 2\pi]$ abgebildet.
Wertevorschläge: 0.0, 0.78, 6.28318
Defaultwert: 0.0

'end_phi': Der Parameter legt den Endwinkel des Kreis- bzw. Ellipsenbogens fest. Für eine Ellipse wird der Endwinkel relativ zur positiven großen Halbachse gemessen und bezieht sich auf den kleinsten die Ellipse umgebenden Kreis. Der tatsächliche Endpunkt der Ellipse ergibt sich aus dem Schnittpunkt der Ellipse mit der Orthogonalprojektion des entsprechenden Kreispunktes auf die große Halbachse. Um einen geschlossenen Kreis bzw. eine geschlossene Ellipse zu erhalten, müssen der Wert des Parameters *'start_phi'* auf 0 und der Wert des Parameters *'end_phi'* auf 2π gesetzt sein (bei positivem Umlaufsinn). Die Eingabewerte werden automatisch auf das Intervall $[0, 2\pi]$ abgebildet.
Wertevorschläge: 0.0, 0.78, 6.28318
Defaultwert: 6.28318

'point_order': Der Parameter legt die Richtung des Kreis- bzw. Ellipsenbogens fest. Wird der Wert auf *'positive'* gesetzt, ist der Bogen zwischen *'start_phi'* und *'end_phi'* im mathematisch positiven Sinn (gegen den Uhrzeigersinn) definiert. Wird der Wert auf *'negative'* gesetzt, ist der Bogen zwischen *'start_phi'* und *'end_phi'* im mathematisch negativen Sinn (im Uhrzeigersinn) definiert.
Wertevorschläge: 'positive', 'negative'
Defaultwert: 'positive'

Parameter

▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **Shape** (input_control) ..... attribute.name(-array) ⇝ *string*
Typ des zu erstellenden Metrology-Messobjekts.
Defaultwert: 'circle'
Werteliste: Shape ∈ {'circle', 'ellipse', 'rectangle2', 'line'}

▷ **ShapeParam** (input_control) ..... attribute.value-array ⇝ *real* / integer
Parameter des zu erstellenden Metrology-Messobjekts.

▷ **MeasureLength1** (input_control) ..... number ⇝ *real* / integer
Halbe Länge der Messregionen senkrecht zum Rand.
Defaultwert: 20.0
Wertevorschläge: MeasureLength1 ∈ {10.0, 20.0, 30.0}
Typischer Wertebereich: $1.0 \leq$ MeasureLength1 ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **MeasureLength2** (input_control) ..... number ⇝ *real* / integer
Halbe Länge der Messregionen tangential zum Rand.
Defaultwert: 5.0
Wertevorschläge: MeasureLength2 ∈ {3.0, 5.0, 10.0}
Typischer Wertebereich: $1.0 \leq$ MeasureLength2 ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **MeasureSigma** (input_control) ..... number ⇝ *real* / integer
Sigma der Gaußfunktion für die Glättung.
Defaultwert: 1.0
Wertevorschläge: `MeasureSigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ `MeasureSigma` ≤ 100 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: 0.4 <= MeasureSigma && MeasureSigma <= 100

▷ **MeasureThreshold** (input_control) ..... number ⇝ *real* / integer
Minimale Amplitude einer Kante.
Defaultwert: 30.0
Wertevorschläge: `MeasureThreshold` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich: 1 ≤ `MeasureThreshold` ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 2

▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'distance_threshold', 'end_phi', 'instances_outside_measure_regions', 'max_num_iterations', 'measure_distance', 'measure_interpolation', 'measure_select', 'measure_transition', 'min_score', 'num_instances', 'num_measures', 'point_order', 'rand_seed', 'start_phi'}

▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *real* / integer / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {1, 2, 3, 4, 5, 10, 20, 'all', 'true', 'false', 'first', 'last', 'positive', 'negative', 'uniform', 'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **Index** (output_control) ..... integer ⇝ *integer*
Index des erstellten Metrology-Messobjekts.

Beispiel

```
create_metrology_model (MetrologyHandle)
read_image (Image, 'fabrik')
get_image_size (Image, Width, Height)
set_metrology_model_image_size (MetrologyHandle, Width, Height)
LinePar := [45,360,415,360]
RectPar1 := [270,232,rad(0),30,25]
RectPar2 := [360,230,rad(0),30,25]
LinePar := [45,360,415,360]
RectPar3 := [245,320,rad(-90),70,35]
* Add two rectangles
add_metrology_object_generic (MetrologyHandle, 'rectangle2', [RectPar1,RectPar2], \
                              20, 5, 1, 30, [], [], Indices)
* Add a rectangle and a line
add_metrology_object_generic (MetrologyHandle, ['rectangle2','line'], \
                             [RectPar3,LinePar], 20, 5, 1, 30, [], [], Index)
get_metrology_object_model_contour (Contour, MetrologyHandle, 'all', 1.5)
apply_metrology_model (Image, MetrologyHandle)
get_metrology_object_result_contour (Contour1, MetrologyHandle, 'all', \
                                     'all', 1.5)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_metrology_object_generic` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`set_metrology_model_image_size`, `set_metrology_model_param`

Nachfolger

`align_metrology_model`, `apply_metrology_model`, `set_metrology_model_param`

Siehe auch

`get_metrology_object_model_contour`

Modul

2D Metrology

`add_metrology_object_line_measure` `( : : MetrologyHandle, RowBegin, ColumnBegin, RowEnd, ColumnEnd, MeasureLength1, MeasureLength2, MeasureSigma, MeasureThreshold, GenParamName, GenParamValue : Index )`

Hinzufügen einer Linie zu einem Messmodell.

`add_metrology_object_line_measure` fügt ein Metrology-Messobjekt vom Typ Line zu einem Messmodell hinzu und bereitet die rechteckigen Messregionen vor. Das Handle des Messmodells muss in `MetrologyHandle` übergeben werden.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Die genäherte geometrische Form des Metrology-Messobjekts vom Typ Linie wird durch die Koordinaten des Startpunkts (`RowBegin`, `ColumnBegin`) und die Koordinaten des Endpunkts (`RowEnd`, `ColumnEnd`) beschrieben. Die Messregionen liegen *senkrecht* zur Linie. Die halbe Kantenlänge der Messregionen senkrecht und tangential zur Linie werden jeweils in `MeasureLength1` und in `MeasureLength2` gesetzt. Die Mittelpunkte der Messregionen liegen auf dem Rand des Objekts. Der Parameter `MeasureSigma` legt eine Standardabweichung für die Glättung der Grauwerte im Bild durch den Operator `apply_metrology_model` fest. Auffällige Kanten können mit dem Parameter `MeasureThreshold`, der einen Schwellenwert für die Kantenamplitude, d.h. den Absolutbetrag der ersten Ableitung, angibt, selektiert werden.

Des weiteren können in `GenParamName` und `GenParamValue` generische Parameter angepasst werden. Als generische Parameter sind alle generischen Parameter möglich, die im Operator `set_metrology_object_param` zur Verfügung stehen. In den meisten Fällen reichen die Standardwerte aus, so dass keine Anpassung nötig ist.

Der Operator `add_metrology_object_line_measure` gibt den Index des hinzugefügten Metrology-Messobjekts im dem Parameter `Index` zurück.

Parameter

- ▷ **`MetrologyHandle`** (input_control) ..... metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **`RowBegin`** (input_control) ..... line.begin.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Startpunktes der Linie.
- ▷ **`ColumnBegin`** (input_control) ..... line.begin.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Startpunktes der Linie.
- ▷ **`RowEnd`** (input_control) ..... line.end.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Endpunktes der Linie.
- ▷ **`ColumnEnd`** (input_control) ..... line.end.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Endpunktes der Linie.

▷ **MeasureLength1** (input_control) . . . . . number ⇝ *real* / integer
Halbe Länge der Messregionen senkrecht zum Rand.
Defaultwert: 20.0
Wertevorschläge: `MeasureLength1` ∈ {10.0, 20.0, 30.0}
Typischer Wertebereich: 1.0 ≤ MeasureLength1
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **MeasureLength2** (input_control) . . . . . number ⇝ *real* / integer
Halbe Länge der Messregionen tangential zum Rand.
Defaultwert: 5.0
Wertevorschläge: `MeasureLength2` ∈ {3.0, 5.0, 10.0}
Typischer Wertebereich: 1.0 ≤ MeasureLength2
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **MeasureSigma** (input_control) . . . . . number ⇝ *real* / integer
Sigma der Gaußfunktion für die Glättung.
Defaultwert: 1.0
Wertevorschläge: `MeasureSigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ MeasureSigma
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: 0.4 <= MeasureSigma && MeasureSigma <= 100

▷ **MeasureThreshold** (input_control) . . . . . number ⇝ *real* / integer
Minimale Amplitude einer Kante.
Defaultwert: 30.0
Wertevorschläge: `MeasureThreshold` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich: 1 ≤ `MeasureThreshold` ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 2

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'distance_threshold', 'instances_outside_measure_regions', 'max_num_iterations', 'measure_distance', 'measure_interpolation', 'measure_select', 'measure_transition', 'min_score', 'num_instances', 'num_measures', 'rand_seed'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {1, 2, 3, 4, 5, 10, 20, 'all', 'true', 'false', 'first', 'last', 'positive', 'negative', 'uniform', 'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **Index** (output_control) . . . . . integer ⇝ *integer*
Index des erstellten Metrology-Messobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_metrology_object_line_measure` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

set_metrology_model_image_size

Nachfolger

align_metrology_model, apply_metrology_model

Alternativen

add_metrology_object_generic

Siehe auch

get_metrology_object_model_contour, set_metrology_model_param, add_metrology_object_circle_measure, add_metrology_object_ellipse_measure, add_metrology_object_rectangle2_measure

Modul

2D Metrology

add_metrology_object_rectangle2_measure (: : MetrologyHandle, Row, Column, Phi, Length1, Length2, MeasureLength1, MeasureLength2, MeasureSigma, MeasureThreshold, GenParamName, GenParamValue : Index)

Hinzufügen eines Rechtecks zu einem Messmodell.

add_metrology_object_rectangle2_measure fügt ein Metrology-Messobjekt vom Typ Rechteck zu einem Messmodell hinzu und bereitet die rechteckigen Messregionen vor. Das Handle des Messmodells muss in MetrologyHandle übergeben werden.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Die genäherte geometrische Form des Metrology-Messobjekts vom Typ Rechteck wird durch seinen Mittelpunkt (Row, Column), die Orientierung der Hauptachse Phi und die halben Kantenlängen Length1 und Length2 beschrieben. Der Eingabewert Phi wird automatisch auf das Intervall $]-\pi,\pi]$ abgebildet. Die Messregionen liegen *senkrecht* zum Rand des Rechtecks. Die halben Kantenlängen der Messregionen senkrecht und tangential zum Rechteck werden jeweils in MeasureLength1 und in MeasureLength2 gesetzt. Die Mittelpunkte der Messregionen liegen auf dem Rand des Rechtecks. Der Parameter MeasureSigma legt eine Standardabweichung für die Glättung der Grauwerte im Bild durch den Operator apply_metrology_model fest. Auffällige Kanten können mit dem Parameter MeasureThreshold, der einen Schwellenwert für die Kantenamplitude, d.h. den Absolutbetrag der ersten Ableitung, angibt, selektiert werden.

Des weiteren können in GenParamName und GenParamValue generische Parameter angepasst werden. Als generische Parameter sind alle generischen Parameter möglich, die im Operator set_metrology_object_param zur Verfügung stehen. In den meisten Fällen reichen die Standardwerte aus, so dass keine Anpassung nötig ist.

Der Operator add_metrology_object_rectangle2_measure gibt den Index des hinzugefügten Metrology-Messobjekts in dem Parameter Index zurück.

Parameter

- ▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **Row** (input_control) ..... rectangle2.center.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Mittelpunkts des Rechtecks.
- ▷ **Column** (input_control) ..... rectangle2.center.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Mittelpunkts des Rechtecks.
- ▷ **Phi** (input_control) ..... rectangle2.angle.rad(-array) ⇝ *real* / integer
  Orientierung der Hauptachse [rad].
- ▷ **Length1** (input_control) ..... rectangle2.hwidth(-array) ⇝ *real* / integer
  Erster Halbmesser (halbe Länge) des Rechtecks.
- ▷ **Length2** (input_control) ..... rectangle2.hheight(-array) ⇝ *real* / integer
  Zweiter Halbmesser (halbe Länge) des Rechtecks.

▷ **MeasureLength1** (input_control) . . . . . number ⇝ *real* / integer
Halbe Länge der Messregionen senkrecht zum Rand.
Defaultwert: 20.0
Wertevorschläge: `MeasureLength1` ∈ {10.0, 20.0, 30.0}
Typischer Wertebereich: 1.0 ≤ MeasureLength1
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: MeasureLength1 < Length1 && MeasureLength1 < Length2

▷ **MeasureLength2** (input_control) . . . . . number ⇝ *real* / integer
Halbe Länge der Messregionen tangential zum Rand.
Defaultwert: 5.0
Wertevorschläge: `MeasureLength2` ∈ {3.0, 5.0, 10.0}
Typischer Wertebereich: 1.0 ≤ MeasureLength2
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **MeasureSigma** (input_control) . . . . . number ⇝ *real* / integer
Sigma der Gaußfunktion für die Glättung.
Defaultwert: 1.0
Wertevorschläge: `MeasureSigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.4 ≤ MeasureSigma
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: 0.4 <= MeasureSigma && MeasureSigma <= 100

▷ **MeasureThreshold** (input_control) . . . . . number ⇝ *real* / integer
Minimale Amplitude einer Kante.
Defaultwert: 30.0
Wertevorschläge: `MeasureThreshold` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich: 1 ≤ `MeasureThreshold` ≤ 255 (lin)
Minimale Schrittweite: 0.5
Empfohlene Schrittweite: 2

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'distance_threshold', 'instances_outside_measure_regions', 'max_num_iterations', 'measure_distance', 'measure_interpolation', 'measure_select', 'measure_transition', 'min_score', 'num_instances', 'num_measures', 'rand_seed'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {1, 2, 3, 4, 5, 10, 20, 'all', 'true', 'false', 'first', 'last', 'positive', 'negative', 'uniform', 'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **Index** (output_control) . . . . . integer ⇝ *integer*
Index des erstellten Metrology-Messobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_metrology_object_rectangle2_measure` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

set_metrology_model_image_size

Nachfolger

align_metrology_model, apply_metrology_model

Alternativen

add_metrology_object_generic

Siehe auch

get_metrology_object_model_contour, set_metrology_model_param, add_metrology_object_circle_measure, add_metrology_object_ellipse_measure, add_metrology_object_line_measure

Modul

2D Metrology

align_metrology_model (: : MetrologyHandle, Row, Column, Angle :)

Ausrichtung eines Messmodells.

align_metrology_model bewegt und rotiert das Messmodell, gegeben durch MetrologyHandle, als Ganzes relativ zum Bildkoordinatensystem, das seinen Ursprung in der linken oberen Ecke hat.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

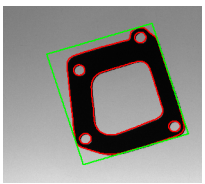
Ein Ausrichten des Messmodells gewährleistet, dass das Messmodell an die Position der zu messenden Objekte im aktuellen Bild angepasst wird. Diese Modellausrichtung wird dann von apply_metrology_model benutzt, um eine Messung durchzuführen. Als erstes wird das Messmodell um Angle rotiert. Dann wird das Messmodell um Row und Column verschoben. Die Werte der Modellausrichtung werden durch den nächsten Aufruf von align_metrology_model überschrieben.

Berechnung der Parameter der Modellausrichtung

Die Parameter der Modellausrichtung können auf verschiedene Weise ermittelt werden. Im Folgenden werden drei Möglichkeiten aufgezeigt:

'Mittels Regionenanalyse':

Falls das Messmodell über Regionenverarbeitung extrahiert werden kann und die Lage und Orientierung der Region in nachfolgenden Bildern des Objekts nur wenig variiert, kann man die Parameter des Referenzsystems und der Modellausrichtung über Regionenanalyse berechnen. Für das nachfolgende Bild wurde dazu threshold und smallest_rectangle2 benutzt.



Die mit threshold extrahierte Region ist in rot dargestellt. Das mit smallest_rectangle2 berechnete Rechteck ist in grün gezeigt.

1. Setzen des Referenzsystems
   In dem Bild, in dem man das Messmodell definiert hat, extrahiert man eine Region, die die Metrology-Messobjekte beinhaltet. Man ermittelt die Lage dieser Region bzgl. des Bildkoordinatensystems und setzt sie mit set_metrology_model_param als Referenzsystem für das Messmodell. Dadurch kann man beim nachfolgenden Ausrichten des Messmodells die Lage der extrahierten Region direkt verwenden.
   Beispiel:

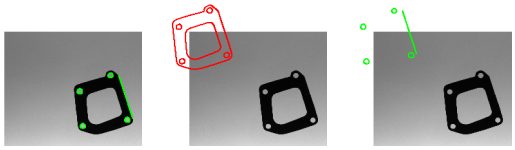
```
threshold(Image, Region, 0, 50)
smallest_rectangle2(Region, RowOrig, ColumnOrig, AngleOrig, Length1,
Length2)
set_metrology_model_param(MetrologyHandle, 'reference_system',
[RowOrig, ColumnOrig, AngleOrig])
```

2. Bestimmen der Modellausrichtung
   In einem Bild, in dem das Messmodell in einer anderen Lage vorkommt, ermittelt man die aktuelle Lage der extrahierten Region. Diese Lage benutzt man dann für die Ausrichtung des Messmodells.
   Beispiel:

```
threshold(CurrentImage, Region, 0, 50)
smallest_rectangle2(Region, RowAlign, ColumnAlign, AngleAlign,
Length1, Length2)
align_metrology_model(MetrologyHandle, RowAlign, ColumnAlign,
AngleAlign)
```

'Mittels eines Formmodells':

Wenn ein Formmodel genutzt wird, um das Messmodell im aktuellen Bild auszurichten, muss das Referenzsystem, bzgl. dem die Metrology-Messobjekte gegeben sind, mit `set_metrology_model_param` so gesetzt werden, dass es mit dem Koordinatensystem des Formmodells übereinstimmt. Nur dann können die Ergebnisse (`Row`, `Column`, `Angle`) von `find_shape_model` direkt in `align_metrology_model` für die Ausrichtung des Messmodells im aktuellen Bild verwendet werden. Die einzelnen Schritte hierfür, werden im Folgenden aufgezeigt.



(1) (2) (3)

(1) Die Konturen des Messmodells (2) Die Konturen des Formmodells (3) Die Konturen des Messmodells nach dem Setzen des Referenzpunktes

1. Setzen des Referenzsystems
   In dem Bild, in dem das Messmodell definiert wurde, wird die Lage der Ursprungs des Formmodells ermittelt.
   Beispiel:

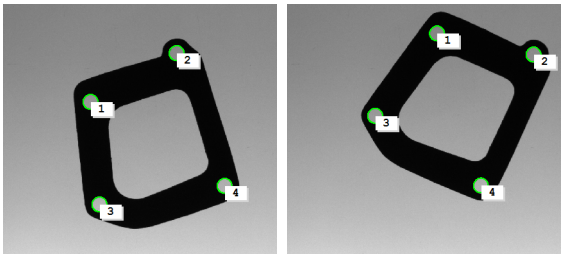
```
create_shape_model(Image, 6, 0, rad(360)
area_center(Image, Area, RowOrig, ColumnOrig)
set_metrology_model_param(MetrologyHandle, 'reference_system',
[RowOrig,ColumnOrig,0])
```

2. Bestimmen der Modellausrichtung
   In einem Bild, in dem das zu messende Objekt in einer anderen Orientierung vorkommt, wird die aktuelle Lage des Formmodells ermittelt und an `align_metrology_model` übergeben.
   Beispiel:

```
find_shape_model(CurrentImage, ShapeModelID, 0, rad(360)
align_metrology_model(MetrologyHandle, RowAlign, ColumnAlign,
AngleAlign)
```

Mittels einer starren 2D-Transformation:

Sind bestimmte Modellpunkte (gegeben als [PRowModel], [PColumnModel]) eindeutig identifizierbar, und können diese in weiteren Bildern, in denen die zu messenden Objekte verschoben und rotiert auftreten, wieder eindeutig identifiziert werden, so kann zwischen diesen Punkten eine starre Abbildung berechnet werden. Die Parameter der Abbildung können dann direkt für die Modellausrichtung benutzt werden. In diesem Fall muss der Referenzpunkt des Messmodells *nicht* verändert werden.



(1) (2)

(1) Die Konturen der Metrology-Messobjekte und vier korrespondierende Punkte im Bild das für die Erstellung des Messmodells benutzt wurde. (2) Die Konturen der Metrology-Messobjekte und die vier korrespondierenden Punkte in einem neuen Bild.

1. Ermitteln der Punktkkorrespondenzen
2. Bestimmen der Modellausrichtung
   Die folgende Operatorsequenz berechnet die Parameter der Modellausrichtung (`Row`, `Column`, `Angle`) aus korrespondierenden Punkten im Modellbild und in einem weiteren Bild.
   Beispiel:

```
vector_to_rigid(PRowModel, PColumnModel, PRowCurrent, PColumnCurrent,
HomMat2D)
hom_mat2d_to_affine_par(HomMat2D, Sx, Sy, Angle, Theta, Row, Column)
align_metrology_model(MetrologyHandle, Row, Column, Angle)
```

Parameter

▷ **`MetrologyHandle`** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells..

▷ **`Row`** (input_control) ..... real ⇝ *real* / integer
Zeilenkoordinate der Ausrichtung.
Defaultwert: 0

▷ **`Column`** (input_control) ..... real ⇝ *real* / integer
Spaltenkoordinate der Ausrichtung.
Defaultwert: 0

▷ **`Angle`** (input_control) ..... angle.rad ⇝ *real* / integer
Rotationswinkel der Ausrichtung.
Defaultwert: 0

Beispiel

```
read_image (Image, 'metal-parts/circle_plate_01')
create_metrology_model (MetrologyHandle)
get_image_size (Image, Width, Height)
set_metrology_model_image_size (MetrologyHandle, Width, Height)
CircleParam := [354,274,53]
CircleParam := [CircleParam,350,519,53]
add_metrology_object_generic (MetrologyHandle, 'circle', CircleParam, 20,\
                              5, 1, 30, [], [], CircleIndices)
create_shape_model (Image, 6, 0, rad(360), 'auto', 'auto', \
                    'use_polarity', 'auto', 20, ShapeModelID)
* Determine location of shape model origin
area_center (Image, Area, RowOrigin, ColOrigin)
set_metrology_model_param (MetrologyHandle, 'reference_system', \
                           [RowOrigin,ColOrigin,0])
read_image (CurrentImage, 'metal-parts/circle_plate_02')
find_shape_model (CurrentImage, ShapeModelID, 0, rad(360), 0.5, 1, 0, \
                  'least_squares', 5, 0.9, RowAlign, ColAlign, AngleAlign,\
                  Score)
align_metrology_model (MetrologyHandle, RowAlign,ColAlign,AngleAlign)
apply_metrology_model (CurrentImage, MetrologyHandle)
get_metrology_object_result (MetrologyHandle, CircleIndices, 'all', \
                             'result_type', 'all_param', Rectangle)
get_metrology_object_result_contour (Contour, MetrologyHandle, CircleIndices,\
                                     'all', 1.5)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `align_metrology_model` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`set_metrology_model_param`, `add_metrology_object_generic`

Nachfolger

`apply_metrology_model`

Siehe auch

`get_metrology_object_model_contour`

Modul

2D Metrology

`apply_metrology_model` (Image : : MetrologyHandle :)

Messen und Anpassen der geometrischen Formen aller Metrology-Messobjekte eines Messmodells.

`apply_metrology_model` lokalisiert die Kanten der Metrology-Messobjekte im Bild `Image` und passt die dazugehörigen geometrischen Formen an die gefundenen Kantenpositionen an.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist im Handle `MetrologyHandle` definiert und beinhaltet genäherte Werte für die geometrischen Formen. Die Messungen werden folgendermaßen durchgeführt:

Ermittlung der Kantenpositionen

Innerhalb der Messregionen der Metrology-Messobjekte werden die Positionen der Kanten ermittelt. Die Lage der Kanten wird intern mit dem Operator `measure_pos` oder `fuzzy_measure_pos` berechnet. Letzterer wird genutzt, wenn mindestens eine Fuzzy-Funktion für das Metrology-Messobjekt gesetzt ist.

Anpassen der geometrischen Formen an die Kantenpositionen

Die geometrischen Formen der Metrology-Messobjekte werden so angepasst, dass sie optimal zu den resultierenden Kantenpositionen passen. Dabei wird ein RANSAC-Verfahren dazu verwendet die Mindestanzahl von Startkantenpositionen auszuwählen, die notwendig ist, um eine Instanz einer bestimmten geometrischen Form zu erstellen. Zum Beispiel werden bei einem Metrology-Messobjekt vom Typ Kreis drei Kantenpositionen ausgewählt. Anschließend werden Kantenpositionen festgelegt, die sich in der Nähe der geometrischen Form der zugehörigen Instanz befinden und, falls die Anzahl der geeigneten Kantenpositionen ausreichend ist (siehe den generischen Parameter *'min_score'* des Operators `set_metrology_object_param`), werden diese für die endgültige Anpassung der geometrischen Form verwendet. Falls die Anzahl der geeigneten Kantenpositionen nicht ausreichend ist, werden andere Startkantenpositionen getestet solange bis eine geeignete Auswahl von Kantenpositionen gefunden wird. An die Kantenpositionen, die für das endgültige Anpassen ausgewählt wurden, wird die geometrische Form angepasst und die Parameter werden im Messmodel abgespeichert. Es ist zu beachten, dass mehr als eine Instanz für jedes Metrology-Messobjekt zurückgegeben wird, wenn der generische Parameter *'num_instances'* auf einen Wert größer als 1 gesetzt wird. Dieser und weitere Parameter können beim Hinzufügen des Metrology-Messobjekts zum Messmodell oder separat mit dem Operator `set_metrology_object_param` gesetzt werden.

Zu beachten ist, dass für jede Instanz des Metrology-Messobjekts verschiedene Kantenpositionen genutzt werden, d.h., dass für die Berechnung der zweiten Instanz keine Kantenpositionen verwendet werden, die bereits für das Fitting der ersten Instanz verwendet wurden. Der Algorithmus bricht ab, wenn *'num_instances'* Instanzen gefunden wurden oder wenn die verbleibende Anzahl an geeigneten Startkantenpositionen zu gering für eine weitere Anpassung einer geometrischen Form sind.

Zugriff auf die Ergebnisse

Auf die Ergebnisse der Messungen aus dem Messmodell kann mithilfe von `get_metrology_object_result` zugegriffen werden. Zu beachten ist, dass, wenn mehr als eine Instanz eines Objekts zurückgegeben wird, die Reihenfolge der zurückgegeben Instanzen zufällig ist, d.h. die

Reihenfolge stellt kein Maß für die Qualität der Anpassung dar. Zu beachten ist weiterhin, dass, wenn die Parameter *'camera_param'* und *'plane_pose'* mittels set_metrology_model_param für das Messmodell gesetzt sind, Weltkoordinaten für das Anpassen genutzt werden. Andernfalls werden Bildkoordinaten verwendet. Die XLD Konturen können mit get_metrology_object_result_contour abgefragt werden.

Achtung

Es ist zu beachten, dass alle Messregionen aller Metrology-Messobjekte des Messmodells neu berechnet werden müssen, falls die Bildbreite oder die Bildhöhe des Eingabebildes Image nicht gleich der Breite oder Höhe ist, die im Metrology-Messobjekt gespeichert ist (kann z.B. im Operator set_metrology_model_image_size gesetzt werden). Dies führt zu längeren Laufzeiten des Operators.

Es ist zu beachten, dass apply_metrology_model den Definitionsbereich von Image aus Geschwindigkeitsgründen ignoriert (siehe auch measure_pos).

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert apply_metrology_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

add_metrology_object_generic, add_metrology_object_circle_measure, add_metrology_object_ellipse_measure, add_metrology_object_line_measure, add_metrology_object_rectangle2_measure, align_metrology_model, set_metrology_model_param, set_metrology_object_param

Nachfolger

get_metrology_object_result, get_metrology_object_result_contour, get_metrology_object_measures

Siehe auch

set_metrology_object_fuzzy_param, read_metrology_model, write_metrology_model

Modul

2D Metrology

clear_metrology_model (: : MetrologyHandle :)

Löschen eines Messmodells und Freigabe des verwendeten Speichers.

clear_metrology_model löscht ein mit create_metrology_model, copy_metrology_model, read_metrology_model oder deserialize_metrology_model angelegtes Messmodell. Ebenfalls werden alle enthaltenen Metrology-Messobjekte gelöscht. Es wird der gesamte vom Messmodell und den Metrology-Messobjekten referenzierte Speicherplatz wieder freigegeben. Das Handle des zu löschenden Messmodells muss in MetrologyHandle übergeben werden. Nach dem Aufruf ist das Messmodell ungültig.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Parameter

▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

Ergebnis

`clear_metrology_model` liefert den Wert TRUE, falls der übergebene Handle gültig ist und das darüber referenzierte Messmodell vollständig freigegeben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`get_metrology_object_result`, `write_metrology_model`

Modul

2D Metrology

clear_metrology_object (: : MetrologyHandle, Index :)

Löschen von Metrology-Messobjekten und Freigabe des verwendeten Speichers.

`clear_metrology_object` löscht in einem Messmodell z.B. mit `add_metrology_object_ellipse_measure`, `add_metrology_object_ellipse_measure`, `add_metrology_object_line_measure` oder `add_metrology_object_rectangle2_measure` angelegte Metrology-Messobjekte und gibt den von den Metrology-Messobjekten referenzierten Speicherplatz wieder frei.

Das Handle des Messmodells muss in `MetrologyHandle` übergeben werden. Der Index der zu löschenden Metrology-Messobjekte muss in `Index` übergeben werden. Wird für den Parameter `Index` als Wert *'all'* übergeben, werden alle Metrology-Messobjekte gelöscht. Nach dem Aufruf sind die Metrology-Messobjekte ungültig.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Parameter

▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **Index** (input_control) . . . . . integer(-array) ⇝ *string* / integer
Index der Metrology-Messobjekte.
Defaultwert: 'all'
Wertevorschläge: `Index` ∈ {'all', 0, 1, 2}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `clear_metrology_object` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Modul

2D Metrology

copy_metrology_model (: : MetrologyHandle, Index : CopiedMetrologyHandle)

Kopieren eines Messmodells.

copy_metrology_model erstellt ein neues Messmodell und kopiert die ausgewählten Metrology-Messobjekte des Eingabemessmodells in das neue Ausgabemessmodell.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Eingabemessmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt die Metrology-Messobjekte fest, die kopiert werden sollen. Wird für den Parameter Index als Wert *'all'* übergeben, werden alle Metrology-Messobjekte kopiert. Der Operator gibt das Handle CopiedMetrologyHandle des neuen Messmodells zurück. Er kann genutzt werden, um Speicherplatz zu sparen. Zugriff auf die Parameter der Metrology-Messobjekte ist z.B. mit dem Operator get_metrology_object_param möglich.

Parameter

- ▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **Index** (input_control) ..... integer(-array) ⇝ *string* / integer
  Index der Metrology-Messobjekte.
  Defaultwert: 'all'
  Wertevorschläge: Index ∈ {'all', 0, 1, 2}
- ▷ **CopiedMetrologyHandle** (output_control) ..... integer ⇝ *integer*
  Handle des kopierten Messmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert copy_metrology_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

2D Metrology

create_metrology_model (: : : MetrologyHandle)

Anlegen der Datenstruktur, die für das Messen von geometrischen Formen notwendig ist.

create_metrology_model legt ein Messmodell an, d.h., die Datenstruktur, die notwendig ist, um Objekte einer bestimmten geometrischen Form (Metrology-Messobjekte) mittels 2D-Messtechnik zu messen, und gibt dessen Handle MetrologyHandle zurück.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Achtung

Es ist zu beachten, dass für eine optimale Laufzeit nach dem Aufruf des Operators

create_metrology_model der Operator set_metrology_model_image_size aufgerufen werden sollte.

Parameter

▷ **MetrologyHandle** (output_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

Beispiel

```
read_image (Image, 'fabrik')
create_metrology_model (MetrologyHandle)
get_image_size (Image, Width, Height)
set_metrology_model_image_size (MetrologyHandle, Width, Height)
add_metrology_object_rectangle2_measure (MetrologyHandle, 270, 230, 0, 30, \
                                         25, 10, 2, 1, 30, [], [], Index)
apply_metrology_model (Image, MetrologyHandle)
get_metrology_object_result (MetrologyHandle, Index, 'all', 'result_type', \
                             'all_param', Rectangle)
get_metrology_object_result_contour (Contour, MetrologyHandle, \
                                     Index, 'all', 1.5)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_metrology_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

set_metrology_model_image_size

Modul

2D Metrology

deserialize_metrology_model (: : SerializedItemHandle : MetrologyHandle)

Deserialisieren eines serialisierten Messmodells.

deserialize_metrology_model deserialisiert ein Messmodell, welches mit serialize_metrology_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Messmodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Messmodell mit dem Handle MetrologyHandle gespeichert. Der Zugriff auf die Parameter des Messmodells ist z.B. mit den Operatoren get_metrology_object_param oder get_metrology_object_fuzzy_param möglich.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **MetrologyHandle** (output_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_metrology_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_metrology_model`

Nachfolger

`get_metrology_object_param`, `get_metrology_object_fuzzy_param`, `apply_metrology_model`

Modul

2D Metrology

get_metrology_model_param (: : MetrologyHandle, GenParamName : GenParamValue)

Abfragen von Parametern, die für das gesamte Messmodell gelten.

`get_metrology_model_param` fragt Parameter ab, die für das ganze Messmodell gelten.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell wird durch das Handle `MetrologyHandle` definiert.

Die folgenden Werte können für `GenParamName` gesetzt werden:

'camera_param': Die internen Kameraparameter, die für das Messmodell gesetzt sind.

'plane_pose': Die 3D-Lage der Messebene, die für das Messmodell gesetzt ist. Die 3D-Lage ist in Kamerakoordinaten gegeben.

'reference_system': Die Translation und Rotation des Referenzkoordinatensystems bezüglich des Bildkoordinatensystems. Das in `GenParamValue` zurückgegebene Tupel hat die Form [row, column, angle].

'scale': Der Skalierungsfaktor bzw. die Einheit der Resultate, die mit `get_metrology_object_result` abgerufen werden.

Parameter

▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'camera_param'
Werteliste: `GenParamName` ∈ {'camera_param', 'plane_pose', 'scale', 'reference_system'}

▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *string* / real / integer
Wert des generischen Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_metrology_object_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_metrology_object_indices`, `set_metrology_model_param`

Nachfolger

`get_metrology_object_param`

Siehe auch

`get_metrology_object_param`, `get_metrology_object_num_instances`

Modul

2D Metrology

`get_metrology_object_fuzzy_param` (: : `MetrologyHandle`, `Index`, `GenParamName` : `GenParamValue`)

Abfragen von Fuzzy-Parametern des Messmodells.

Mit Hilfe des Operators `get_metrology_object_fuzzy_param` lassen sich die Fuzzy-Parameter der Metrology-Messobjekte abfragen.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle `MetrologyHandle` definiert. Der Parameter `Index` legt fest für welche Metrology-Messobjekte die Information abgefragt werden soll. Wird für den Parameter `Index` als Wert *'all'* übergeben, werden die Parameter aller Metrology-Messobjekte abgefragt. Die Namen der Parameter, die abgefragt werden können, werden in `GenParamName` übergeben, die korrespondierenden Werte werden in `GenParamValue` in der gleichen Reihenfolge zurückgeliefert. Sämtliche Fuzzy-Parameter können mit `set_metrology_object_fuzzy_param` gesetzt und jederzeit verändert werden.

Im Einzelnen kann auf die folgenden Parameter zugegriffen werden:

'fuzzy_thresh': Die Bedeutung und Verwendung des Parameters entspricht dem Parameter `FuzzyThresh` des Operators `fuzzy_measure_pos` und ist dort beschrieben.

'function_contrast': Mit diesem Parameter kann die gesetzte Fuzzy-Funktion vom Typ contrast abgerufen werden, die im Operator `set_metrology_object_param` gesetzt werden kann. Die Bedeutung und Verwendung des Parameters entspricht dem Parameter `SetType` mit dem Wert *'contrast'* des Operators `set_fuzzy_measure` und ist dort beschrieben. Der Rückgabewert `GenParamValue` enthält die gesetzte Funktion.

'function_position': Mit diesem Parameter kann die gesetzte Fuzzy-Funktion vom Typ position abgerufen werden. Da nur eine Fuzzy-Funktion eines Typs gegeneinander exklusiv gesetzt werden kann, kann nur die zuletzt gesetzte Funktion eines Typs zurückgegeben werden. Der Typ kann *'function_position'*, *'function_position_center'*, *'function_position_end'*, *'function_position_first_edge'* oder *'function_position_last_edge'* sein.

Parameter

▷ **`MetrologyHandle`** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **`Index`** (input_control) . . . . . integer(-array) ⇝ *string* / integer
Index der Metrology-Messobjekte.
Defaultwert: 'all'
Wertevorschläge: `Index` ∈ {'all', 0, 1, 2}

▷ **`GenParamName`** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'fuzzy_thresh'
Werteliste: `GenParamName` ∈ {'function_contrast', 'function_position', 'fuzzy_thresh'}

▷ **`GenParamValue`** (output_control) . . . . . attribute.value-array ⇝ *real* / integer
Werte der generischen Parameter.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_metrology_object_fuzzy_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_metrology_object_indices`, `set_metrology_object_fuzzy_param`

Nachfolger

`set_metrology_object_fuzzy_param`

Siehe auch

`get_metrology_object_param`

Modul

2D Metrology

`get_metrology_object_indices ( : : MetrologyHandle : Indices )`

Abfragen der Indizes der Metrology-Messobjekte eines Messmodells.

Mit Hilfe des Operators `get_metrology_object_indices` lassen sich die Indizes der Metrology-Messobjekte abfragen.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle `MetrologyHandle` definiert. Der Operator `get_metrology_object_indices` gibt die Indizes der Metrology-Messobjekte im Parameter `Indices` zurück. Zugriff auf die Parameter der Metrology-Messobjekte ist z.B. mit dem Operator `get_metrology_object_param` möglich.

Parameter

▷ **`MetrologyHandle`** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **`Indices`** (output_control) ..... integer(-array) ⇝ *integer*
Indizes der Metrology-Messobjekte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_metrology_object_indices` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_metrology_model`

Nachfolger

`get_metrology_object_param`, `get_metrology_object_fuzzy_param`

Siehe auch

`get_metrology_object_num_instances`

Modul

2D Metrology

get_metrology_object_measures (: Contours : MetrologyHandle, Index, Transition : Row, Column)

Abfragen der Messregionen und der Ergebnisse der Kantenextraktion für die Metrology-Messobjekte eines Messmodells.

Mit Hilfe des Operators get_metrology_object_measures lassen sich die Messregionen welche mit add_metrology_object_generic, add_metrology_object_circle_measure, etc. angelegt werden, als XLD-Konturen und die Ergebnisse der Kantenextraktion, die von apply_metrology_model berechnet werden, in Bildkoordinaten abfragen.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt fest, für welche Metrology-Messobjekte die Information abgefragt werden soll. Wird dem Parameter Index als Wert *'all'* übergeben, werden die Messregionen und die Ergebnisse der Kantenextraktion für alle Metrology-Messobjekte abgefragt.

Sind in den Messregionen positive und negative Kantenpositionen vorhanden (siehe generischer Parameterwert *'measure_transition'* des Operators set_metrology_object_param), können diese mit dem Parameter Transition selektiert werden. Wenn Transition auf *'positive'* gesetzt ist, werden nur positive Kanten abgefragt. Wenn Transition auf *'negative'* gesetzt ist, werden nur negative Kanten abgefragt. Alle Kanten werden zurückgegeben, wenn Transition auf *'all'* gesetzt ist.

Der Operator get_metrology_object_measures gibt für jede Messregion eine rechteckige XLD-Kontur mit der Umrandung der Messregion im Parameter Contours zurück. Nach einem Aufruf von apply_metrology_model werden zusätzlich die Bildkoordinaten des Ergebnisses der Kantenextraktion in den Parametern Row und Column als einzelne Punkte zurückgegeben. Zu beachten ist hierbei, dass die Reihenfolge der Punktkoordinaten nicht festgelegt ist. Des weiteren gibt es keine Möglichkeit die Ergebnisse der Kantenextraktion den Messregionen zuzuordnen. Wird get_metrology_object_measures vor apply_metrology_model aufgerufen, bleiben die Parameter Row und Column leer.

Parameter

- ▷ **Contours** (output_object) ..... xld_cont-array ⇝ *object*
  Rechteckige XLD-Konturen der Messregionen.
- ▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **Index** (input_control) ..... integer(-array) ⇝ *string* / integer
  Index der Metrology-Messobjekte.
  Defaultwert: 'all'
  Wertevorschläge: Index ∈ {'all', 0, 1, 2}
- ▷ **Transition** (input_control) ..... string ⇝ *string*
  Kanten mit Hell-Dunkel-('positive') oder Dunkel-Hell-('negative')-Übergängen.
  Defaultwert: 'all'
  Werteliste: Transition ∈ {'all', 'negative', 'positive'}
- ▷ **Row** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der gemessenen Kanten.
- ▷ **Column** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der gemessenen Kanten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_metrology_object_measures den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

apply_metrology_model

Siehe auch

add_metrology_object_generic, add_metrology_object_ellipse_measure, add_metrology_object_line_measure, add_metrology_object_rectangle2_measure, add_metrology_object_circle_measure

Modul

2D Metrology

get_metrology_object_model_contour (: Contour : MetrologyHandle, Index, Resolution :)

Abfragen der Modellkonturen in Bildkoordinaten.

get_metrology_object_model_contour gibt die Konturen für die gewählten Metrology-Messobjekte in Bildkoordinaten zurück.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt fest für welche Metrology-Messobjekte die Ergebnisse abgefragt werden sollen. Wird für den Parameter Index als Wert *'all'* übergeben, werden die Ergebnisse aller Metrology-Messobjekte abgefragt.

Die Form und Lage der Konturen wird durch die Parameter bestimmt, die beim Hinzufügen des Metrology-Messobjekts zum Messmodell mittels z.B. add_metrology_object_generic, add_metrology_object_circle_measure, etc. angegeben wurden. Wurde mit set_metrology_model_param ein Referenzsystem gesetzt oder wurde das Messmodell mit align_metrology_model ausgerichtet, so gehen diese Werte ebenfalls in die aktuelle Lage des Messmodells und damit in die Lage der zurückgegebenen Konturen ein.

Die Auflösung der zurückgegebenen XLD Kontur Contour wird über den Parameter Resolution festgelegt. Er bestimmt den euklidischen Abstand zwischen benachbarten Konturpunkten. Unterschreitet der Eingabewert den intern gesetzten Mindestwert (1.192e-7), wird die Auflösung auf den kleinsten zulässigen Wert gesetzt.

Parameter

- ▷ **Contour** (output_object) ..... xld_cont(-array) ⇝ *object*
  Modellkontur.
- ▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **Index** (input_control) ..... integer(-array) ⇝ *integer* / string
  Index des Metrology-Messobjekts.
  Defaultwert: 0
  Wertevorschläge: Index ∈ {'all', 0, 1, 2}
- ▷ **Resolution** (input_control) ..... real ⇝ *real*
  Abstand zwischen benachbarten Konturpunkten.
  Defaultwert: 1.5
  Restriktion: Resolution >= 1.192e-7

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_metrology_object_model_contour den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_metrology_object_generic, add_metrology_object_circle_measure, add_metrology_object_ellipse_measure, add_metrology_object_rectangle2_measure, add_metrology_object_line_measure

Nachfolger

apply_metrology_model

Siehe auch

set_metrology_model_param, get_metrology_object_measures, align_metrology_model

Modul

2D Metrology

get_metrology_object_num_instances (: : MetrologyHandle, Index : NumInstances)

Abfragen der Anzahl der Instanzen der Metrology-Messobjekte eines Messmodells.

get_metrology_object_num_instances erlaubt das Abfragen der Anzahl der Instanzen (Ergebnisse) der Messungen der Metrology-Messobjekte, die von apply_metrology_model berechnet wurden. Standardmäßig ist die maximale Anzahl an Instanzen eines jeden Messobjekts auf *1* gesetzt, sodass typischerweise auch das Ergebnis von get_metrology_object_num_instances *1* ergibt. Um mehr Instanzen zu erlauben, muss vor der Messung mit apply_metrology_model der Parameter *'num_instances'* mit set_metrology_object_param auf einen höheren Wert oder auf *'all'* gesetzt werden.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt fest für welche Metrology-Messobjekte die Anzahl der Instanzen abgefragt werden sollen. Wird für den Parameter Index als Wert *'all'* übergeben, wird die Anzahl der Instanzen aller Metrology-Messobjekte abgefragt. Der Parameter NumInstances gibt für jedes in Index übergebene Metrology-Messobjekt die Anzahl der Instanzen zurück.

Parameter

- ▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **Index** (input_control) . . . . . integer(-array) ⇝ *integer* / string
  Index der Metrology-Messobjekte.
  Defaultwert: 0
  Wertevorschläge: Index ∈ {'all', 0, 1, 2}
- ▷ **NumInstances** (output_control) . . . . . integer(-array) ⇝ *real* / integer
  Anzahl der Instanzen der Metrology-Messobjekte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_metrology_object_num_instances den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

apply_metrology_model

Nachfolger

clear_metrology_model

Siehe auch

get_metrology_object_indices

Modul

2D Metrology

get_metrology_object_param (: : MetrologyHandle, Index, GenParamName : GenParamValue)

Abfragen eines oder mehrerer Parameter des Messmodells.

Mit Hilfe des Operators get_metrology_object_param lassen sich die Parameter, durch die ein Metrology-Messobjekt beschrieben wird, abfragen.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt fest für welche Metrology-Messobjekte die Information abgefragt werden soll. Wird für den Parameter Index als Wert *'all'* übergeben, werden die Parameter aller Metrology-Messobjekte abgefragt. Die Namen der Parameter, die abgefragt werden sollen, werden in GenParamName übergeben, die korrespondierenden Werte werden in GenParamValue in der gleichen Reihenfolge zurückgeliefert.

Sämtliche allgemeine Parameter können mit set_metrology_object_param gesetzt und jederzeit verändert werden. Parameter, die die Geometrie eines Metrology-Messobjekts beschreiben, können nur beim Erzeugen des Objekts mit den Operatoren add_metrology_object_circle_measure, add_metrology_object_ellipse_measure, add_metrology_object_line_measure oder add_metrology_object_rectangle2_measure gesetzt werden.

Im Einzelnen kann auf die folgenden Parameter zugegriffen werden:

- Gültig für alle Objektarten:

  'min_score', *'num_instances'*, *'instances_outside_measure_regions'*: Die Bedeutung und Verwendung der Parameter ist für den Operator set_metrology_object_param beschrieben.

  'rand_seed', *'distance_threshold'*, *'max_num_iterations'*: Die Bedeutung und Verwendung der Parameter ist für den Operator set_metrology_object_param beschrieben.

  'measure_length1', *'measure_length2'*: Die Bedeutung und Verwendung der Parameter ist für den Operator set_metrology_object_param beschrieben.

  'measure_sigma', *'measure_threshold'*, *'measure_transition'*, *'measure_select'*: Die Bedeutung und Verwendung der Parameter ist für den Operator measure_pos mit den Parametern Sigma, Threshold, Transition und Select beschrieben.

  'measure_interpolation': Die Bedeutung und Verwendung des Parameters ist für den Operator gen_measure_rectangle2 mit dem Parameter Interpolation beschrieben.

  'measure_distance_min': Gibt den minimalen Abstand zwischen den Zentren der erstellten Messregionen zurück, welcher abhängig von der Geometrie des Objekts und dem Wert des Eingabeparameters *'measure_distance'* oder dem Wert des Eingabeparameters *'num_measures'* des Operators set_metrology_object_param ist. Für ein Metrology-Messobjekt Kreis oder Linie ist der Abstand zwischen den Messregionen gleich verteilt. In diesem Fall geben *'measure_distance_min'* und *'measure_distance_max'* den gleichen Wert zurück.

  'measure_distance_max': Gibt den maximalen Abstand zwischen den Zentren der erstellten Messregionen zurück, welcher abhängig von der Geometrie des Objekts und dem Wert des Eingabeparameters *'measure_distance'* oder dem Wert des Eingabeparameters *'num_measures'* des Operators set_metrology_object_param ist. Für ein Metrology-Messobjekt Kreis oder Linie ist der Abstand zwischen den Messregionen gleich verteilt. In diesem Fall geben *'measure_distance_min'* und *'measure_distance_max'* den gleichen Wert zurück.

  'num_measures': Gibt die Anzahl der erstellten Messregionen zurück, welche abhängig von der Geometrie des Objekts und dem Wert des Eingabeparameters *'measure_distance'* oder dem Wert des Eingabeparameters *'num_measures'* des Operators set_metrology_object_param ist.

  'object_type': Typ der geometrischen Form des Metrology-Messobjekts. Ist das Metrology-Messobjekt vom Typ Kreis, enthält der Ausgabeparameter GenParamValue die Wert *'circle'*. Ist das Metrology-Messobjekt vom Typ Ellipse, enthält der Ausgabeparameter GenParamValue die Wert *'ellipse'*. Ist das Metrology-Messobjekt vom Typ Linie, enthält der Ausgabeparameter GenParamValue die Wert *'line'*. Ist das Metrology-Messobjekt vom Typ Rechteck, enthält der Ausgabeparameter GenParamValue die Wert *'rectangle'*.

  'object_params': Die Parameter der geometrischen Form des Metrology-Messobjekts. Ist das Metrology-Messobjekt vom Typ Kreis, enthält der Ausgabeparameter GenParamValue die Geometrie des Kreises in der Reihenfolge: *'row'*, *'column'*, *'radius'*. Die Bedeutung und Verwendung der Parameter ist für den Operator add_metrology_object_circle_measure beschrieben. Ist das

Metrology-Messobjekt vom Typ Ellipse, enthält der Ausgabeparameter GenParamValue die Geometrie der Ellipse in der Reihenfolge: *'row'*, *'column'*, *'phi'*, *'radius1'*, *'radius2'*. Die Bedeutung und Verwendung der Parameter ist für den Operator add_metrology_object_ellipse_measure beschrieben. Ist das Metrology-Messobjekt vom Typ Linie, enthält der Ausgabeparameter GenParamValue die Geometrie der Linie in der Reihenfolge: *'row_begin'*, *'column_begin'*, *'row_end'*, *'column_end'*. Die Bedeutung und Verwendung der Parameter ist für den Operator add_metrology_object_line_measure beschrieben. Ist das Metrology-Messobjekt vom Typ Rechteck, enthält der Ausgabeparameter GenParamValue die Geometrie des Rechtecks in der Reihenfolge: *'row'*, *'column'*, *'phi'*, *'length1'*, *'length2'*. Die Bedeutung und Verwendung der Parameter ist für den Operator add_metrology_object_rectangle2_measure beschrieben.

- Nur gültig für ein Metrology-Messobjekt vom Typ Kreis:

  'row', *'column'*, *'radius'*: Dies sind Parameter eines Metrology-Messobjekts vom Typ Kreis. Die Bedeutung und Verwendung der Parameter ist für den Operator add_metrology_object_circle_measure beschrieben.

- Nur gültig für ein Metrology-Messobjekt vom Typ Ellipse:

  'row', *'column'*, *'phi'*, *'radius1'*, *'radius2'*: Dies sind Parameter eines Metrology-Messobjekts vom Typ Ellipse. Die Bedeutung und Verwendung der Parameter ist für den Operator add_metrology_object_ellipse_measure beschrieben.

- Nur gültig für ein Metrology-Messobjekt vom Typ Linie:

  'row_begin', *'column_begin'*, *'row_end'*, *'column_end'*: Dies sind Parameter eines Metrology-Messobjekts vom Typ Linie. Die Bedeutung und Verwendung der Parameter ist für den Operator add_metrology_object_line_measure beschrieben.

- Nur gültig für ein Metrology-Messobjekt vom Typ Rechteck:

  'row', *'column'*, *'phi'*, *'length1'*, *'length2'*: Dies sind Parameter eines Metrology-Messobjekts vom Typ Rechteck. Die Bedeutung und Verwendung der Parameter ist für den Operator add_metrology_object_rectangle2_measure beschrieben.

Parameter

▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **Index** (input_control) . . . . . integer(-array) ⇝ *string* / integer
Index der Metrology-Messobjekte.
Defaultwert: 'all'
Wertevorschläge: Index ∈ {'all', 0, 1, 2}

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'num_measures'
Werteliste: GenParamName ∈ {'column', 'column_begin', 'column_end', 'distance_threshold', 'end_phi', 'instances_outside_measure_regions', 'length1', 'length2', 'max_num_iterations', 'measure_distance_min', 'measure_distance_min', 'measure_interpolation', 'measure_length1', 'measure_length2', 'measure_select', 'measure_sigma', 'measure_threshold', 'measure_transition', 'min_score', 'num_instances', 'num_measures', 'object_params', 'object_type', 'phi', 'point_order', 'radius', 'radius1', 'radius2', 'rand_seed', 'row', 'row_begin', 'row_end', 'start_phi', 'x', 'y', 'x_begin', 'y_begin', 'x_end', 'y_end'}

▷ **GenParamValue** (output_control) . . . . . attribute.value-array ⇝ *string* / real / integer
Werte der generischen Parameter.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_metrology_object_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_metrology_object_indices, set_metrology_object_param

Nachfolger

set_metrology_object_param

Siehe auch

get_metrology_object_fuzzy_param, get_metrology_object_num_instances

Modul

2D Metrology

get_metrology_object_result (: : MetrologyHandle, Index, Instance, GenParamName, GenParamValue : Parameter)

Abfragen der Ergebnisse der Anpassung des Messmodells.

get_metrology_object_result erlaubt den Zugriff auf die Ergebnisse der Messungen der Metrology-Messobjekte, die von apply_metrology_model berechnet wurden.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt fest für welche Metrology-Messobjekte die Ergebnisse abgefragt werden sollen. Wird für den Parameter Index als Wert *'all'* übergeben, werden die Ergebnisse aller Metrology-Messobjekte abgefragt. Mit dem Parameter Instance wird angegeben, welche Instanz der Ergebnisse in Parameter zurückgegeben wird. Wird Instance auf *'all'* gesetzt, werden alle Instanzen zurückgegeben. Es können verschiedene generische Parameter angegeben werden, mit denen die Rückgabewerte in Parameter gesteuert werden können. Die generischen Parameternamen werden in GenParamName gesetzt. Die dazugehörigen Werte werden im Parameter GenParamValue übergeben.

Folgende Parameter und Werte sind möglich:

'result_type': Setzt man GenParamName auf *'result_type'* kann mittels GenParamName bestimmt werden, wie und welche Resultate für ein Metrology-Messobjekt ausgegeben werden. Alle für ein Metrology-Messobjekt gemessenen Parameter können auf einmal abgerufen werden. Weiterhin können spezielle Parameter oder der Score abgerufen werden.

Abfragen aller Parameter: Wenn GenParamValue auf *'all_param'* gesetzt wurde, dann werden alle für ein Metrology-Messobjekt gemessenen Parameter zurückgegeben. Wurden Kameraparameter und eine Messebene gesetzt (siehe set_metrology_model_param), so werden die Ergebnisse in metrischen Koordinaten zurückgegeben, andernfalls in Pixeln.
Für einen *Kreis* ist, werden die Mittelpunktskoordinaten (Zeile *'row'* und Spalte *'column'*) sowie der Radius *'radius'* des Kreises zurückgegeben. Die Reihenfolge ist [*'row'*, *'column'*, *'radius'*] oder [*'x'*, *'y'*, *'radius'*].
Für eine *Ellipse* ist, werden die Mittelpunktskoordinaten (Zeile *'row'* und Spalte *'column'*), die Orientierung der Hauptachse *'phi'*, die Länge der großen Halbachse *'radius1'* sowie die Länge der kleinen Halbachse *'radius2'* der Ellipse zurückgegeben. Die Reihenfolge ist [*'row'*, *'column'*, *'phi'*, *'radius1'*, *'radius2'*] oder [*'x'*, *'y'*, *'phi'*, *'radius1'*, *'radius2'*].
Für eine *Linie* ist, wird der Start- und Endpunkt der Linie zurückgegeben Die Reihenfolge ist [*'row_begin'*, *'column_begin'*, *'row_end'*, *'column_end'*] oder [*'x_begin'*, *'y_begin'*, *'x_end'*, *'y_end'*].
Für ein *Rechteck* ist, werden die Mittelpunktskoordinaten (Zeile *'row'* und Spalte *'column'*), die Orientierung der Hauptachse *'phi'*, der erste Halbmesser (halbe Länge) *'length1'* und der zweite Halbmesser (halbe Länge) *'length2'* des Rechtecks zurückgegeben. Die Reihenfolge ist [*'row'*, *'column'*, *'phi'*, *'length1'*, *'length2'*] oder [*'x'*, *'y'*, *'phi'*, *'length1'*, *'length2'*].

Abfragen spezieller Parameter: Gemessene Objektparameter können auch einzeln abgerufen werden, indem GenParamName auf den gewünschten Parameternamen gesetzt wird.
Wurden *keine Kameraparameter* und keine Messebene gesetzt, können die folgenden Parameter einzeln abgerufen werden, wenn sie für das jeweilige Metrology-Messobjekts verfügbar sind. Bitte beachten Sie, dass für Linien zusätzlich die drei Parameter der Hesse-Normalform abgefragt werden können, d.h. der Einheitsvektor der Normale *'nrow'*, *'ncolumn'* und der orthogonale Abstand [*distance*] der Linie vom Ursprung des Koordinatensystems. Das Vorzeichen des Abstands legt die Seite der Linie fest, auf welcher sich der Ursprung befindet.

Wertevorschläge: 'row', 'column', 'radius', 'phi' , 'radius1', 'radius2', 'length1', 'length2', 'row_begin', 'column_begin', 'row_end', 'column_end', 'nrow', 'ncolumn', 'distance'
Wurden *Kameraparameter* und eine Messebene gesetzt, können die folgenden metrischen Parameter einzeln abgerufen werden, wenn sie für das jeweilige Metrology-Messobjekts verfügbar sind. Bitte beachten Sie, dass für Linien zusätzlich die drei Parameter der Hesse-Normalform abgefragt werden können, d.h. der Einheitsvektor der Normale *'nx'*, *'ny'* und der orthogonale Abstand [*distance*] der Linie vom Ursprung des Koordinatensystems. Das Vorzeichen des Abstands legt die Seite der Linie fest, auf welcher sich der Ursprung befindet.
Wertevorschläge: 'x', 'y', 'radius', 'phi' , 'radius1', 'radius2', 'length1', 'length2', 'x_begin', 'y_begin', 'x_end', 'y_end', 'nx', 'ny', 'distance'

Abfragen des Scores: Wird `GenParamName` auf den Wert *'score'* gesetzt, wird der Score der Ergebnisse zurückgegeben. Der Score ist die Anzahl der Messungen, die für die Berechnung des Ergebnisses verwendet werden dividiert durch die maximale Anzahl von Messregionen.

'used_edges': Mit den folgenden Werten für `GenParamValue` können Details zu den Kanten abgefragt werden, die für die Anpassung eines Metrology-Messobjekts verwendet wurden.

'row': Die Zeilenkoordinaten der Kanten, die für die Anpassung eines Metrology-Messobjektes verwendet wurden, werden zurückgegeben.

'column': Die Spaltenkoordinaten der Kanten, die für die Anpassung eines Metrology-Messobjektes verwendet wurden, werden zurückgegeben.

'amplitude': Die Amplitude der Kanten, die für die Anpassung eines Metrology-Messobjektes verwendet wurden, werden zurückgegeben.

Wertevorschläge: 'row', 'column', 'amplitude'

'angle_direction': Der Parameter legt die Drehrichtung für Winkel fest, die Ergebnisse der Anpassung sind. Wird der Parameter *'angle_direction'* auf *'positive'* gesetzt, ist der Winkel zwischen der Hauptachse des Objekts und der horizontalen Achse des Koordinatensystems im mathematisch positiven Sinn (gegen den Uhrzeigersinn) definiert. Wird der Parameter *'angle_direction'* auf *'negative'* gesetzt, ist der Winkel zwischen der Hauptachse des Objekts und der horizontalen Achse des Koordinatensystems im mathematisch negativen Sinn (im Uhrzeigersinn) definiert. Die Winkelergebnisse werden im Bogenmaß ausgegeben.

Wertevorschläge: 'positive', 'negative'

Defaultwert: 'positive'

Es ist möglich die Ergebnisse mehrerer Metrology-Messobjekte (siehe Parameter `Index`) und mehrerer Instanzen (siehe Parameter `Instance`) der Metrology-Messobjekte gleichzeitig zurückzugeben. Die Ergebnisse werden in `Parameter` in der folgenden Reihenfolge zurückgegeben: 1. Instanz des 1. Metrology-Messobjekts, 2. Instanz des 1. Metrology-Messobjekts, usw., 1. Instanz des 2. Metrology-Messobjekts, 2. Instanz des 2. Metrology-Messobjekts, usw.

Parameter

▷ **`MetrologyHandle`** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **`Index`** (input_control) ..... integer(-array) ⇝ *integer* / string
Index des Metrology-Messobjektes.
Defaultwert: 0
Wertevorschläge: `Index` ∈ {'all', 0, 1, 2}

▷ **`Instance`** (input_control) ..... integer(-array) ⇝ *string* / integer
Instanz des Metrology-Messobjektes
Defaultwert: 'all'
Wertevorschläge: `Instance` ∈ {'all', 0, 1, 2}

▷ **`GenParamName`** (input_control) ..... attribute.name-array ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'result_type'
Werteliste: `GenParamName` ∈ {'result_type', 'angle_direction', 'used_edges'}

▷ **`GenParamValue`** (input_control) ..... attribute.value-array ⇝ *string* / real
Wert des generischen Parameters.
Defaultwert: 'all_param'
Wertevorschläge: `GenParamValue` ∈ {'all_param', 'score', 'true', 'false', 'row', 'column', 'amplitude', 'radius', 'phi', 'radius1', 'radius2', 'length1', 'length2', 'row_begin', 'column_begin', 'row_end',

'column_end', 'nrow', 'ncolumn', 'distance', 'x', 'y', 'x_begin', 'y_begin', 'x_end', 'y_end', 'nx', 'ny', 'positive', 'negative'}

▷ **Parameter** (output_control) ..... real(-array) ⇝ *real* / integer / string
Ergebniswerte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_metrology_object_result` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`apply_metrology_model`

Nachfolger

`clear_metrology_model`

Siehe auch

`get_metrology_object_result_contour`, `get_metrology_object_measures`

Modul

2D Metrology

get_metrology_object_result_contour (: Contour : MetrologyHandle, Index, Instance, Resolution :)

Abfragen der Ergebniskonturen eines Metrology-Messobjektes.

`get_metrology_object_result_contour` gibt für die gewählten Metrology-Messobjekte und Instanzen in `Contour` die Ergebniskonturen einer Messung mit `apply_metrology_model` in Bildkoordinaten zurück.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle `MetrologyHandle` definiert. Der Parameter `Index` legt fest für welche Metrology-Messobjekte die Ergebnisse abgefragt werden sollen. Wird für den Parameter `Index` als Wert *'all'* übergeben, werden die Ergebnisse aller Metrology-Messobjekte abgefragt. Wurden für ein Metrology-Messobjekt mehrere Ergebnisse (Instanzen) berechnet, so kann mit dem Parameter `Instance` angegeben werden, für welche Instanz die Kontur zurückgegeben werden soll. Wird `Instance` auf *'all'* gesetzt, werden die Konturen für alle Instanzen zurückgegeben.

Die Auflösung der resultierenden XLD Kontur `Contour` wird über den Parameter `Resolution` festgelegt. Er bestimmt den euklidischen Abstand n Pixeln zwischen benachbarten Konturpunkten. Unterschreitet der Eingabewert den intern gesetzten Mindestwert (1.192e-7), wird die Auflösung auf den kleinsten zulässigen Wert gesetzt.

Parameter

▷ **Contour** (output_object) ..... xld_cont(-array) ⇝ *object*
Ergebniskontur des Metrology-Messobjektes.

▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **Index** (input_control) ..... integer(-array) ⇝ *integer* / string
Index des Metrology-Messobjektes.
Defaultwert: 0
Wertevorschläge: `Index` ∈ {'all', 0, 1, 2}

▷ **Instance** (input_control) ..... integer(-array) ⇝ *string* / integer
Instanz des Metrology-Messobjektes
Defaultwert: 'all'
Wertevorschläge: `Instance` ∈ {'all', 0, 1, 2}

▷ **Resolution** (input_control) . . . . . real ⇝ *real*
Abstand zwischen benachbarten Konturpunkten.
Defaultwert: 1.5
Restriktion: Resolution >= 1.192e-7

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_metrology_object_result_contour` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`apply_metrology_model`

Siehe auch

`get_metrology_object_result`, `get_metrology_object_measures`

Modul

2D Metrology

read_metrology_model (: : FileName : MetrologyHandle)

Einlesen eines Messmodells aus einer Datei.

`read_metrology_model` liest ein Messmodell, welches mit `write_metrology_model` geschrieben wurde, aus der Datei `FileName` ein. Die Dateiendung für ein Messmodell ist in HALCON 'mtr'. Die im gelesenen Messmodell vorhanden Werte werden in einem Messmodell mit dem Handle `MetrologyHandle` gespeichert. Zugriff auf die Parameter ist z.B. mit dem Operator `get_metrology_object_param` oder `get_metrology_object_fuzzy_param` möglich.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.mtr`

▷ **MetrologyHandle** (output_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_metrology_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`get_metrology_object_indices`, `apply_metrology_model`

Siehe auch

`write_metrology_model`

Modul

2D Metrology

reset_metrology_object_fuzzy_param (: : MetrologyHandle, Index :)

Löschen aller Fuzzy-Parameter und Fuzzy-Funktionen eines Messmodells.

reset_metrology_object_fuzzy_param löscht alle Fuzzy-Parameter und Fuzzy-Funktionen von Metrology-Messobjekten, welche mit dem Operator set_metrology_object_fuzzy_param gesetzt werden können, und stellt die Defaultwerte wieder her.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt die Metrology-Messobjekte fest, die zurückgesetzt werden sollen. Wird für den Parameter Index als Wert *'all'* übergeben, werden alle Metrology-Messobjekte zurückgesetzt.

Parameter

▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **Index** (input_control) ..... integer(-array) ⇝ *string* / integer
Index der Metrology-Messobjekte.
Defaultwert: 'all'
Wertevorschläge: Index ∈ {'all', 0, 1, 2}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert reset_metrology_object_fuzzy_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

set_metrology_object_fuzzy_param

Siehe auch

reset_metrology_object_param

Modul

2D Metrology

reset_metrology_object_param (: : MetrologyHandle, Index :)

Löschen aller Parameter eines Messmodells.

reset_metrology_object_param löscht alle Einstellungen der Parameter von Metrology-Messobjekten, welche mit dem Operator set_metrology_object_param gesetzt werden können, und stellt die Defaultwerte wieder her.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt die Metrology-Messobjekte fest, die zurückgesetzt werden sollen. Wird für den Parameter Index als Wert *'all'* übergeben, werden alle Metrology-Messobjekte zurückgesetzt.

Parameter

▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **Index** (input_control) . . . . . integer(-array) ⇝ *string* / integer
Index der Metrology-Messobjekte.
Defaultwert: 'all'
Wertevorschläge: `Index` ∈ {'all', 0, 1, 2}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `reset_metrology_object_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`set_metrology_object_param`

Siehe auch

`reset_metrology_object_fuzzy_param`

Modul

2D Metrology

serialize_metrology_model (: : MetrologyHandle : SerializedItemHandle)

Serialisieren eines Messmodells.

`serialize_metrology_model` serialisiert die Daten eines Messmodells (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung).

Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_metrology_model` in eine Datei geschrieben werden. Das Messmodell wird in dem Handle `MetrologyHandle` übergeben. Das serialisierte Messmodell wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_metrology_model` wieder deserialisiert werden.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Parameter

▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_metrology_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_metrology_model`, `add_metrology_object_circle_measure`, `add_metrology_object_ellipse_measure`, `add_metrology_object_line_measure`, `add_metrology_object_rectangle2_measure`, `set_metrology_object_param`, `set_metrology_object_fuzzy_param`, `read_metrology_model`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_metrology_model`

Modul

2D Metrology

`set_metrology_model_image_size` (: : `MetrologyHandle`, `Width`, `Height` :)

Setzen der Bildgröße für die Metrology-Messobjekte.

`set_metrology_model_image_size` erlaubt es, die bei der im Rahmen der 2D-Messtechnik benötigten Kantendetektion verwendete Bildgröße zu setzen.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle `MetrologyHandle` definiert. Die Bildbreite wird mittels des Parameters `Width` und die Bildhöhe mittels des Parameters `Height` angegeben.

Achtung

Zu beachten ist, dass der Operator `set_metrology_model_image_size` vor dem Hinzufügen von Metrology-Messobjekten zum Messmodell z.B. mit dem Operator `add_metrology_object_circle_measure`, `add_metrology_object_ellipse_measure`, `add_metrology_object_line_measure`, oder `add_metrology_object_rectangle2_measure` aufgerufen werden sollte. Andernfalls werden alle Messregionen der schon vorhanden Metrology-Messobjekte nach dem Aufruf von `set_metrology_model_image_size` oder `apply_metrology_model` automatisch neu berechnet.

Parameter

▷ **`MetrologyHandle`** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **`Width`** (input_control) . . . . . extent.x ⇝ *integer*
Breite des zu verarbeitenden Bildes.
Defaultwert: 640
Wertevorschläge: `Width` ∈ {128, 192, 256, 512, 640, 768, 1024, 1280, 2048}
Typischer Wertebereich: 0 ≤ Width (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 16

▷ **`Height`** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des zu verarbeitenden Bildes.
Defaultwert: 480
Wertevorschläge: `Height` ∈ {128, 192, 256, 512, 640, 768, 1024, 1280, 2048}
Typischer Wertebereich: 0 ≤ Height (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 16

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_metrology_model_image_size` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_metrology_model`

Nachfolger

`set_metrology_model_param`, `add_metrology_object_circle_measure`, `add_metrology_object_ellipse_measure`, `add_metrology_object_line_measure`, `add_metrology_object_rectangle2_measure`, `add_metrology_object_generic`

Modul

2D Metrology

set_metrology_model_param (: : MetrologyHandle, GenParamName, GenParamValue :)

Setzen von Parametern, die für das gesamte Messmodell gelten.

`set_metrology_model_param` setzt Parameter des Messmodells `MetrologyHandle`, die für das gesamte Modell, d.h. für alle Objekte, gültig sind.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Die folgenden Werte können für `GenParamName` und `GenParamValue` gesetzt werden:

Kalibrierung

Um mit `apply_metrology_model` kalibrierte Messungen in metrischen Koordinaten durchführen zu können, müssen sowohl die Werte der internen Kameraparameter als auch die 3D-Lage der Messebene gesetzt werden.

'camera_param': In vielen Fällen sind die internen Kameraparameter das Ergebnis einer vorherigen Kamerakalibrierung mit dem Operator `calibrate_cameras` (siehe Kalibrierung für die Reihenfolge der internen Kameraparameter und das zu Grunde liegende Kameramodell). Falls *'camera_param'* auf [] gesetzt wird, werden die Werte der internen Kameraparameter gelöscht.

Default value: [].

'plane_pose': Die 3D-Lage der Messebene in Kamerakoordinaten. Falls *'plane_pose'* auf [] gesetzt wird, wird die Messebene gelöscht.

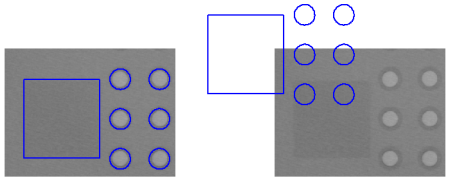
Default value: [].

Definition des Referenzkoordinatensystems

Wenn Metrology-Messobjekte z.B. mit `add_metrology_object_generic`, `add_metrology_object_circle_measure` etc. zum Messmodell hinzugefügt werden, werden Positionen und Orientierungen bzgl. des Bildkoordinatensystems angegeben, welches seinen Ursprung in der linken oberen Bildecke hat. In manchen Fällen ist es nötig, das Referenzsystem bzgl. dem die Metrology-Messobjekte angegeben sind zu ändern. Dies ist z.B. der Fall, wenn ein Formmodell genutzt wird um das Messmodell im aktuellen Bild auszurichten. Die Ergebnisse von `find_shape_model` können nur direkt in `align_metrology_model` genutzt werden, wenn das Referenzsystem, in dem das Messmodel gegeben ist, mit dem Referenzsystems des Formmodells übereinstimmt (siehe `align_metrology_model` für ausführlichere Informationen)

'reference_system': Das in `GenParamValue` angegebene Tupel hat die Form [row, column, angle]. Per default ist das Referenzsystem das Bildkoordinatensystem, das sich in der linken oberen Ecke des Bildes befindet. Wird das Referenzkoordinatensystem neu gesetzt, so gibt das Tupel [row, column, angle] die Translation und Rotation des neuen Referenzkoordinatensystems bezüglich des Bildkoordinatensystems an. Alle Bestandteile (Objekte) des Messmodells werden in das neu gesetzte Koordinatensystem umgerechnet. Im folgenden Bild wird eine neues Referenzsystem auf den Mittelpunkt des Bildes gesetzt.

```
set_metrology_model_param (MetrologyHandle, 'reference_system',
[Height/2,Width/2,0])
```



(1) (2)

(1) Mehrere Metrology-Messobjekte und ihre Konturen sind in blau gezeigt. (2) Das neue Referenzsystem des Messmodells wird im Mittelpunkt des Bildes platziert. Als Folge davon, werden die Posen der Metrology-Messobjekte in die entgegen gesetzte Richtung bewegt. Die resultierenden Konturen der Metrology-Messobjekte sind in blau dargestellt.

Default value: [0, 0, 0]

Skalierung der Ergebnisse

Die Ergebnisse der Messung, die mit `get_metrology_object_result` abgerufen werden, können mit einem Faktor skaliert werden.

'scale': Der Parameter legt die Skalierung oder die Einheit der Ergebnisse der Messungen fest. Der Parameter *'scale'* entspricht dem Verhältnis der *gewünschten Einheit* zur *ursprünglichen Einheit*.

Sind keine Kameraparameter gegeben, so ist die Standardeinheit Pixel.

Wurden *'camera_param'* and *'plane_pose'* gesetzt, so wird die Einheit durch die Koordinaten des Kalibrierungobjektes bestimmt. Die standard HALCON Kalibrierplatte ist in metrischen Koordinaten definiert. Wird sie für die Kalibrierung verwendet, kann die gewünschte Einheit direkt gesetzt werden. Die Zuordnung von Einheiten zu Faktoren ergibt sich aus der folgenden Tabelle:

Einheit	Skalierungsfaktor
m	1
dm	10
cm	100
mm	1000
um, microns	10^6

Wertevorschläge: 1.0, 0.1, 'm', 'cm', 'mm', 'microns', 'um'
Defaultwert: 1.0

Parameter

▷ **`MetrologyHandle`** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **`GenParamName`** (input_control) ..... attribute.name ⇝ *string*
Name des generischen Parameter.
Defaultwert: 'camera_param'
Werteliste: `GenParamName` ∈ {'camera_param', 'plane_pose', 'scale', 'reference_system'}

▷ **`GenParamValue`** (input_control) ..... attribute.value(-array) ⇝ *string* / real / integer
Wert des generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {1.0, 0.1, 'm', 'cm', 'mm', 'microns', 'um'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_metrology_model_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_metrology_model`, `set_metrology_model_image_size`

Nachfolger

`add_metrology_object_generic`, `get_metrology_object_model_contour`

Siehe auch

`set_metrology_object_param`, `align_metrology_model`, `get_metrology_model_param`

Modul

2D Metrology

`set_metrology_object_fuzzy_param` `( : : MetrologyHandle, Index, GenParamName, GenParamValue : )`

Setzen der Fuzzy-Parameter oder Fuzzy-Funktionen eines Messmodells.

`set_metrology_object_fuzzy_param` erlaubt es, Fuzzy-Parameter oder Fuzzy-Funktionen eines Metrology-Messobjekts zu setzen oder zu verändern, um damit das Verhalten des Modells bei der Auswahl der Kanten mittels des Operators `apply_metrology_model` zu beeinflussen.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle `MetrologyHandle` definiert. Der Parameter `Index` legt die Metrology-Messobjekte fest, für die die Parameter gesetzt werden sollen. Die Parameter aller Metrology-Messobjekte werden gesetzt wenn für den Parameter `Index` *'all'* übergeben wird.

Der Fuzzy-Parameter oder der Typ der Fuzzy-Funktion werden im Parameter `GenParamName` übergeben. Der korrespondierende Wert oder die Fuzzy-Funktion wird im Parameter `GenParamValue` übergeben. Ist mindestens eine Fuzzy-Funktion gesetzt, wird bei der Suche der Metrology-Messobjekte mit dem Operator `apply_metrology_model` intern der Operator `fuzzy_measure_pos` verwendet. Mehr Information über Fuzzy-Funktionen ist daher im Operator `fuzzy_measure_pos` zu finden.

Es folgt eine Übersicht über die verschiedenen generischen Parameter und Parameterwerte. Die folgenden generischen Parameter `GenParamName` und die dazugehörigen Werte `GenParamValue` sind verfügbar:

'fuzzy_thresh': Der Parameter legt die minimale Fuzzy Bewertung fest. Die Bedeutung und Verwendung des Parameters ist für den Operator `fuzzy_measure_pos` beschrieben und entspricht dort dem Parameter `FuzzyThresh`.

Defaultwert: 0.5

'function_contrast': Der Parameter legt eine Fuzzy-Funktion des Typs *contrast* fest. Die Bedeutung und Verwendung des Parameters ist für den Operator `set_fuzzy_measure` beschrieben. Der Parameter entspricht dem Parameter `SetType` mit dem Wert *'contrast'* und sein Wert dem Parameter `Function`.

Defaultwert: 'disabled'

'function_position': Der Parameter legt eine Fuzzy-Funktion des Typs *position* fest. Die Bedeutung und Verwendung des Parameters ist für den Operator `set_fuzzy_measure` beschrieben. Der Parameter entspricht dem Parameter `SetType` mit dem Wert *'position'* und sein Wert dem Parameter `Function`.

Defaultwert: 'disabled'

'function_position_center': Der Parameter legt eine Fuzzy-Funktion des Typs *position_center* fest. Die Bedeutung und Verwendung des Parameters ist für den Operator `set_fuzzy_measure` beschrieben. Der Parameter entspricht dem Parameter `SetType` mit dem Wert *'position_center'* und sein Wert dem Parameter `Function`.

'function_position_end': Der Parameter legt eine Fuzzy-Funktion des Typs *position_end* fest. Die Bedeutung und Verwendung des Parameters ist für den Operator `set_fuzzy_measure` beschrieben. Der Parameter entspricht dem Parameter `SetType` mit dem Wert *'position_end'* und sein Wert dem Parameter `Function`.

Defaultwert: 'disabled'

'function_position_first_edge': Der Parameter legt eine Fuzzy-Funktion des Typs *position_first_edge* fest. Die Bedeutung und Verwendung des Parameters ist für den Operator `set_fuzzy_measure` beschrieben. Der Parameter entspricht dem Parameter `SetType` mit dem Wert *'position_first_edge'* und sein Wert dem Parameter `Function`.

Defaultwert: 'disabled'

'function_position_last_edge': Der Parameter legt eine Fuzzy-Funktion des Typs *position_last_edge* fest. Die Bedeutung und Verwendung des Parameters ist für den Operator `set_fuzzy_measure` beschrieben. Der Parameter entspricht dem Parameter `SetType` mit dem Wert *'position_last_edge'* und sein Wert dem Parameter `Function`.

Defaultwert: 'disabled'

Wird der Funktionswert für eine Fuzzy-Funktion auf *'disabled'* gesetzt, wird nur diese Fuzzy-Funktion gelöscht. Alle definierten Fuzzy-Funktionen und Fuzzy-Parameter werden mit Hilfe des Operators `reset_metrology_object_fuzzy_param` gelöscht. Die aktuelle Konfiguration kann mit Hilfe von `get_metrology_object_fuzzy_param` ausgelesen werden. Zu beachten ist, dass nur, wenn mindestens eine Fuzzy-Funktion gesetzt ist, der Operator `fuzzy_measure_pos` bei der Kantendetektion mit `apply_metrology_model` verwendet wird. Ansonsten wird `measure_pos` verwendet.

Parameter

▷ **`MetrologyHandle`** (input_control) ..... metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **`Index`** (input_control) ..... integer(-array) ⇝ *string* / integer
Index der Metrology-Messobjekte.
Defaultwert: 'all'
Wertevorschläge: `Index` ∈ {'all', 0, 1, 2}

▷ **`GenParamName`** (input_control) ..... attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'fuzzy_thresh'
Werteliste: `GenParamName` ∈ {'function_contrast', 'function_position', 'function_position_center', 'function_position_end', 'function_position_first_edge', 'function_position_last_edge', 'fuzzy_thresh'}

▷ **`GenParamValue`** (input_control) ..... attribute.value-array ⇝ *real* / integer
Werte der generischen Parameter.
Defaultwert: 0.5
Wertevorschläge: `GenParamValue` ∈ {0.1, 0.3, 0.5, 0.6, 0.7, 0.9, 1, 2, 3, 4, 5, 10, 20}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_metrology_object_fuzzy_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`get_metrology_object_fuzzy_param`

Nachfolger

`apply_metrology_model`, `reset_metrology_object_fuzzy_param`, `get_metrology_object_fuzzy_param`

Siehe auch

`set_metrology_object_param`

Modul

2D Metrology

`set_metrology_object_param` (: : `MetrologyHandle`, `Index`, `GenParamName`, `GenParamValue` :)

Setzen von Parametern für die Metrology-Messobjekte eines Messmodells.

`set_metrology_object_param` erlaubt es, zahlreiche Parameter der Metrology-Messobjekte zu verändern.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle `MetrologyHandle` definiert. Der Parameter `Index` legt die Metrology-Messobjekte fest, für die die Parameter gesetzt werden sollen. Die Parameter aller Metrology-Messobjekte werden gesetzt, wenn für den Parameter `Index` *'all'* übergeben wird. Alle Parameter können auch direkt beim Anlegen des Metrology-Messobjekts mit `add_metrology_object_generic`, `add_metrology_object_circle_measure`, `add_metrology_object_ellipse_measure`, `add_metrology_object_line_measure` oder `add_metrology_object_rectangle2_measure` übergeben werden. Die aktuelle Konfiguration kann mit Hilfe von `get_metrology_object_param` ausgelesen werden. Alle Parameter, die mit dem Operator `set_metrology_object_param` gesetzt werden können, können mit dem Operator `reset_metrology_object_param` zurückgesetzt werden.

Im Folgenden werden alle generischen Parameter mit den Defaultwerten aufgelistet. In den meisten Fällen reichen die Standardwerte aus, so dass keine Anpassung nötig ist. Die folgenden generischen Parameter `GenParamName` und die dazugehörigen Werte `GenParamValue` sind verfügbar - sortiert nach verschiedenen Kategorien:

Erstellen der Messregionen:

'measure_length1': Der Wert des Parameters legt die halbe Kantenlänge der Messregionen senkrecht zum Objektrand fest. Der Wert entspricht der Messtoleranz. Die Einheit des Wertes ist Pixel.

Wertevorschläge: 10.0, 20.0, 30.0

Defaultwert: 20.0

Restriktion: 'measure_length1' >= 1.0

'measure_length2': Der Wert des Parameters legt die halbe Kantenlänge der Messregionen tangential zum Objektrand fest. Die Einheit des Wertes ist Pixel.

Wertevorschläge: 3.0, 5.0, 10.0

Defaultwert: 5.0

Restriktion: 'measure_length2' >= 0.0

'measure_distance': Der Wert des Parameters legt den gewünschten Abstand zwischen den Mittelpunkten zweier Messregionen fest. Führt der Abstand zu zu wenig Messregionen, hat dieser Parameter keinen Einfluss auf die Anzahl an Messregionen und wird auf die benötigte Mindestanzahl an Messregionen erhöht. Für ein Metrology-Messobjekt vom Typ Kreis ist diese Anzahl 3. Für ein Metrology-Messobjekt vom Typ Ellipse ist diese Anzahl 5. Für ein Metrology-Messobjekt vom Typ Linie ist diese Anzahl 2. Für ein Metrology-Messobjekt vom Typ Rechteck ist diese Anzahl pro Seite 2, d.h. insgesamt 8. Wird dieser Wert gesetzt, hat der Wert des Parameters *'num_measures'* keinen Einfluss. Die Einheit des Wertes ist Pixel.

Wertevorschläge: 5.0, 15.0, 20.0, 30.0

Defaultwert: 10.0

'num_measures': Der Wert des Parameters legt die gewünschte Anzahl der Messregionen fest. Wird der Wert zu klein gewählt, hat dieser Parameter keinen Einfluss auf die Anzahl an Messregionen und wird auf die benötigte Mindestanzahl an Messregionen erhöht. Für ein Metrology-Messobjekt vom Typ Kreis ist diese Anzahl 3. Für ein Metrology-Messobjekt vom Typ Ellipse ist diese Anzahl 5. Für ein Metrology-Messobjekt vom Typ Linie ist diese Anzahl 2. Für ein Metrology-Messobjekt vom Typ Rechteck ist diese Anzahl pro Seite 2, d.h. insgesamt 8. Wird dieser Wert gesetzt, hat der Wert des Parameters *'measure_distance'* keinen Einfluss.

Wertevorschläge: 8, *10*, *16*, *20*, *30*, *50*, *100*

Kantendetektion:

'measure_sigma': Der Wert des Parameters legt das Sigma für die Gaußglättung fest. Die Bedeutung, die Verwendung und der Defaultwert des Parameters ist für den Operator `measure_pos` mit den Parametern `Sigma`, `Threshold` und `Select` beschrieben.

'measure_threshold': Der Wert des Parameters legt die minimale Amplitude einer Kante fest. Die Bedeutung, die Verwendung und der Defaultwert des Parameters ist für den Operator `measure_pos` mit dem Parameter `Threshold` beschrieben.

'measure_select': Der Wert des Parameters legt die Auswahl der Endpunkte der Kanten fest. Die Bedeutung, die Verwendung und der Defaultwert des Parameters ist für den Operator `measure_pos` mit dem Parameter `Select` beschrieben.

'measure_transition': Der Wert des Parameters legt fest, ob die Hell-Dunkel- oder Dunkel-Hell-Übergänge für die Kanten genutzt werden. Die Bedeutung und die Verwendung der Werte *'all'*, *'positive'* und *'negative'*, die für den Parameter *'measure_transition'* gesetzt werden können, ist für den Operator `measure_pos` mit dem Parameter `Transition` beschrieben. Zusätzlich kann *'measure_transition'* auf den Wert *'uniform'* gesetzt werden. Dann werden alle positiven Kanten (Hell-Dunkel-Übergänge) und alle negativen Kanten (Dunkel-Hell-Übergänge) bei der Kantendetektion detektiert, aber beim Anpassen der geometrischen Formen werden die Kanten unterschiedlicher Typen getrennt behandelt, d.h. für jede Instanz einer geometrischen Form werden entweder nur die positiven Kanten oder nur die negativen Kanten verwendet.

Die Messrichtung innerhalb der Messregion verläuft von innen nach außen für ein Metrology-Messobjekt vom Typ Kreis, Ellipse und Rechteck. Für ein Metrology-Messobjekt vom Typ Linie verläuft die Messrichtung innerhalb der Messregion von links nach rechts, gesehen vom ersten Punkt der Linie (siehe `RowBegin` und `ColumnBegin` des Operators `add_metrology_object_line_measure`).

Wertevorschläge: 'all', *'negative'*, *'positive'*, *'uniform'*

Defaultwert: 'all'

'measure_interpolation': Der Wert des Parameters legt die Art der zu verwendenden Interpolation fest. Die Bedeutung, die Verwendung und der Defaultwert des Parameters ist für den Operator `gen_measure_rectangle2` mit dem Parameter `Interpolation` beschrieben.

Anpassen der geometrischen Formen:

'min_score': Der Parameter legt fest, welche Bewertung eine potentielle Instanz des Metrology-Messobjekts mindestens besitzen muss, damit diese als eine gültige Instanz des Metrology-Messobjekts angesehen wird. Der Score ist die Anzahl der Kanten, die für die Berechnung des Ergebnisses verwendet wird, dividiert durch die maximale Anzahl von Messregionen (siehe `apply_metrology_model`). Falls erwartet werden kann, dass alle Kanten des Metrology-Messobjekts in den Modellregionen vorhanden sind, kann der Parameter *'min_score'* auf so hohe Werte wie *0.8* oder sogar *0.9* gesetzt werden. Zu beachten ist, dass in Bildern mit vielen Störobjekten oder mit starker Hintergrundtextur der Wert für den Parameter *'min_score'* nicht deutlich niedriger als *0.7* gesetzt werden sollte, da ansonsten falsche Instanzen des Metrology-Messobjekts gefunden werden könnten.

Wertevorschläge: 0.5, *0.7*, *0.9*

Defaultwert: 0.7

'num_instances': Der Parameter legt die maximale Anzahl der anzupassenden Instanzen für jedes Metrology-Messobjekt fest, d.h. wenn diese Anzahl an Instanzen erreicht wurde, wird die Anpassung abgebrochen (siehe `apply_metrology_model`). Der Score für erfolgreich angepasste Instanzen der geometrischen Formen muss mindestens den Wert *'min_score'* haben.

Wertevorschläge: 1, *2*, *3*, *4*

Defaultwert: 1

'distance_threshold': `apply_metrology_model` benutzt einen randomisierten Auswahlalgorithmus (RANSAC) zum Anpassen einer geometrischen Form. Damit eine Kante zu einer angepassten geometrischen Form gehört, darf die euklidische Distanz zwischen der Kante und der geometrischen Form den Wert von *'distance_threshold'* nicht übersteigen.

Wertevorschläge: 0, *1.0*, *2.0*, *3.5*, *5.0*

Defaultwert: 3.5

'max_num_iterations': Der RANSAC-Algorithmus schätzt die Anzahl von Iterationen, die benötigt werden, um die gesuchte geometrische Form anzupassen. Die Schätzung berücksichtigt dabei die Kantendaten und die Komplexität der Form. Mit dem Parameter *'max_num_iterations'* kann eine obere Grenze für die vom RANSAC-Algorithmus berechnete Anzahl an Iterationen gesetzt werden. Die Anzahl der Iterationen wird weiterhin vom RANSAC-Algorithmus geschätzt, kann den Wert von *'max_num_iterations'* aber nicht übersteigen. Das Setzen von *'max_num_iterations'*, kann hilfreich sein, wenn die Qualität der gefundenen geometrischen Form weniger wichtig ist als das Einhalten zeitlicher Grenzen. Wird der Parameter zu niedrig gesetzt, liefert der Algorithmus schlechtere oder keine Ergebnisse.

Standardmäßig ist der Wert auf -1 gesetzt, d.h. es ist keine obere Grenze für die berechnete Anzahl an Iterationen vorhanden.

Wertevorschläge: 10, *100*, *1000*

Defaultwert: -1

'rand_seed': Der Parameter legt den Startwert für den Zufallsgenerator fest, welcher vom RANSAC-Verfahren innerhalb des Operators `apply_metrology_model` zur Auswahl der Kanten genutzt wird. Wird der Parameter *'rand_seed'* auf einen Wert ungleich *0* gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit dem Wert des Parameters *'rand_seed'* initialisiert wird.

Wird der Parameter *'rand_seed'* auf den Wert *0* gesetzt, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Wertevorschläge: 0, *1*, *42*

Defaultwert: 42

'instances_outside_measure_regions': Der Parameter legt fest, welche der Ergebnisse der Messungen gültig sind. Wird der Parameter *'instances_outside_measure_regions'* auf den Wert *'false'* gesetzt, sind nur Instanzen eines Metrology-Messobjekts gültig, die innerhalb der Hauptachsen der Messregionen dieses Metrology-Messobjekts liegen. Instanzen, die ungültig sind, werden nicht gespeichert. Wird der Parameter *'instances_outside_measure_regions'* auf den Wert *'true'* gesetzt, so sind alle Instanzen des Metrology-Messobjekts gültig.

Wertevorschläge: 'false', *'true'*

Defaultwert: 'false'

Parameter

▷ **`MetrologyHandle`** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **`Index`** (input_control) . . . . . integer(-array) ⇝ *string* / integer
Index der Metrology-Messobjekte.
Defaultwert: 'all'
Wertevorschläge: `Index` ∈ {'all', 0, 1, 2}

▷ **`GenParamName`** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'num_instances'
Werteliste: `GenParamName` ∈ {'distance_threshold', 'instances_outside_measure_regions', 'max_num_iterations', 'measure_distance', 'measure_interpolation', 'measure_length1', 'measure_length2', 'measure_select', 'measure_sigma', 'measure_threshold', 'measure_transition', 'min_score', 'num_instances', 'num_measures', 'rand_seed'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value-array ⇝ *string* / real / integer
Werte der generischen Parameter.
Defaultwert: 1
Wertevorschläge: `GenParamValue` ∈ {1, 2, 3, 4, 5, 10, 20, 'all', 'true', 'false', 'first', 'last', 'positive', 'negative', 'uniform', 'nearest_neighbor', 'bilinear', 'bicubic'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_metrology_object_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`get_metrology_object_param`

Nachfolger

`apply_metrology_model`, `reset_metrology_object_param`, `get_metrology_object_param`

Siehe auch

`set_metrology_object_fuzzy_param`

Modul

2D Metrology

`write_metrology_model ( : : MetrologyHandle, FileName : )`

Schreiben eines Messmodells in eine Datei.

`write_metrology_model` schreibt ein Messmodell in die Datei `FileName`. Das Messmodell ist durch das Handle `MetrologyHandle` definiert. Das Messmodell kann mit `read_metrology_model` wieder eingelesen werden. Die Dateiendung für ein Messmodell ist in HALCON 'mtr'.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Achtung

Es ist zu beachten, dass nur die Eingabewerte gespeichert werden, d.h. es werden keine vom Operator `apply_metrology_model` erzeugten Messregionen und Ergebnisse gespeichert.

Parameter

▷ **`MetrologyHandle`** (input_control) . . . . . metrology_model ⇝ *handle*
Handle des Messmodells.

▷ **`FileName`** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.mtr`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_metrology_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`apply_metrology_model`

Nachfolger

`clear_metrology_model`

Siehe auch

`read_metrology_model`

Modul

2D Metrology

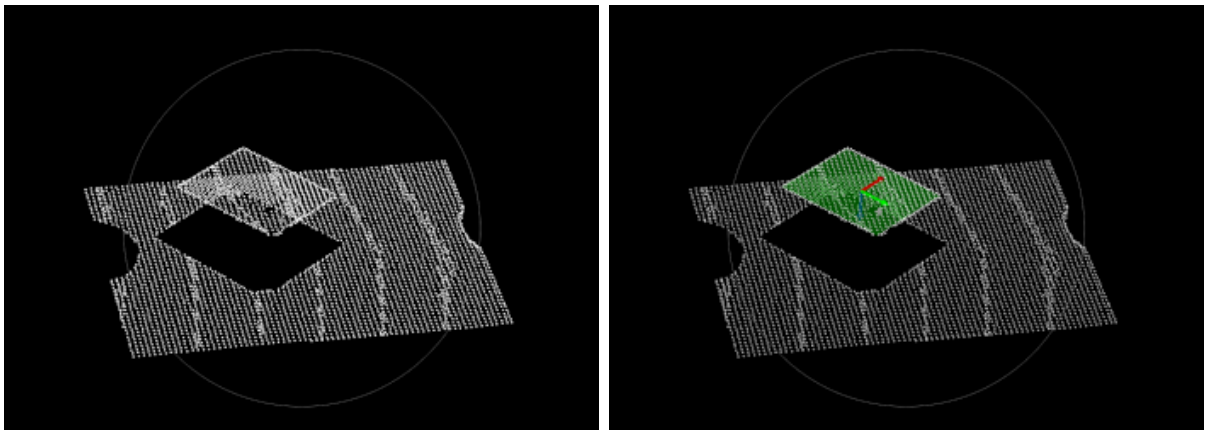
Kapitel 3

3D-Matching

Dieses Kapitel gibt einen Überblick über die verschiedenen 3D Matching Ansätze in HALCON.

3D Box Finder

Wie schon dem Namen entnommen werden kann, kann der Box Finder dazu benutzt werden um Box-förmige Objekte in 3D-Daten zu lokalisieren. Dazu ist kein Model des Objekts als Eingabe für den Operator `find_box_3d` notwendig, sondern lediglich die Seitenlängen der gesuchten Boxen. Als Ergebnis erhält man die Pose eines Greifpunkts, was insbesondere für Anwendungen bei denen ein Griff in die Kiste erfolgt sehr nützlich sein kann.



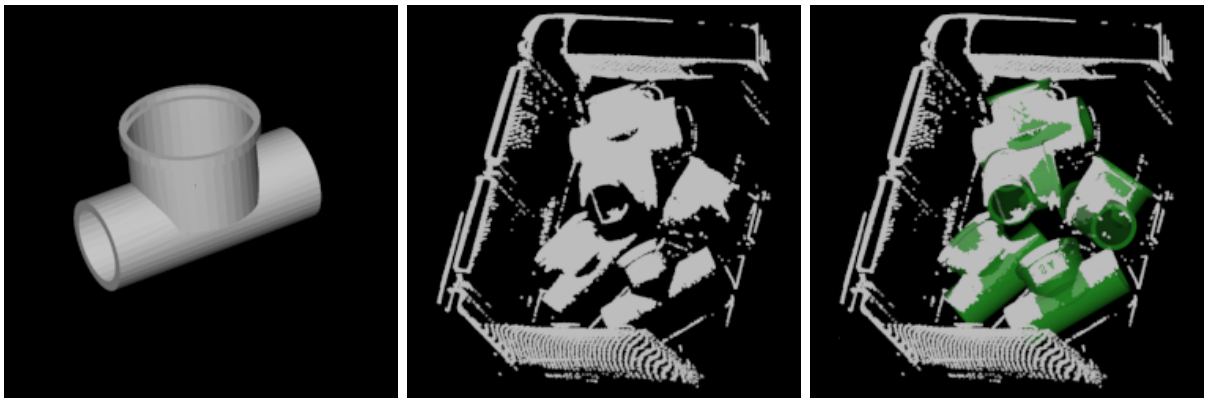
(1) (2)

(1) 3D Eingabedaten (Szene), (2) gefundene Instanz mit Greifpunkt.

Oberflächenbasiertes Matching

Der oberflächenbasierte Matching-Ansatz eignet sich auch für das Lokalisieren komplexerer Objekte. Die Form des Objektes wird dem Operator `find_surface_model` bzw. `find_surface_model_image` als Oberflächenmodell übergeben. Zurückgegeben werden daraufhin die Posen der in der Szene gefunden Instanzen.

Es ist zu beachten, dass es mehrere verschiedene Ansätze zum oberflächenbasierten Matching gibt. Für detaillierte Erklärungen bezüglich der Anwendung dieser Ansätze, Tipps, Tricks und Informationen zur Fehlersuche sei auf die Technical Note `Surface-Based Matching` verwiesen.

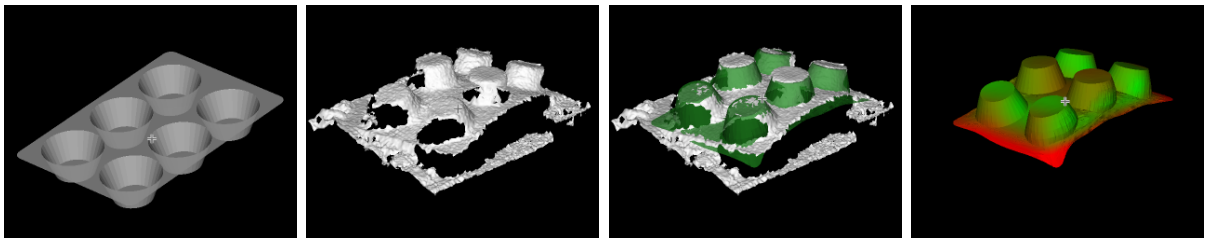


(1) (2) (3)

(1) 3D-Model für die Suche, (2) zu durchsuchende 3D Eingabedaten (Szene), (3) Ergebnis des Matchings.

Deformierbares oberflächenbasiertes Matching

Falls das gesuchte Objekt in verschieden deformierten Zuständen in der Szene auftreten kann, kann ein deformierbares Oberflächenmodell verwendet werden um diese Objekte zu lokalisieren. Wird eine Instanz eines solchen Objektes vom Operator `find_deformable_surface_model` gefunden, wird das Modell samt Pose und Deformation zurückgegeben.

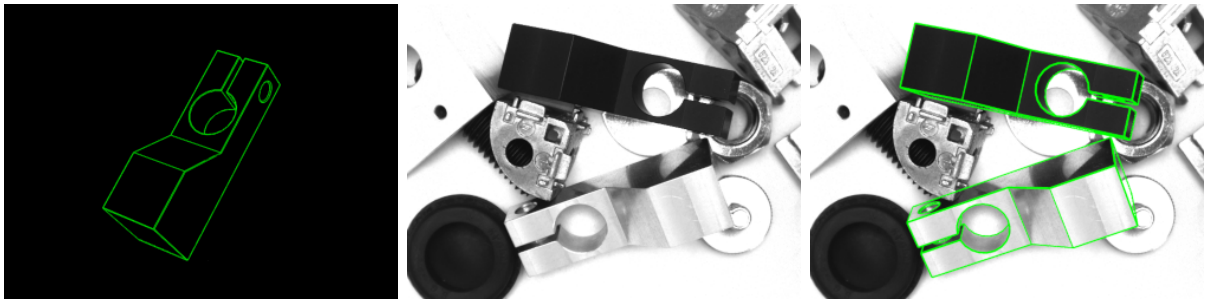


(1) (2) (3) (4)

(1) 3D-Objektmodell (2) zu durchsuchende 3D Eingabedaten (Szene), (3) Modell, transformiert in die gefundene Pose, (4) deformiertes Objektmodell.

Formbasiertes Matching

Mit formbasiertem 3D-Matching können Instanzen eines CAD-Modells in 2D-Bildern statt in 3D-Punktwolken gesucht werden. Dazu müssen die Kanten des gesuchten Objekts deutlich im Bild sichtbar sein und die verwendete Kamera muss zuvor kalibriert worden sein. Die gefundenen Posen werden vom Operator `find_shape_model_3d` als Ergebnis zurückgegeben.



(1) (2) (3)

(1) 3D-Formmodell, (2) Eingabebild, (3) gefundenes Formmodell, projiziert in das Eingabebild.

3.1 3D-Box

```
find_box_3d ( : : ObjectModel3DScene, SideLen1, SideLen2, SideLen3,
    MinScore, GenParam : GrippingPose, Score, ObjectModel3DBox,
    BoxInformation )
```

Findet Boxen in 3D-Daten.

`find_box_3d` findet Boxen im 3D-Objektmodell `ObjectModel3DScene` und gibt für jede Box die Pose eines Greifpunktes `GrippingPose`, ein 3D-Objektmodell `ObjectModel3DBox`, eine Bewertung `Score` und ein Dictionary `BoxInformation` mit weiteren Informationen zu den gefundenen Boxen zurück.

Die Kantenlängen der zu suchenden Boxen werden in `SideLen1`, `SideLen2` und `SideLen3` übergeben. Jede dieser Längen besteht aus einem Tupel mit zwei Werten, die den möglichen Wertebereich der Kantenlänge angeben. Wenn zu erwarten ist, dass nur eine Seite der Box zu sehen ist, oder die übrige Seitenlänge nicht beschränkt werden soll, kann `SideLen3` auch auf *-1* gesetzt werden.

Der Parameter `MinScore` setzt die Mindestbewertung für die zurückgegebenen Boxen fest. Boxen, die eine Bewertung unterhalb dieses Wertes haben, werden nicht zurückgegeben.

`ObjectModel3DScene` muss ein XYZ-Mapping aufweisen, was beispielsweise der Fall ist, wenn das Modell mit `xyz_to_object_model_3d` erzeugt wurde.

Üblicher Arbeitsablauf

Ein üblicher Arbeitsablauf zum Finden von 3D-Boxen in 3D-Daten sieht wie folgt aus:

1. Die 3D-Daten werden entweder als XYZ-Bilder, oder direkt als 3D-Objektmodell mit XYZ-Mapping erzeugt.
2. Es sollte so viel wie möglich vom Hintergrund und von anderen Störobjekten entfernt werden, die nicht Teil einer Box sind. Dies macht das Auffinden von Boxen robuster und schneller. Dafür kann z.B. `threshold` und `reduce_domain` auf die XYZ-Bilder angewendet werden, um im Anschluss das Modell mit `xyz_to_object_model_3d` zu erzeugen. Andere Möglichkeiten zur Verbesserung der Laufzeit finden sich weiter unten im dem Abschnitt über Problembehandlung.
3. Wenn die 3D-Daten als XYZ-Bilder vorliegen, müssen sie mittels `xyz_to_object_model_3d` in ein 3D-Objektmodell konvertiert werden.
4. Die ungefähren Seitenlängen der zu findenden Boxen müssen ermittelt werden. Zu beachten ist, dass spätere Änderungen dieser Seitenlängen möglicherweise auch die Anpassung anderer Parameter notwendig macht, etwa `MinScore`.
5. `find_box_3d` wird aufgerufen. Als Parameter wird das 3D-Objektmodell der Szene sowie die ungefähren Seitenlängen übergeben.
6. Mittels der Prozedur `visualize_object_model_3d` können, wenn notwendig, die Ergebnisse visualisiert werden.

Interpretation der Ergebnisse

Die gefundenen Boxen werden auf verschiedene Weisen zurückgegeben.

Zunächst wird für jede Box die Pose eines Greifpunktes in `GrippingPose` zurückgegeben. Die verwendete Boxseite und die z-Achse der Greifpose werden anhand des XYZ-Mapping bestimmt. Ist nur eine Seite der Box sichtbar, dann liegt das Zentrum der Greifpose in der Mitte dieser Seite und ihre z-Achse zeigt vom Aufnahmepunkt des XYZ-Mapping weg. Wenn mehrere Seiten der Box sichtbar sind, liegt die Pose des Greifpunkts in der Mitte der Seite, die am ehesten parallel zum Aufnahmepunkt des XYZ-Mapping liegt. Die z-Achse der Pose zeigt wieder vom Aufnahmepunkt des XYZ-Mapping weg. Für die x-Achse der Pose wird die Boxachse gewählt, die grob am besten mit der Spaltenrichtung des XYZ-Mapping übereinstimmt. Die y-Achse wird aus der x- und z-Achse berechnet.

Die Boxen werden außerdem in `ObjectModel3DBox` als vermaschtes 3D-Objektmodell zurückgegeben. Dies ermöglicht eine einfache Visualisierung der Ergebnisse.

Für jede gefundene Box wird in `Score` eine Bewertung zwischen *0* und *1* zurückgegeben. Diese gibt an, wie gut die Box und ihre Kanten sichtbar sind, und wie gut die gefundene Box zu den spezifizierten Kantenlängen passt.

Schlussendlich wird mit `BoxInformation` ein Dictionary zurückgegeben, welches weitere Informationen über die Ergebnisse enthält. Diese können mit `get_dict_param` und `get_dict_tuple` abgerufen werden, oder mit dem Handle Inspect-Fenster in HDevelop untersucht werden.

Das Dictionary `BoxInformation` enthält folgende Schlüssel:

results: Dieser Schlüssel verweist auf ein Dictionary, das die gefundenen Boxen enthält. Diese sind absteigend nach Bewertung sortiert und sind durch aufsteigende Integer-Werte (Startwert `0`) als Schlüssel referenziert.

Für jede gefundene Box wird ein Dictionary mit folgenden Schlüsseln zurückgegeben:

box_pose: Beschreibt die Pose der Box relativ zum Koordinatensystem der Szene. Diese Pose wird zur Visualisierung der Box verwendet.

box_length_x, box_length_y, box_length_z: Die Kantenlängen der gefundenen Box entsprechend der Pose `box_pose`. `box_length_x` und `box_length_y` enthalten immer einen positiven Wert. Wenn nur eine Seite der Box sichtbar ist wird `box_length_z` auf *0* gesetzt.

gripping_pose: Entspricht der Pose, die in `GrippingPose` zurückgegeben wird.

gripping_length_x, gripping_length_y, gripping_length_z: Die Kantenlängen der gefundenen Box entsprechend der Pose `GrippingPose`. `gripping_length_x` und `gripping_length_y` enthalten immer einen positiven Wert. Wenn nur eine Seite der Box sichtbar ist wird `gripping_length_z` auf *0* gesetzt.

score: Entspricht der Bewertung, die in `Score` zurückgegeben wird.

one_side_only: Gibt an, ob nur eine Seite der Box sichtbar ist (*'true'*) oder mehr (*'false'*).

gen_param: Dieses Dictionary enthält die Parameter, die an `find_box_3d` übergeben wurden. `SideLen1`, `SideLen2`, und `SideLen3` sind in einem Tupel mit dem Schlüssel `lengths` zusammengefasst. Der Schlüssel `min_score` verweist auf `MinScore`. Die weiteren Schlüssel sind analog zu den generischen Parametern im Dictionary `GenParam` bezeichnet (siehe unten).

sampled_edges: Dieses 3D-Objektmodell mit abgetasteten Kanten enthält die Blickrichtung der Kantenpunkte als Normalenvektoren.

sampled_edges_direction: Dieses 3D-Objektmodell mit abgetasteten Kanten (analog zu `sampled_edges`) enthält die Kantenrichtungen der Kantenpunkte als Normalenvektoren.

sampled_scene: Dieser Schlüssel enthält die abgetastete Szene, in der die Boxen gesucht werden. Sie kann zur Visualisierung oder zur Überprüfung der Abtastdistanz verwendet werden.

sampled_reference_points: Dieser Schlüssel enthält ein 3D-Objektmodell mit allen Punkten der 3D-Szene, die als Referenzpunkte für das Matching verwendet wurde. Für jeden dieser Referenzpunkte wird eine optimale Box-Pose berechnet, in der Annahme, dass sich der Punkt auf der Oberfläche der Box befindet.

Generische Parameter

Weitere generische Parameter können als Schlüssel-Tupeldaten-Paare in das Dictionary `GenParam` übergeben werden um den Matching-Vorgang zu verbessern. Die folgenden Parameternamen fungieren als Schlüssel zu den dazugehörigen Tupeldaten (siehe `create_dict` und `set_dict_tuple`).

3d_edges: Mit diesem Schlüssel können die zu verwendenden 3D-Kanten übergeben werden. Als Wert muss ein 3D-Objektmodell übergeben werden. Dieses Modell wird üblicherweise mit `edges_object_model_3d` erzeugt , und kann bei Bedarf noch weiter gefiltert werden um z.B. Ausreißer zu entfernen. Wenn dieser Parameter nicht angegeben wird, werden die Kanten intern aus der Szene ähnlich zum Operator `edges_object_model_3d` extrahiert.

3d_edge_min_amplitude: Legt die minimale Amplitude einer Unstetigkeit fest, um als Kante klassifiziert zu werden. Falls Kanten manuell in `3d_edges` übergeben werden ist zu beachten, dass dieser Parameter ignoriert wird. Andernfalls verhält sich `3d_edge_min_amplitude` ähnlich wie `MinAmplitude` für den Operator `edges_object_model_3d`.

Restriktion: `3d_edge_min_amplitude` >= 0

Standardwert: 10% der kleinsten Boxdiagonale.

max_gap: Wenn keine 3D-Kanten mittels `3d_edges` übergeben wurden, werden diese intern automatisch extrahiert. Der Parameter steuert diese Kantenextraktion.

`max_gap` hat die gleiche Bedeutung wie in `edges_object_model_3d`.

remove_outer_edges: Entfernt die am äußersten gelegenen Kanten (wenn auf *'true'* gesetzt). Für Anwendungen bei denen ein Griff in die Kiste erfolgt kann dies zum Beispiel nützlich sein, um die Kiste aus den 3D-Daten zu entfernen.

Standardwert: 'false'

Mögliche Werte: 'false', 'true'

max_num_boxes: Begrenzt die Anzahl an zurückgegebenen Boxen. Standardmäßig gibt `find_box_3d` alle gefundenen Boxen zurück, deren Bewertung über `MinScore` liegt. Mit diesem Parameter kann die Anzahl entsprechend begrenzt werden.

Standardwert: 0 (alle Boxen zurückgeben)

box_type: Legt die Arten von Boxen fest, die gesucht werden sollen. Für *'full_box_visible'* werden nur dann Boxen zurückgegeben, wenn mehr als eine Seite der Box sichtbar ist. Wird *'single_side_visible'* gesetzt, wird nur nach Boxen mit einer sichtbaren Seite gesucht. Ist trotzdem eine weitere Boxseite sichtbar wird diese nicht berücksichtigt. Mit *'all'* werden alle Boxen zurückgegeben.

Standardwert: 'all'

Mögliche Werte: 'all', 'single_side_visible', 'full_box_visible'

Problembehandlung

***Visualisierung der gefundenen Kanten und der abgetasteten Szene*:** Um die Ergebnisse der Boxerkennung besser zu verstehen, können einige der intern verwendeten Daten mittels `get_dict_tuple` aus `BoxInformation` ausgelesen werden

Die abgetastete Szene kann mit dem Schlüssel `sampled_scene` ausgelesen werden. Das Finden von kleineren Boxen braucht eine dichter abgetastete Szene, was die Boxerkennung langsamer macht.

Die abgetasteten 3D-Kanten können mit den Schlüsseln `sampled_edges` und `sampled_edges_directions`. ausgelesen werden. Die beiden 3D-Objektmodelle enthalten die gleichen Punkte, allerdings sind in `sampled_edges` die Normalenrichtungen auf die Blickrichtungen der Kanten gesetzt, während in `sampled_edges_directions` die Normalenrichtungen auf die Kantenrichtung gesetzt werden. Die Kantenrichtung steht senkrecht auf den Kanten und zeigt nach außen.

***Verbesserung der Laufzeit*:** Wenn `find_box_3d` zu lange dauert, können die folgenden Schritte helfen, die Laufzeit zu verringern.

- **Entfernen von Hintergrund und anderen Störobjekten**: Eine deutliche Verbesserung der Laufzeit und Genauigkeit kann i.d.R. durch das Entfernen von möglichst vielen Teilen der Szene erreicht werden, die nicht auf Boxen liegen.
  Übliche Ansätze zum Entfernen dieser Daten sind:
  - Schwellenwerte auf den X-, Y- und Z-Koordinaten, entweder durch `threshold` und `reduce_domain` auf den XYZ-Bildern vor dem Aufruf von `xyz_to_object_model_3d`, oder durch die Anwendung von `select_points_object_model_3d` direkt auf das 3D-Objektmodell, welches die Szene beinhaltet.
  - Einige Sensoren liefern neben den 3D-Daten auch Intensitätsbilder zurück. Filter, die auf diesen Bildern aufbauen, können ggf. ebenfalls verwendet werden um Hintergrund zu entfernen.
  - Falls die Szene zum Sensor statisch ist und sich der Hintergrund in seiner Art und Lage nicht ändert, z.B. wenn ein Sensor fest über einem Laufband verbaut ist, kann der Hintergrund einmalig in einer Szene ohne Boxen aufgenommen werden, und in späteren Szenen mittels `sub_image` und `threshold` auf den Z-Bildern entfernt werden.
- **Erhöhung der Mindestbewertung**: Ein Erhöhen der Mindestbewertung `MinScore` kann dazu führen, dass Boxen während der Erkennung schon früher verworfen werden können.
- **Vergrößern der kleinstmöglichen Box**: Je kleiner die kleinstmögliche Boxseite ist, desto höher die Laufzeit von `find_box_3d`, da die Szene dichter abgetastet werden muss. Kann z.B. eine Box nur von einer Seite gesehen werden, könnte es sinnvoll sein, `SideLen3` auf *-1* zu setzen. Weiterhin kann mit `box_type` eingestellt werden, welche Arten von Boxen gesucht werden sollen.
- **Manuelle Suche und Vorverarbeitung der Kanten**: Die 3D-Kanten können manuell mittels `edges_object_model_3d` aus der Szene extrahiert werden und über den generischen Parameter `3d_edges` an den Operator übergeben werden. Eine solche manuelle Extraktion ermöglicht ein weiteres Filtern der 3D-Kanten.

Parameter

▷ **ObjectModel3DScene** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells, welches die Szene enthält.

▷ **SideLen1** (input_control) . . . . . real-array ⇝ *real*
Länge der ersten Boxkante.

▷ **SideLen2** (input_control) . . . . . real-array ⇝ *real*
Länge der zweiten Boxkante.

▷ **SideLen3** (input_control) . . . . . real-array ⇝ *real*
Länge der dritten Boxkante.
Defaultwert: -1

▷ **MinScore** (input_control) . . . . . real ⇝ *real* / integer
Mindestbewertung der zurückgegebenen Boxen.
Defaultwert: 0.6
Restriktion: 0 <= MinScore <= 1

▷ **GenParam** (input_control) . . . . . dict ⇝ *handle*
Dictionary für generische Parameter.
Defaultwert: []

▷ **GrippingPose** (output_control) . . . . . pose(-array) ⇝ *real* / integer
Greifposen der gefundenen Boxen.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertungen der gefundenen Boxen.

▷ **ObjectModel3DBox** (output_control) . . . . . object_model_3d-array ⇝ *handle*
Gefundene Boxen als 3D-Objektmodell.

▷ **BoxInformation** (output_control) . . . . . dict ⇝ *handle*
Weitere Informationen als Dictionary.

Ergebnis

Wenn alle Parameter korrekt sind liefert `find_box_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`gen_box_object_model_3d`, `get_dict_tuple`

Alternativen

`find_surface_model`

Modul

3D Metrology

3.2 Deformierbar Oberflächenbasiert

```
add_deformable_surface_model_reference_point (
    : : DeformableSurfaceModel, ReferencePointX, ReferencePointY,
    ReferencePointZ : ReferencePointIndex )
```

Hinzufügen eines Referenzpunktes zu einem deformierbaren Oberflächenmodell.

Der Operator `add_deformable_surface_model_reference_point` fügt dem deformierbaren Oberflächenmodell `DeformableSurfaceModel` einen oder mehrere Referenzpunkte hinzu. Die 3D-Koordinaten der Referenzpunkte werden in den Parametern `ReferencePointX`, `ReferencePointY` und `ReferencePointZ` übergeben. Der Index der neuen Referenzpunkte wird in `ReferencePointIndex` zurückgegeben.

Referenzpunkte werden in Modellkoordinaten definiert, d.h. im Koordinatensystem des an `create_deformable_surface_model` übergebenen 3D-Objektmodells. Die Operatoren `find_deformable_surface_model` und `refine_deformable_surface_model` liefern die Lage aller hinzugefügten Referenzpunkte zurück, so wie diese in der Szene gefunden wurden.

Parameter

▷ **DeformableSurfaceModel** (input_control) . . . . . deformable_surface_model ⇝ *handle*
Handle des deformierbaren Oberflächenmodells.

▷ **ReferencePointX** (input_control) . . . . . real(-array) ⇝ *real* / integer
X-Koordinaten der Referenzpunkte.

▷ **ReferencePointY** (input_control) . . . . . real(-array) ⇝ *real* / integer
Y-Koordinaten der Referenzpunkte.

▷ **ReferencePointZ** (input_control) . . . . . real(-array) ⇝ *real* / integer
Z-Koordinaten der Referenzpunkte.

▷ **ReferencePointIndex** (output_control) . . . . . integer(-array) ⇝ *integer*
Indizes der hinzugefügten Referenzpunkte.

Ergebnis

Wenn alle Parameter korrekt sind liefert `add_deformable_surface_model_reference_point` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DeformableSurfaceModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_deformable_surface_model`, `read_deformable_surface_model`

Nachfolger

`find_deformable_surface_model`, `refine_deformable_surface_model`, `write_deformable_surface_model`

Siehe auch

`create_deformable_surface_model`, `find_deformable_surface_model`, `refine_deformable_surface_model`

Modul

3D Metrology

add_deformable_surface_model_sample (: : DeformableSurfaceModel, ObjectModel3D :)

Fügt einem deformierbaren Oberflächenmodell ein Deformationsbeispiel hinzu.

Der Operator `add_deformable_surface_model_sample` fügt dem deformierbaren Oberflächenmodell `DeformableSurfaceModel` die in `ObjectModel3D` übergebene Deformation als Beispiel hinzu. Das Modell ist anschließend in der Lage, Deformationen zu erkennen die dem Beispiel ähneln. Die in `ObjectModel3D` übergebene Punktwolke muss genau so viele Punkte enthalten wie das abgetastete Modell und ist üblicherweise ein Ergebnis von `find_deformable_surface_model` oder `refine_deformable_surface_model`. Das deformierbare Oberflächenmodell muss vorher bereits erstellt worden sein, z.B. mit `create_deformable_surface_model`.

Parameter

▷ **DeformableSurfaceModel** (input_control) . . . . . deformable_surface_model ⇝ *handle*
Handle des deformierbaren Oberflächenmodells.

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des deformierten 3D-Objektmodells

Ergebnis

Wenn alle Parameter korrekt sind liefert `add_deformable_surface_model_sample` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DeformableSurfaceModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_deformable_surface_model`, `find_deformable_surface_model`, `refine_deformable_surface_model`

Nachfolger

`find_deformable_surface_model`, `refine_deformable_surface_model`, `get_deformable_surface_model_param`, `write_deformable_surface_model`, `clear_deformable_surface_model`

Alternativen

`read_deformable_surface_model`

Siehe auch

`find_deformable_surface_model`, `refine_deformable_surface_model`, `read_deformable_surface_model`, `create_deformable_surface_model`, `write_deformable_surface_model`, `clear_deformable_surface_model`

Modul

3D Metrology

clear_deformable_surface_matching_result (: : DeformableSurfaceMatchingResult :)

Freigabe eines Ergebnisses des deformierbaren oberflächenbasierten Matchings.

`clear_deformable_surface_matching_result` gibt den Speicher eines Ergebnisses des deformierbaren oberflächenbasierten Matchings frei, welches mit `find_deformable_surface_model` oder `refine_deformable_surface_model` angelegt wurde. Das Ergebnis kann nach dem Aufruf nicht mehr verwendet werden. Das Handle `DeformableSurfaceMatchingResult` ist nach dem Aufruf ungültig.

Parameter

▷ **DeformableSurfaceMatchingResult** (input_control) . . . . . .
deformable_surface_matching_result(-array) ⇝ *handle*

Ergebnishandle des deformierbaren oberflächenbasierten Matchings.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `clear_deformable_surface_matching_result` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DeformableSurfaceMatchingResult

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`find_deformable_surface_model`, `refine_deformable_surface_model`

Siehe auch

`find_deformable_surface_model`, `refine_deformable_surface_model`

Modul

3D Metrology

clear_deformable_surface_model (: : DeformableSurfaceModel :)

Freigabe des Speichers eines deformierbaren Oberflächenmodells.

`clear_deformable_surface_model` gibt den Speicher eines deformierbaren Oberflächenmodells, welches z.B. mit `create_deformable_surface_model` angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `DeformableSurfaceModel` ist nach dem Aufruf ungültig.

Parameter

▷ **DeformableSurfaceModel** (input_control) . . . . . deformable_surface_model(-array) ⇝ *handle*
Handle des deformierbaren Oberflächenmodells

Ergebnis

Ist der Handle des Modells gültig, dann liefert `clear_deformable_surface_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DeformableSurfaceModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`read_deformable_surface_model`, `create_deformable_surface_model`

Siehe auch

`read_deformable_surface_model`, `create_deformable_surface_model`

Modul

3D Metrology

create_deformable_surface_model (: : ObjectModel3D, RelSamplingDistance, GenParamName, GenParamValue : DeformableSurfaceModel)

Erstellt die für deformierbares oberflächenbasiertes Matching notwendigen Datenstrukturen.

`create_deformable_surface_model` erstellt aus dem 3D-Objektmodell `ObjectModel3D` ein Modell für deformierbares oberflächenbasiertes Matching. Das 3D-Oberflächenmodell kann beispielsweise mittels

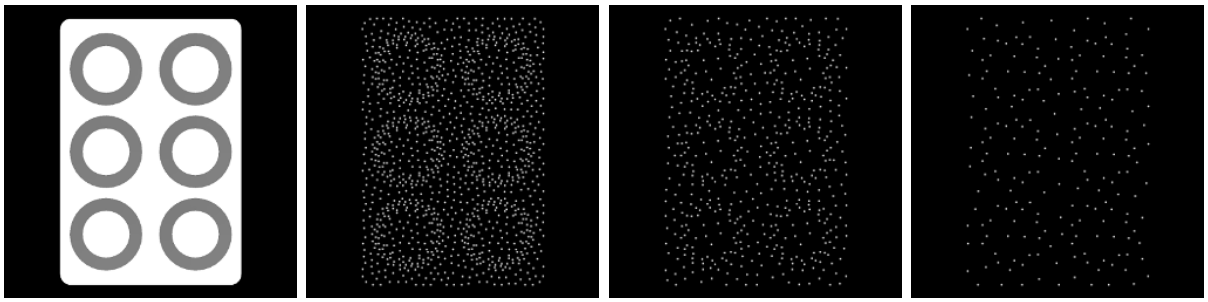
read_object_model_3d aus einer Datei gelesen worden sein oder mittels xyz_to_object_model_3d erstellt worden sein. Das Handle des erstellten Oberflächenmodells wird in DeformableSurfaceModel zurückgegeben.

Für die Erstellung des deformierbaren Oberflächenmodells aus dem 3D-Objektmodell werden Punkte und Normalen aus dem 3D-Objektmodell benötigt. Hier sind die folgenden Kombinationen möglich:

- Punkte und Punktnormalen, z.B. aus einem Aufruf von surface_normals_object_model_3d
- Punkte und ein Dreiecks- oder Polygongitter, z.B. aus einer CAD-Datei
- Punkte und ein 2D-Mapping, z.B. für 3D-Objektmodelle, die mittels xyz_to_object_model_3d erstellt wurden

Es ist zu beachten, dass die Richtung und Ausrichtung (einwärts oder auswärts) der Normalen des Modells wichtig für das Matching sind.

Für das deformierbare Oberflächenmodell wird die Oberfläche des 3D-Objektmodells mit einem bestimmten Abstand abgetastet. Die Abtastdistanz wird im Parameter RelSamplingDistance spezifiziert, und wird relativ zum Durchmesser des umschließenden achsenparallelen Quaders angegeben. Wird z.B. RelSamplingDistance auf *0.05* gesetzt und ist der Durchmesser von ObjectModel3D gleich 10 cm, dann haben die Punkte der abgetasteten Oberfläche einen Abstand von etwa 5 mm. Die abgetasteten Punkte werden für das Matching in find_deformable_surface_model verwendet. Die abgetasteten Punkte können mit dem Operator get_deformable_surface_model_param als *'sampled_model'* abgerufen werden. Es ist zu beachten, dass einzelne Ausreißerpunkte im 3D-Objektmodell den Durchmesser des umschließenden Quaders deutlich verändern können, was aufgrund der relativen Parametrisierung die Abtastdistanz stark stören kann. Ausreißerpunkte im 3D-Objektmodell sind daher vor dem Aufruf des Operators zu entfernen. Verringert man RelSamplingDistance, so werden mehr Punkte abgetastet, was zu einem stabileren aber langsameren Matching führt. Erhöht man RelSamplingDistance, so werden weniger Punkte abgetastet, was zu einem schnelleren aber weniger stabilen Matching führt.



(1) (2) (3) (4)

(1) Ursprüngliches 3D-Modell. (2) 3D-Modell abgetastet mit RelSamplingDistance = *0.02*. (3) RelSamplingDistance = *0.03*. (4) RelSamplingDistance = *0.05*.

Standardmäßig können mit create_deformable_surface_model erstellte deformierbare Oberflächenmodelle bereits eine moderate Deformation handhaben. Der Operator add_deformable_surface_model_sample kann verwendet werden um dem Modell weitere Trainingsbeispiele hinzuzufügen, was den Bereich der möglichen Deformationen erweitert. Der Deformationsbereich kann auch über die generischen Parameter *'scale_min'*, *'scale_max'* und *'bending_max'* kontrolliert werden (siehe unten).

In den Parametern GenParamName und GenParamValue können generische Parameter und deren Werte übergeben werden. GenParamName enthält ein Tupel von Parameternamen die gesetzt werden sollen, und GenParamValue die dazugehörigen Werte. Die folgenden Werte sind für GenParamName möglich:

'model_invert_normals': Invertieren der Normalenausrichtung des 3D-Objektmodells. Die Normalenausrichtungen (einwärts oder auswärts) müssen für die Erstellung des Oberflächenmodells bekannt sein und mit der Ausrichtung in der Szene übereinstimmen. Wenn sowohl das Modell als auch die Szene mit demselben Aufbau aufgenommen werden, so zeigen die Normalen bereits in die gleiche Richtung. Wenn das Modell aus einer CAD-Datei geladen wurde dann könnten die Normalen in unterschiedliche Richtungen zeigen. Wenn beim Matching der Effekt auftritt, dass das Modell auf der 'Außenseite' der Szene gefunden wird und das Modell aus einer CAD-Datei stammt, so sollte versucht werden diesen Parameter auf *'true'* zu setzen. Es

ist auch darauf zu achten, dass die Normalen in der CAD-Datei alle entweder nach innen oder nach außen zeigen, d.h. konsistent orientiert sind.

Werteliste: 'false', 'true'.

Default: 'false'.

'scale_min' **and** *'scale_max'***:** Kontrollieren des Skalierungsbereichs des Modells. Wird einer der beiden Parameter gesetzt, dann muss auch der andere Parameter gesetzt werden.

Wertevorschläge: 0.8, 1, 1.2.

Default: Keine Skalierung.

Restriktion: 0 < *'scale_min'* < *'scale_max'*

*'bending_max'***:** Kontrollieren der automatisch erlernten Deformation des Modells. Das Modell wird automatisch deformiert in dem es um einen Winkel bis zu *'bending_max'* verbogen wird. Dies erlaubt das Finden von Deformationen innerhalb dieses Biegebereichs. Der Winkel wird in Grad angegeben.

Wertevorschläge: 5, 10, 30.

Default: 20.

Restriktion: 0 <= *'bending_max'* < 90

*'stiffness'***:** Kontrollieren der Steifheit des Modells während der Lageverfeinerung. Größere Werte führen zu einem steiferen Modell das weniger stark deformiert werden kann. Kleinere Werte dieses Parameters führen zu einem weniger steifen Modell, welches mehr Deformation erlaubt.

Wertevorschläge: 0.2, 0.5, 0.8.

Default: 0.5.

Restriktion: 0 < *'stiffness'* <= 1

Parameter

▷ **`ObjectModel3D`** (input_control) ..... object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells.

▷ **`RelSamplingDistance`** (input_control) ..... real ⇝ *real*
Abtastdistanz relativ zum Durchmesser des Objekts.
Defaultwert: 0.05
Wertevorschläge: `RelSamplingDistance` ∈ {0.1, 0.05, 0.03, 0.02, 0.01}
Restriktion: 0 < RelSamplingDistance < 1

▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamName` ∈ {'model_invert_normals', 'scale_min', 'scale_max', 'bending_max', 'stiffness'}

▷ **`GenParamValue`** (input_control) ..... attribute.value(-array) ⇝ *string* / real / integer
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false', 1, 0.9, 1.1, 5, 10, 20, 30, 0.05, 0.1, 0.2}

▷ **`DeformableSurfaceModel`** (output_control) ..... deformable_surface_model ⇝ *handle*
Handle des deformierbaren Oberflächenmodells.

Ergebnis

Wenn alle Parameter korrekt sind liefert `create_deformable_surface_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d, get_object_model_3d_params

Nachfolger

add_deformable_surface_model_sample, add_deformable_surface_model_reference_point, find_deformable_surface_model, refine_deformable_surface_model, get_deformable_surface_model_param, write_deformable_surface_model, clear_deformable_surface_model

Alternativen

read_deformable_surface_model

Siehe auch

find_deformable_surface_model, refine_deformable_surface_model, read_deformable_surface_model, add_deformable_surface_model_sample, add_deformable_surface_model_reference_point, write_deformable_surface_model, clear_deformable_surface_model

Literatur

Bertram Drost, Slobodan Ilic: „Graph-Based Deformable 3D Object Matching.“ Proceedings of the 37th German Conference on Pattern Recognition, pp. 222-233, 2015.

Modul

3D Metrology

deserialize_deformable_surface_model (: : SerializedItemHandle : DeformableSurfaceModel)

Deserialisieren eines deformierbaren Oberflächenmodells.

deserialize_deformable_surface_model deserialisiert ein deformierbares Oberflächenmodell, welches mit serialize_deformable_surface_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte deformierbare Oberflächenmodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten deformierbaren Oberflächenmodell mit dem Handle DeformableSurfaceModel gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **DeformableSurfaceModel** (output_control) . . . . . deformable_surface_model ⇝ *handle*
  Handle des gelesenen deformierbaren Oberflächenmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_deformable_surface_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_deformable_surface_model

Nachfolger

find_deformable_surface_model, refine_deformable_surface_model, get_deformable_surface_model_param, clear_deformable_surface_model

Alternativen

create_deformable_surface_model

Siehe auch

`create_deformable_surface_model`, `read_deformable_surface_model`, `write_deformable_surface_model`, `serialize_deformable_surface_model`

Modul

3D Metrology

`find_deformable_surface_model` `( : : DeformableSurfaceModel, ObjectModel3D, RelSamplingDistance, MinScore, GenParamName, GenParamValue : Score, DeformableSurfaceMatchingResult )`

Suchen des besten Matches eines deformierbaren Oberflächenmodells in einer Szene.

`find_deformable_surface_model` sucht den besten Match des deformierbaren Oberflächenmodells `DeformableSurfaceModel` in der 3D-Szene `ObjectModel3D`. Das deformierbare Oberflächenmodell `DeformableSurfaceModel` muss zuvor z.B. mittels `create_deformable_surface_model` oder `read_deformable_surface_model` erstellt worden sein. Das 3D-Objektmodell der Szene kann beispielsweise mittels `read_object_model_3d` aus einer Datei gelesen worden sein oder aus Bildern mittels `xyz_to_object_model_3d` erstellt worden sein.

Zum Auffinden des Oberflächenmodells werden Punkte und Normalen der 3D-Szene verwendet. Die Szene sollte eine der folgenden Optionen zur Verfügung stellen:

- Punkte und Punktnormalen
- Punkte und ein 2D-Mapping, z.B. für 3D-Objektmodelle die mittels `xyz_to_object_model_3d` erstellt wurden

Für eine genaue Pose ist es wichtig, dass die Normalen der Szene und des Modells in die gleiche Richtung zeigen (siehe *'scene_invert_normals'*). Wird eine vermaschte Szene mit Dreiecken oder Polygonen übergeben, z.B. ein CAD-Modell, so werden die Flächen ignoriert und nur die Punkte der Szene verwendet. In der Regel sollte daher nicht in einer vermaschten Szene gesucht werden. Stattdessen sollte eine solche Szene zuvor beispielsweise mit `sample_object_model_3d` in Punkte mit Normalen umgewandelt werden. Werden verrauschte Daten verwendet, z.B. von Time-of-Flight-Kameras, sollte der generische Parameter *'scene_normal_computation'* auf *'mls'* gesetzt werden um robustere Ergebnisse zu erhalten (s.u.).

Die 3D-Szene `ObjectModel3D` wird zunächst gleichmäßig abgetastet. Der Parameter `RelSamplingDistance` gibt die Abtastdistanz relativ zum Durchmesser des Oberflächenmodells an. Verkleinert man `RelSamplingDistance`, so werden mehr Punkte der Szene abgetastet, was zu einem stabileren aber langsameren Matching führt. Erhöht man `RelSamplingDistance`, so werden weniger Punkte abgetastet, was zu einem weniger stabilen aber schnelleren Matching führt. Eine Illustration zur `RelSamplingDistance` befindet sich bei dem Operator `create_deformable_surface_model`.

Die abgetasteten Punkte können mit dem Operator `get_deformable_surface_matching_result` abgerufen werden. Für ein robustes Matching sollten mindestens 50-100 Punkte auf jeder zu findenden Objektinstanz liegen.

Zunächst wird die ungefähre Lage des Objektes gesucht. Diese wird anschließend verfeinert, d.h. genauer an die Szene angepasst. Die generischen Parameter zum Steuern der Verfeinerung sind weiter unten beschrieben.

Wurde eine Objektinstanz gefunden, so wird in `Score` eine Bewertung des Ergebnisses zurückgeliefert und im Ergebnishandle `DeformableSurfaceMatchingResult` dessen Lage und Deformation. Die Details des Ergebnisses, z.B. das deformierte Modell oder die Lage von Referenzpunkten, können über den Operator `get_deformable_surface_matching_result` abgerufen werden.

Die Bewertung ist zwischen *0* und *1* normalisiert und repräsentiert welcher Anteil der Oberfläche des Modells in der Szene gefunden wurde. Ein Wert von *1* bedeutet dabei ein perfektes Ergebnis, d.h. das gesamte Modell wurde gefunden. Der Parameter `MinScore` kann verwendet werden um das Ergebnis zu filtern. Ein Match wird nur zurückgegeben, wenn sein Score größer als `MinScore` ist.

In den Parametern `GenParamName` und `GenParamValue` können generische Parameter und deren Werte übergeben werden. `GenParamName` enthält ein Tupel von Parameternamen die gesetzt werden sollen, und `GenParamValue` die dazugehörigen Werte. Die folgenden Werte sind für `GenParamName` möglich:

'scene_normal_computation': Dieser Parameter steuert die Normalenberechnung der abgetasteten 3D-Szene. Im Standard-Modus *'fast'* werden die Normalen anhand einer relativ kleinen Nachbarschaft berechnen. Im Modus *'mls'* wird eine größere Nachbarschaft einbezogen und das komplexere aber genauere *'mls'*-Verfahren für die Normalenberechnung verwendet. Eine genauere Beschreibung des *'mls'*-Verfahrens findet sich beim Operator `surface_normals_object_model_3d`. Der *'mls'*-Modus ist vor allem für Daten mit hohem Rauschen gedacht, z.B. für Bilder aus Time-of-Flight-Kameras.

Mögliche Werte: 'fast', *'mls'*

Standardwert: 'fast'

'scene_invert_normals': Dieser Parameter invertiert die Normalenausrichtung der 3D-Szene. Die Normalenausrichtungen (einwärts oder auswärts) der Szene müssen mit den Ausrichtungen des 3D-Objektmodells übereinstimmen. Wenn sowohl das Modell als auch die Szene mit demselben Aufbau aufgenommen werden, so zeigen die Normalen bereits in die gleiche Richtung. Wenn beim Matching der Effekt auftritt, dass das Modell auf der 'Außenseite' der Szene gefunden wird, so sollte versucht werden, diesen Parameter auf *'true'* zu setzen. Es ist auch darauf zu achten, dass die Normalen in der Szene alle entweder nach innen oder nach außen zeigen, d.h. konsistent orientiert sind.

Mögliche Werte: 'false', *'true'*

Standardwert: 'false'

'pose_ref_num_steps': Anzahl der Iterationen der deformierbaren Verfeinerung. Erhöht man diesen Wert, so wird die Verfeinerung genauer aber langsamer. Verringert man den Wert, so wird sie entsprechend ungenauer und schneller. Ab einer gewissen Anzahl an Iterationen konvergiert das Verfahren, so dass eine weitere Erhöhung keinen Effekt mehr auf die Genauigkeit hat.

Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist.

Vorgeschlagene Werte: 1, *10*, *25*, *50*

Standardwert: 25

Restriktion: 'pose_ref_num_steps' > 0

'pose_ref_dist_threshold_rel': Setzt den Abstandsschwellenwert für die deformierbare Verfeinerung relativ zum Durchmesser des Modells. Nur Punkte deren Abstand zur Objektinstanz diesen Schwellenwert unterschreiten gehen in die Verfeinerung ein. Punkte, die weiter von der Instanz entfernt sind, werden ignoriert. Nur einer der beiden Parameter *'pose_ref_dist_threshold_rel'* und *'pose_ref_dist_threshold_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.

Vorgeschlagene Werte: 0.05, *0.1*, *0.25*, *0.3*

Standardwert: 0.25

Restriktion: 0 < *'pose_ref_dist_threshold_rel'*

'pose_ref_dist_threshold_abs': Setzt den Abstandsschwellenwert für die deformierbare Verfeinerung als absoluten Wert. Für Details siehe den Parameter *'pose_ref_dist_threshold_rel'*. Nur einer der beiden Parameter *'pose_ref_dist_threshold_rel'* und *'pose_ref_dist_threshold_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.

Restriktion: 0 < *'pose_ref_dist_threshold_abs'*

'pose_ref_scoring_dist_rel': Setzt den Abstandsschwellenwert für die Score-Berechnung relativ zum Durchmesser des Modells. Für Details siehe den Parameter *'pose_ref_scoring_dist_abs'*. Nur einer der beiden Parameter *'pose_ref_scoring_dist_rel'* und *'pose_ref_scoring_dist_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.

Vorgeschlagene Werte: 0.1, *0.05*, *0.03*, *0.005*

Standardwert: 0.03

Restriktion: 0 < *'pose_ref_scoring_dist_rel'*

'pose_ref_scoring_dist_abs': Setzt den Abstandsschwellenwert für die Score-Berechnung als absoluten Wert. Nur Szenepunkte deren Abstand zur gefundenen Objektinstanz diesen Schwellenwert unterschreiten gehen in die Berechnung der Bewertung ein und werden als 'auf dem Objekt liegend' klassifiziert. Alle anderen Szenepunkte werden ignoriert. Der Schwellenwert sollte entsprechend dem zu erwartenden Rauschen und der Genauigkeit der Pose-Verfeinerung gesetzt werden. Nur einer der beiden Parameter *'pose_ref_scoring_dist_rel'* und *'pose_ref_scoring_dist_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.

Restriktion: 0 < *'pose_ref_scoring_dist_abs'*

Parameter

▷ **DeformableSurfaceModel** (input_control) . . . . . deformable_surface_model ⇝ *handle*
Handle des deformierbaren Oberflächenmodells.

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells, welches die Szene enthält.

▷ **RelSamplingDistance** (input_control) . . . . . real ⇝ *real*
Abtastdistanz der Szene relativ zum Durchmesser des Oberflächenmodells.
Defaultwert: 0.05
Wertevorschläge: `RelSamplingDistance` ∈ {0.1, 0.07, 0.05, 0.04, 0.03}
Restriktion: 0 < RelSamplingDistance < 1

▷ **MinScore** (input_control) . . . . . real ⇝ *real* / integer
Mindestbewertung des zurückgegebenen Matches.
Defaultwert: 0
Restriktion: MinScore >= 0

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'scene_normal_computation', 'scene_invert_normals', 'pose_ref_num_steps', 'pose_ref_dist_threshold_rel', 'pose_ref_dist_threshold_abs', 'pose_ref_scoring_dist_abs', 'pose_ref_scoring_dist_rel'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *string* / real / integer
Werte der generischen Parameter
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'fast', 'mls', 0, 1, 10, 25, 50, 0.05, 0.1, 0.25, 0.3, 0.05, 0.03, 0.005}

▷ **Score** (output_control) . . . . . real(-array) ⇝ *real*
Bewertung der gefundenen Objektinstanz.

▷ **DeformableSurfaceMatchingResult** (output_control) . . . . . deformable_surface_matching_result(-array) ⇝ *handle*
Ergebnishandle des Matchings.

Ergebnis

Wenn alle Parameter korrekt sind liefert `find_deformable_surface_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`, `get_object_model_3d_params`, `read_deformable_surface_model`, `create_deformable_surface_model`, `get_deformable_surface_model_param`, `add_deformable_surface_model_reference_point`, `add_deformable_surface_model_sample`

Nachfolger

`refine_deformable_surface_model`, `get_deformable_surface_matching_result`, `clear_deformable_surface_matching_result`, `clear_object_model_3d`

Alternativen

`refine_deformable_surface_model`

Siehe auch

`refine_deformable_surface_model`

Modul

3D Metrology

get_deformable_surface_matching_result (
: : DeformableSurfaceMatchingResult, ResultName,
ResultIndex : ResultValue)

Auslesen von Ergebnissen des deformierbaren oberflächenbasierten Matchings.

Der Operator get_deformable_surface_matching_result liefert detaillierte Ergebnisse des deformierbaren oberflächenbasierten Matchings oder der deformierbaren oberflächenbasierten Lageverfeinerung. Die Ergebnisse sind in DeformableSurfaceMatchingResult gespeichert, das mit find_deformable_surface_model oder refine_deformable_surface_model angelegt wurde.

Der Parameter ResultName wählt das abzurufende Detailergebnis aus. Bei einigen Details kann mittels ResultIndex der Index des Details gesetzt werden, wobei die Zählung bei *0* beginnt. Für einige Werte von ResultName wird ResultIndex ignoriert.

Die folgenden Werte sind für ResultName möglich:

'sampled_scene': Es wird ein 3D-Objektmodell zurückgegeben, welches die für das Matching oder die Verfeinerung abgetasteten Szenepunkte enthält. Dies ist hilfreich um die Abtastdistanz einzustellen (siehe Parameter RelSamplingDistance von find_deformable_surface_model und refine_deformable_surface_model.). Der Parameter ResultIndex wird ignoriert.

'rigid_pose': Wenn DeformableSurfaceMatchingResult mittels find_deformable_surface_model erzeugt wurde, wird die rigide Pose des Modells zurückgeliefert welche das deformierte Ergebnis approximiert. Der Parameter ResultIndex wird ignoriert. Dieser Parameter ist nicht verfügbar, wenn DeformableSurfaceMatchingResult mittels refine_deformable_surface_model erzeugt wurde.

'reference_point_x':

'reference_point_y':

'reference_point_z': Liefert die X-, Y- oder Z-Koordinaten von transformierten Referenzpunkten zurück. Die Referenzpunkte müssen vorher mittels add_deformable_surface_model_reference_point hinzugefügt worden sein. Die Indices der Referenzpunkte werden in ResultIndex übergeben. Wird *'all'* in ResultIndex übergeben, so wird die Lage aller gesetzten Referenzpunkte zurückgegeben.

'deformed_model': Liefert eine deformierte Variante des ursprünglich an create_deformable_surface_model übergebenen 3D-Objektmodells zurück. Das ursprüngliche Modell wird dazu entsprechend der gefundenen Deformation verformt. Im Modell enthaltene Dreiecke und Polygone sowie erweiterte Attribute bleiben dabei erhalten. Der Parameter ResultIndex wird ignoriert.

'deformed_sampled_model': Liefert ein 3D-Objektmodell zurück welches eine deformierte Variante des abgetasteten Modells enthält. Das zurückgegebene 3D-Objektmodell hat die gleich Anzahl an Punkten wie das abgetastete deformierbare Oberflächenmodell, und in der gleichen Reihenfolge. Details zur Abtastung sind in create_deformable_surface_model beschrieben. Das ursprüngliche, nicht-deformierte abgetastete Modell kann mittels get_deformable_surface_model_param abgerufen werden. Der Parameter ResultIndex wird ignoriert.

Parameter

▷ **DeformableSurfaceMatchingResult** (input_control) ..... deformable_surface_matching_result ⇝ *handle*

Ergebnishandle des deformierbaren oberflächenbasierten Matchings.

▷ **ResultName** (input_control) ..... string(-array) ⇝ *string*
Name des Ergebnisdetails.
Defaultwert: 'sampled_scene'
Werteliste: ResultName ∈ {'sampled_scene', 'rigid_pose', 'reference_point_x', 'reference_point_y', 'reference_point_z', 'deformed_model', 'deformed_sampled_model'}

▷ **ResultIndex** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Optionaler Index des Ergebnisses, beginnend bei 0.
Defaultwert: 0
Wertevorschläge: ResultIndex ∈ {0, 1, 2, 3, 'all'}
Restriktion: ResultIndex >= 0

▷ **ResultValue** (output_control) . . . . . integer(-array) ⇝ *integer* / string / real / handle
Wert des Ergebnisses.

Ergebnis

Ist der Handle des Modells gültig, dann liefert get_deformable_surface_matching_result den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

find_deformable_surface_model, refine_deformable_surface_model

Nachfolger

clear_deformable_surface_model

Siehe auch

find_deformable_surface_model, refine_deformable_surface_model, read_deformable_surface_model, write_deformable_surface_model, clear_deformable_surface_model

Modul

3D Metrology

get_deformable_surface_model_param (: : DeformableSurfaceModel, GenParamName : GenParamValue)

Auslesen von Einstellungen und Eigenschaften eines deformierbaren Oberflächenmodells.

Der Operator get_deformable_surface_model_param liefert Parameter und Eigenschaften des deformierbaren Oberflächenmodells DeformableSurfaceModel zurück. Das deformierbare Oberflächenmodell muss vorher z.B. mit create_deformable_surface_model erstellt worden sein.

Die folgenden Werte sind für GenParamName möglich:

'diameter': Durchmesser der Modellpunktwolke. Der Durchmesser ist die Länge der Diagonale des kleinsten umschließenden achsenparallelen Quaders.

'sampled_model': Die für das Matching abgetasteten 3D-Punkte des Modells. Zurückgegeben wird das Handle eines 3D-Objektmodells, welches alle Punkte enthält, die für das Matching abgetastet wurden (siehe create_deformable_surface_model).

'training_models': Liste aller trainierten 3D-Objektmodelle. Zurückgegeben wird eine Liste von 3D-Objektmodellhandles welche mittels add_deformable_surface_model_sample trainiert wurden, inklusive des ursprünglichen Modells.

'reference_points_x':

'reference_points_y':

'reference_points_z': Liefert die X-, Y- und Z-Koordinaten aller Referenzpunkte zurück, die dem deformierbaren Oberflächenmodell mittels add_deformable_surface_model_reference_point hinzugefügt wurden.

Parameter

▷ **DeformableSurfaceModel** (input_control) . . . . . deformable_surface_model ⇝ *handle*
Handle des deformierbaren Oberflächenmodells.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name der abzurufenden Eigenschaft.
Defaultwert: 'sampled_model'
Werteliste: `GenParamName` ∈ {'diameter', 'sampled_model', 'sampled_pose_refinement', 'training_models', 'reference_points_x', 'reference_points_y', 'reference_points_z', 'original_model'}

▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *real* / string / integer
Wert der abzurufenden Eigenschaft.

Ergebnis

Wenn alle Parameter korrekt sind liefert `get_deformable_surface_model_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_deformable_surface_model`, `read_deformable_surface_model`, `add_deformable_surface_model_reference_point`

Nachfolger

`find_deformable_surface_model`, `refine_deformable_surface_model`, `write_deformable_surface_model`

Siehe auch

`create_deformable_surface_model`

Modul

3D Metrology

read_deformable_surface_model (: : FileName : DeformableSurfaceModel)

Liest ein deformierbares Oberflächenmodell aus einer Datei.

`read_deformable_surface_model` liest ein deformierbares Oberflächenmodell welches zuvor mittels `write_deformable_surface_model` geschrieben wurde aus der Datei `FileName` ein. Das Handle des eingelesenen deformierbaren Oberflächenmodells wird in `DeformableSurfaceModel` zurückgegeben. Enthält `FileName` keine absolute Pfadangabe, so wird die Datei im aktuellen Verzeichnis des HALCON-Prozesses gesucht. Die Standard-Dateiendung für deformierbare Oberflächenmodelle in HALCON ist 'dsfm'. Existiert die Datei `FileName` nicht, so wird diese Standard-Dateiendung angehängt und erneut gesucht.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname der DSFM-Datei.
Dateiendung: `.dsfm`

▷ **DeformableSurfaceModel** (output_control) . . . . . deformable_surface_model ⇝ *handle*
Handle des gelesenen deformierbaren Oberflächenmodells.

Ergebnis

Wenn alle Parameter korrekt sind und die Datei gelesen werden konnte liefert `read_deformable_surface_model` den Wert TRUE. Wenn die Datei kein Oberflächenmodell enthält wird der Fehler 9506 zurückgegeben. Enthält die Datei ein Oberflächenmodell in einer Version, die nicht mit dieser HALCON-Version gelesen werden kann, wird der Fehler 9507 zurückgegeben. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`write_deformable_surface_model`

Nachfolger

`find_deformable_surface_model`, `refine_deformable_surface_model`, `get_deformable_surface_model_param`, `clear_deformable_surface_model`

Alternativen

`create_deformable_surface_model`

Siehe auch

`create_deformable_surface_model`, `write_deformable_surface_model`

Modul

3D Metrology

refine_deformable_surface_model (: : DeformableSurfaceModel, ObjectModel3D, RelSamplingDistance, InitialDeformationObjectModel3D, GenParamName, GenParamValue : Score, DeformableSurfaceMatchingResult)

Verfeinern der Lage und Deformation eines deformierbaren Oberflächenmodells in einer 3D-Szene.

`refine_deformable_surface_model` verfeinert die in `InitialDeformationObjectModel3D` übergebene näherungsweise Lage und Deformation des deformierbaren Oberflächenmodells `DeformableSurfaceModel` in der 3D-Szene `ObjectModel3D`. Das deformierbare Oberflächenmodell `DeformableSurfaceModel` muss zuvor z.B. mittels `create_deformable_surface_model` erstellt worden sein. Das 3D-Objektmodell der Szene kann beispielsweise mittels `read_object_model_3d` aus einer Datei gelesen worden sein oder mittels `xyz_to_object_model_3d` erstellt worden sein.

`refine_deformable_surface_model` ist nützlich wenn die Lage und Deformation eines 3D-Objekts in einer 3D-Szene ungefähr bekannt ist und lediglich verfeinert werden soll.

`InitialDeformationObjectModel3D` muss genau so viele Punkte und in der gleichen Reihenfolge enthalten wie das abgetastete Modell, welches mit `get_deformable_surface_model_param` abgerufen werden kann.

In `Score` wird eine Bewertung des Ergebnisses zurückgeliefert und im Ergebnishandle `DeformableSurfaceMatchingResult` die Lage und Deformation. Die Details des Ergebnisses, z.B. das deformierte Modell oder die Lage von Referenzpunkten, können über den Operator `get_deformable_surface_matching_result` abgerufen werden.

Die Bewertung ist zwischen *0* und *1* normalisiert und repräsentiert welcher Anteil der Oberfläche des Modells in der Szene gefunden wurde. Ein Wert von *1* bedeutet dabei ein perfektes Ergebnis, d.h. das gesamte Modell wurde gefunden.

In den Parametern `GenParamName` und `GenParamValue` können generische Parameter und deren Werte übergeben werden. Eine genauere Beschreibung der generischen sowie der übrigen Parameter findet sich in der Dokumentation zu `find_deformable_surface_model`.

Parameter

▷ **DeformableSurfaceModel** (input_control) . . . . . deformable_surface_model ⇝ *handle*
Handle des deformierbaren Oberflächenmodells.

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells, welches die Szene enthält.

▷ **RelSamplingDistance** (input_control) . . . . . real ⇝ *real*
Abtastdistanz der Szene relativ zum Durchmesser des Oberflächenmodells.
Defaultwert: 0.05
Wertevorschläge: RelSamplingDistance ∈ {0.1, 0.07, 0.05, 0.04, 0.03}
Restriktion: 0 < RelSamplingDistance < 1

▷ **InitialDeformationObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
Initiale Lage und Deformation des Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'scene_normal_computation', 'pose_ref_num_steps', 'pose_ref_dist_threshold_rel', 'pose_ref_dist_threshold_abs', 'pose_ref_scoring_dist_abs', 'pose_ref_scoring_dist_rel'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / real / integer
Werte der generischen Parameter
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'fast', 'mls', 0, 1, 10, 25, 50, 0.05, 0.1, 0.25, 0.3, 0.05, 0.03, 0.005}

▷ **Score** (output_control) . . . . . real(-array) ⇝ *real*
Bewertung der verfeinerten Lage und Deformation.

▷ **DeformableSurfaceMatchingResult** (output_control) . . . . . deformable_surface_matching_result(-array) ⇝ *handle*
Ergebnishandle der Verfeinerung.

Ergebnis

Wenn alle Parameter korrekt sind liefert refine_deformable_surface_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d, get_object_model_3d_params, read_deformable_surface_model, create_deformable_surface_model, get_deformable_surface_model_param, find_deformable_surface_model

Nachfolger

get_deformable_surface_matching_result, clear_deformable_surface_matching_result, clear_object_model_3d

Alternativen

find_deformable_surface_model

Siehe auch

create_deformable_surface_model, find_deformable_surface_model

Modul

3D Metrology

serialize_deformable_surface_model (
: : DeformableSurfaceModel : SerializedItemHandle)

Serialisieren eines deformierbaren Oberflächenmodells.

serialize_deformable_surface_model serialisiert die Daten eines deformierbaren Oberflächenmodells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der

Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_deformable_surface_model` in eine Datei geschrieben werden. Das deformierbare Oberflächenmodell wird in dem Handle `DeformableSurfaceModel` übergeben. Das serialisierte deformierbare Oberflächenmodell wird in dem Handle `SerializedItemHandle` zurückgegeben. und kann mit `deserialize_deformable_surface_model` wieder deserialisiert werden.

Parameter

▷ **`DeformableSurfaceModel`** (input_control) . . . . . deformable_surface_model ⇝ *handle*
Handle des deformierbaren Oberflächenmodells.

▷ **`SerializedItemHandle`** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_deformable_surface_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_deformable_surface_model`, `create_deformable_surface_model`

Nachfolger

`clear_deformable_surface_model`, `fwrite_serialized_item`, `send_serialized_item`, `deserialize_deformable_surface_model`

Siehe auch

`create_deformable_surface_model`, `read_deformable_surface_model`, `write_deformable_surface_model`, `deserialize_deformable_surface_model`

Modul

3D Metrology

`write_deformable_surface_model` `( : : DeformableSurfaceModel, FileName : )`

Schreibt ein deformierbares Oberflächenmodell in eine Datei.

`write_deformable_surface_model` schreibt das deformierbare Oberflächenmodell `DeformableSurfaceModel` in die Datei `FileName`. Die Datei kann mit `read_deformable_surface_model` wieder gelesen werden. Die Standard-Dateiendung für deformierbare Oberflächenmodelle in HALCON ist 'dsfm'.

Parameter

▷ **`DeformableSurfaceModel`** (input_control) . . . . . deformable_surface_model ⇝ *handle*
Handle des deformierbaren Oberflächenmodells.

▷ **`FileName`** (input_control) . . . . . filename.write ⇝ *string*
Dateiname.
Dateiendung: `.dsfm`

Ergebnis

Wenn alle Parameter korrekt sind und die Datei geschrieben werden konnte liefert `write_deformable_surface_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_deformable_surface_model`, `create_deformable_surface_model`, `get_deformable_surface_model_param`

Nachfolger

`clear_deformable_surface_model`

Siehe auch

`create_deformable_surface_model`, `read_deformable_surface_model`

Modul

3D Metrology

3.3 Formbasiert

clear_shape_model_3d (: : ShapeModel3DID :)

Freigabe des Speichers eines 3D-Formmodells.

`clear_shape_model_3d` gibt den Speicher eines 3D-Formmodells, das mit `create_shape_model_3d` angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `ShapeModel3DID` ist nach dem Aufruf ungültig.

Parameter

▷ **ShapeModel3DID** (input_control) . . . . . shape_model_3d(-array) ⇝ *handle*
Handle des 3D-Formmodells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `clear_shape_model_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ShapeModel3DID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_shape_model_3d`, `read_shape_model_3d`, `write_shape_model_3d`

Modul

3D Metrology

create_cam_pose_look_at_point (: : CamPosX, CamPosY, CamPosZ, LookAtX, LookAtY, LookAtZ, RefPlaneNormal, CamRoll : CamPose)

Erstellen einer 3D-Kamera-Pose aus Kamerazentrum und Blickrichtung.

Der Operator `create_cam_pose_look_at_point` erstellt eine 3D-Kamera-Pose bezüglich eines Weltkoordinatensystems basierend auf zwei Punkten und dem Kamerarollwinkel.

Der erste der beiden Punkte legt die Position des optischen Zentrums der Kamera, d.h. den Ursprung des Kamerakoordinatensystems fest. Seine drei Koordinaten werden in `CamPosX`, `CamPosY` und `CamPosZ` übergeben. Der zweite Punkt legt die Blickrichtung der Kamera fest. Er wird ebenfalls durch seine drei Koordinaten `LookAtX`, `LookAtY` und `LookAtZ` beschrieben. Folglich liegt der zweite Punkt auf der z-Achse des Kamerakoordinatensystems.

Der verbleibende festzulegende Freiheitsgrad ist eine Rotation der Kamera um ihre z-Achse, die auch als Kamerarollwinkel bezeichnet wird. Um diese Rotation zu bestimmen, kann die Normale einer Referenzebene in `RefPlaneNormal` übergeben werden, die die Referenzorientierung der Kamera festlegt. Bezüglich dieser Referenzorientierung kann schließlich der Kamerarollwinkel in `CamRoll` angegeben werden. Der Kamerarollwinkel beschreibt die Rotation der Kamera um ihre z-Achse relativ zur Referenzorientierung.

Die Referenzebene kann als Ebene im Weltkoordinatensystem interpretiert werden, die parallel zur x-Achse der Kamera (in der Referenzorientierung, d.h. mit Kamerarollwinkel 0) liegt. In einer alternativen Interpretation legt der Normalenvektor der Referenzebene projiziert in die Bildebene die Richtung fest, die in der Kamera oben ist. Der in die Bildebene projizierte Normalenvektor zeigt somit in Richtung der negativen y-Achse des Kamerakoordinatensystems. Der Parameter `RefPlaneNormal` kann einen der folgenden Werte annehmen:

'x': Die Referenzebene ist die yz-Ebene des Weltkoordinatensystems. Die projizierte x-Achse des Weltkoordinatensystems zeigt in der Bildebene nach *oben*.

'-x': Die Referenzebene ist die yz-Ebene des Weltkoordinatensystems. Die projizierte x-Achse des Weltkoordinatensystems zeigt in der Bildebene nach *unten*.

'y': Die Referenzebene ist die xz-Ebene des Weltkoordinatensystems. Die projizierte y-Achse des Weltkoordinatensystems zeigt in der Bildebene nach *oben*.

'-y': Die Referenzebene ist die xz-Ebene des Weltkoordinatensystems. Die projizierte y-Achse des Weltkoordinatensystems zeigt in der Bildebene nach *unten*.

'z': Die Referenzebene ist die xy-Ebene des Weltkoordinatensystems. Die projizierte z-Achse des Weltkoordinatensystems zeigt in der Bildebene nach *oben*.

'-z': Die Referenzebene ist die xy-Ebene des Weltkoordinatensystems. Die projizierte z-Achse des Weltkoordinatensystems zeigt in der Bildebene nach *unten*.

Alternativ zu den obigen Werten kann in `RefPlaneNormal` auch ein beliebiger Normalenvektor angegeben werden, der nicht mit einer Koordinatenachse übereinstimmt. Dazu kann ein Tupel mit den drei Komponenten des Normalenvektors übergeben werden.

Es ist zu beachten, dass sich die Position des optischen Zentrums der Kamera und der Punkt, der die Blickrichtung der Kamera festlegt, unterscheiden müssen. Außerdem darf der Normalenvektor der Referenzebene nicht parallel zur z-Achse der Kamera sein. Andernfalls ist die Kamera-Pose nicht eindeutig definiert.

`create_cam_pose_look_at_point` ist vor allem für die Visualisierung eines 3D-Objektmodells oder eines 3D-Formmodells von einer bestimmten Kameraposition aus hilfreich. In diesem Fall kann die mit `create_cam_pose_look_at_point` generierte Pose an `project_object_model_3d` bzw. `project_shape_model_3d` übergeben werden.

Es ist auch möglich, für unterschiedliche Parameter Tupel unterschiedlicher Länge zu übergeben. In diesem Fall wird intern zunächst die maximale Anzahl an übergebenen Werten über alle Eingabeparameter ermittelt. Dieser Wert bestimmt die Anzahl der zu berechnenden Kamera-Poses. Jeder Eingabeparameter kann dann einen einzelnen Wert oder genauso viele Werte wie Kamera-Poses enthalten. Im ersten Fall wird der Wert für die Berechnen aller Kamera-Poses verwendet, im zweiten Fall wird der jeweilige Wert im Eingabeparameter für die Berechnung der entsprechenden Kamera-Pose verwendet.

Parameter

▷ **CamPosX** (input_control) . . . . . real(-array) ⇝ *real*
X-Koordinate des optischen Zentrums der Kamera.

▷ **CamPosY** (input_control) . . . . . real(-array) ⇝ *real*
Y-Koordinate des optischen Zentrums der Kamera.

▷ **CamPosZ** (input_control) . . . . . real(-array) ⇝ *real*
Z-Koordinate des optischen Zentrums der Kamera.

▷ **LookAtX** (input_control) . . . . . real(-array) ⇝ *real*
X-Koordinate des 3D-Punktes, in dessen Richtung die Kamera gerichtet ist.

▷ **LookAtY** (input_control) . . . . . real(-array) ⇝ *real*
Y-Koordinate des 3D-Punktes, in dessen Richtung die Kamera gerichtet ist.

▷ **LookAtZ** (input_control) . . . . . real(-array) ⇝ *real*
Z-Koordinate des 3D-Punktes, in dessen Richtung die Kamera gerichtet ist.

▷ **RefPlaneNormal** (input_control) . . . . . string-array ⇝ *string* / real
Normalenvektor der Bezugsebene (zeigt nach oben).
Defaultwert: '-y'
Werteliste: `RefPlaneNormal` ∈ {'x', 'y', 'z', '-x', '-y', '-z'}

▷ **CamRoll** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Kamerarollwinkel.
Defaultwert: 0

▷ **CamPose** (output_control) . . . . . pose(-array) ⇝ *real* / integer
3D-Lage der Kamera.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_cam_pose_look_at_point` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn die Parameter so gewählt worden sind, dass die 3D-Lage der Kamera nicht eindeutig definiert ist, wird die Fehlermeldung 8940 zurückgegeben.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`convert_point_3d_spher_to_cart`

Alternativen

`create_pose`

Modul

3D Metrology

create_shape_model_3d (: : ObjectModel3D, CamParam, RefRotX, RefRotY, RefRotZ, OrderOfRotation, LongitudeMin, LongitudeMax, LatitudeMin, LatitudeMax, CamRollMin, CamRollMax, DistMin, DistMax, MinContrast, GenParamName, GenParamValue : ShapeModel3DID)

Vorbereiten eines 3D-Objektmodells für das Matching.

`create_shape_model_3d` bereitet ein 3D-Objektmodell, das in `ObjectModel3D` übergeben wird, als 3D-Formmodell für das Matching vor. Das 3D-Objektmodell muss zuvor mit `read_object_model_3d` eingelesen worden sein.

Das erstellte 3D-Formmodell umfasst unterschiedliche Ansichten des 3D-Objektmodells. Die Ansichten werden innerhalb des vom Benutzer vorgegebenen Pose-Bereichs generiert. Die Ansichtengenerierung erfolgt durch das automatische Platzieren virtueller Kameras um das 3D-Objektmodell und das Projizieren des 3D-Objektmodells in die Bildebene jeder virtuellen Kamera. Für jede auf diese Weise erstellte Ansicht wird dann eine 2D-Formbeschreibung erzeugt. Für die Erstellung des 3D-Formmodells werden also keine Bilder des Objektes benötigt sondern lediglich das 3D-Objektmodell, das in `ObjectModel3D` übergeben wird. Die Formbeschreibungen aller Ansichten werden im 3D-Formmmodell gespeichert, das in `ShapeModel3DID` zurückgeliefert wird. Während des Matching-Prozesses mit `find_shape_model_3d` werden die Formbeschreibungen dazu genutzt, die am besten passende Ansicht zu ermitteln, deren Pose schließlich verfeinert und zurückgegeben wird.

Um die Modellansichten korrekt erstellen zu können, müssen die Parameter der zum Matching verwendeten Kamera in `CamParam` übergeben werden. Die Kameraparameter sind notwendig um z.B. die Skalierung der Projektionen mit Hilfe der Brennweite festzulegen. Des weiteren werden sie benötigt, um radiale Verzeichnungen des Objektivs korrekt berücksichtigen zu können. Daher ist es absolut notwendig, die verwendete Kamera vor der Erstellung des 3D-Formmodells mit `calibrate_cameras` zu kalibrieren. Auf der einen Seite führt dies

dazu, dass die von find_shape_model_3d zurückgegebenen Ergebnisse genauer sind. Auf der anderen Seite wird dadurch die Anwendung des 3D-Matchings selbst für Objektive mit deutlichen radialen Verzeichnungen ermöglicht.

Der Pose-Bereich, für den Modellansichten erstellt werden, kann mit den Parametern RefRotX, RefRotY, RefRotZ, OrderOfRotation, LongitudeMin, LongitudeMax, LatitudeMin, LatitudeMax, CamRollMin, CamRollMax, DistMin und DistMax festgelegt werden. Zu beachten ist, dass das Modell nur dann während des Matchings erkannt wird, wenn es innerhalb des angegebenen Pose-Bereichs liegt. Die Parameter werden im Folgenden beschrieben:

Vor der Berechnung der Ansichten wird der Ursprung des Koordinatensystems des 3D-Objektmodells in den Referenzpunkt des 3D-Objektmodells verschoben. Der Referenzpunkt ist der Mittelpunkt des kleinsten umschließenden achsenparallelen Quaders und kann mit get_object_model_3d_params abgefragt werden. Die virtuellen Kameras, die zum Erstellen der Ansichten verwendet werden, werden so um das 3D-Objektmodell angeordnet, dass deren Blickrichtung auf den Ursprung des Koordinatensystems gerichtet ist, d.h. die z-Achsen der Kameras gehen durch den Ursprung. Der Pose-Bereich kann dann dadurch angegeben werden, dass die Ansichten auf ein bestimmtes Viereck auf einer Kugel um den Ursprung beschränkt werden. Dabei bietet sich die Verwendung der sphärischen Koordinaten Länge, Breite und Radius an. Das sphärische Koordinatensystem ist so gewählt, dass die Äquatorebene der xz-Ebene des kartesischen Koordinatensystem entspricht, wobei die y-Achse zum Südpol (negative Breite) und die negative z-Achse zum Nullmeridian zeigt (vgl. convert_point_3d_spher_to_cart oder convert_point_3d_cart_to_spher für weitere Details zur Konvertierung zwischen kartesischen und sphärischen Koordinaten). Der Vorteil dieser Definition ist, dass eine Kamera mit der Pose [0,0,z,0,0,0,0] ihr optisches Zentrum bei Länge=0, Breite=0 und Radius=z hat. In diesem Fall entspricht der Radius der Distanz des optischen Zentrums der Kamera zum Referenzpunkt des 3D-Objektmodells.

Der Längenbereich, für den Ansichten generiert werden sollen, kann durch LongitudeMin und LongitudeMax angegeben werden (beide im Bogenmaß). Entsprechend kann der Breitenbereich durch LatitudeMin und LatitudeMax festgelegt werden (ebenso im Bogenmaß). LongitudeMin und LongitudeMax sind so eingestellt, dass sie einen Bereich von 360° (2π) beibehalten. Wenn eine Anpassung möglich ist, bleiben LongitudeMin und der Bereich erhalten. Die minimale und maximale Distanz zwischen Kamerazentrum und Modell-Referenzpunkt können in DistMin und DistMax übergeben werden. Dabei befindet sich der Ursprung des Modells im Zentrum des kleinsten einschließenden Quaders und stimmt nicht notwendigerweise mit dem Ursprung des CAD-Koordinatensystems überein. Zu beachten ist, dass die Distanzen in der Einheit Meter angegeben werden müssen (davon ausgehend, dass der Parameter Scale beim Einlesen der CAD-Datei mit read_object_model_3d korrekt gesetzt wurde). Schließlich können in CamRollMin und CamRollMax der minimale und maximale Rollwinkel der Kamera angegeben werden. Dieses Intervall beschreibt die erlaubte Kamerarotation um deren z-Achse bezüglich des 3D-Objektmodells. Falls die Bildebene parallel zur Ebene liegt, auf der sich die Objekte befinden, und bekannt ist, dass die Objekte in dieser Ebene nur innerhalb eines eingeschränkten Bereichs rotieren, ist es sinnvoll, diesen Bereich in CamRollMin und CamRollMax anzugeben. In allen anderen Fällen ist die Interpretation des Kamera-Rollwinkels schwierig. In diesen Fällen sollte daher das Intervall auf $[-\pi, +\pi]$ gesetzt sein. Zu beachten ist, dass je größer der Pose-Bereich gewählt wird desto größer ist der Speicherbedarf für das Modell und desto langsamer wird das Matching.

Die Orientierung des Koordinatensystems des 3D-Objektmodells ist durch die Koordinaten innerhalb der mit read_object_model_3d eingelesenen CAD-Datei festgelegt. Daher ist es sinnvoll, das 3D-Objektmodell vorab in eine Referenzorientierung zu drehen, die ungefähr im Zentrum des Pose-Bereiches liegt. Dies kann durch die Angabe entsprechender Werte für RefRotX, RefRotY, RefRotZ und OrderOfRotation erreicht werden. Die Rotation erfolgt dabei um die Achsen des 3D-Objektmodells, dessen Ursprung in den Referenzpunkt verschoben wurde. Folglich können die angegebenen Längen- und Breitenbereiche als Variationen der 3D-Objektmodell-Pose bezüglich der Referenzorientierung interpretiert werden. Die Referenzorientierung kann auf zwei verschiedene Arten angegeben werden. In der ersten Möglichkeit können die drei Rotationswinkel in RefRotX, RefRotY und RefRotZ und die Reihenfolge, in der die Rotationen angewendet werden, übergeben werden. Für die Rotationsreihenfolge kann dabei entweder *'gba'* oder *'abg'* gewählt werden. In der zweiten Möglichkeit können die drei Komponenten des Rodriguez-Rotationsvektors in RefRotX, RefRotY und RefRotZ übergeben werden. In diesem Fall muss für OrderOfRotation *'rodriguez'* übergeben werden (für weitere Details über Rotationsreihenfolgen und die Definition des Rodriguez-Vektors siehe create_pose).

Demnach werden zwei Transformationen auf das 3D-Objektmodell angewendet bevor die Modellansichten erstellt werden. Die erste Transformation ist die Translation des Koordinatenursprungs in den Modell-Referenzpunkt. Die zweite Transformation ist die Rotation des 3D-Objektmodells in die gewünschte Referenzorientierung um die Achsen des Referenzkoordinatensystems. Durch Kombinieren beider Transformationen erhält man die Referenz-Pose des 3D-Formmodells. Demnach beschreibt die Referenz-Pose des 3D-Formmodells die Lage des Referenzkoordinatensystems bezüglich des durch die CAD-Datei definierten Koordinatensystems des 3D-Objektmodells. Seien

$t = (x, y, z)'$ die Koordinaten des Referenzpunktes des 3D-Objektmodells und R die Rotationsmatrix mit der Referenzorientierung. Dann wird ein Punkt p_m, der im 3D-Objektmodellkoordinatensystem gegeben ist, durch die Anwendung der folgenden Formel in einen Punkt p_r im Referenzkoordinatensystem des 3D-Formmodels transformiert:

$$p_r = R \cdot (p_m - t)$$

Diese Transformation kann durch eine homogene 3D-Transformationsmatrix ausgedrückt werden oder alternativ dazu in Form einer 3D-Lage. Letztere kann mit dem Operator `get_shape_model_3d_params` abgefragt werden, indem für `GenParamName` *'reference_pose'* übergeben wird. Obige Formel kann man sich am besten als Pose des Pose-Typs 8, 10 oder 12 vorstellen, abhängig vom gewählten Wert für `OrderOfRotation` (für weitere Details über die verschiednen Typen von Poses siehe `create_pose`). Zu beachten ist allerdings, dass die mit `get_shape_model_3d_params` zurückgegebene Pose immer vom Typ 0 ist. Eine Pose, die in einem der beiden Koordinatensysteme gegeben ist kann mit dem Operator `trans_pose_shape_model_3d` in das andere Koordinatensystem transformiert werden.

Zu beachten ist außerdem, dass das Referenzkoordinatensystem nur deswegen verwendet wird, um die Festlegung des Pose-Bereiches möglichst einfach zu gestalten. Die beim 3D-Matching von `find_shape_model_3d` zurückgelieferte Pose bezieht sich dagegen stets auf das ursprüngliche durch die CAD-Datei definierten Koordinatensystems des 3D-Objektmodells.

Mit `MinContrast` wird festgelegt, welchen Grauwertkontrast das Modell später bei der Erkennung mit `find_shape_model_3d` im Bild mindestens besitzen muss. Mit anderen Worten stellt dieser Parameter somit eine Abgrenzung des Modells von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte `MinContrast` auf 10 gesetzt werden. Falls mehrkanalige Bilder für die Suchbilder verwendet werden, muss das Rauschen in einem Kanal noch mit der Wurzel der Anzahl der Kanäle multipliziert werden, um `MinContrast` zu bestimmen. Falls die Grauwerte in einem Kanal z.B. in einem Bereich von 10 Graustufen schwanken, sollte `MinContrast` bei einem dreikanaligen Bild auf 17 gesetzt werden. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss `MinContrast` entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte `MinContrast` etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells mit `find_shape_model_3d` zu gewährleisten.

Die oben beschriebenen Parameter sind anwendungsabhängig und müssen demnach immer bei der Erstellung eines 3D-Formmodells angegeben werden. Daneben gibt es generische Parameter, die optional angegeben werden können, um die Modellerstellung zu beeinflussen. Für die meisten Anwendungen brauchen diese Parameter nicht spezifiziert werden sondern können mit den Default-Werten übernommen werden. Fall erwünscht, können die generischen Parameter und deren Werte in `GenParamName` und `GenParamValue` übergeben werden. Die folgenden Werte für `GenParamName` sind möglich:

'num_levels': Aus Effizienzgründen werden die Modellansichten auf mehreren Pyramidenebenen erstellt. Auf höheren Ebenen werden weniger Ansichten generiert als auf niedrigeren Ebenen. Mit dem Parameter *'num_levels'* kann die Anzahl der Pyramidenebenen angegeben werden. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von *'num_levels'* ist aber darauf zu achten, dass die Formbeschreibungen der Ansichten auf der obersten Pyramidenstufe noch erkennbar sind und genügend viele Punkte besitzen (mindestens vier). Falls nicht genügend Modellpunkte für eine bestimmte Ansicht erzeugt werden, wird diese Ansicht aus dem Modell gelöscht und durch eine Ansicht auf einer darunter liegenden Ebene ersetzt. Wenn für alle Ansichten einer Ebene nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe für mindestens eine Ansicht genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entstehen würde, d.h. falls für alle Ansichten bereits auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert `create_shape_model_3d` eine Fehlermeldung zurück. Falls für *'num_levels' 'auto'* (Default-Wert) übergeben wird, wählt `create_shape_model_3d` die Anzahl der Pyramidenstufen automatisch. In diesem Fall werden die Ansichten aller Pyramidenebenen dahingehend überprüft, ob die entsprechenden Formbeschreibungen noch erkennbar sind. Stellt sich dabei heraus, dass die Formbeschreibung zu unspezifisch ist, wird die entsprechende Ansicht aus dem Modell entfernt und durch eine Ansicht auf einer darunter liegenden Ebene ersetzt. Zu beachten ist, dass es vorkommen kann, dass für unterschiedliche Ansichten unterschiedlich viele Pyramidenebenen erstellt werden, wenn für *'num_levels' 'auto'* übergeben wird, In seltenen Fällen kann es vorkommen, dass `create_shape_model_3d` die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für `MinScore`

oder `Greediness` in `find_shape_model_3d` selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in `find_shape_model_3d` kommen. In diesen Fällen sollten die Ansichten auf den verschiedenen Pyramidenebenen mit `get_shape_model_3d_contours` überprüft werden.

Wertevorschläge: 'auto', *3*, *4*, *5*, *6*

Defaultwert: 'auto'

'fast_pose_refinement': Der Parameter legt fest, ob die Pose-Verfeinerung während der Suche mit `find_shape_model_3d` in einem beschleunigten Modus durchgeführt werden soll. Falls für *'fast_pose_refinement' 'false'* gewählt wird, kann die Pose-Verfeinerung bei komplexen Modellen mit vielen Flächen einen signifikanten Teil der Rechenzeit ausmachen. Falls für *'fast_pose_refinement' 'true'* gewählt wird, werden für die Pose-Verfeinerung notwendige Berechnungen bereits während der Modellerzeugung ausgeführt und im Modell gespeichert. Die Pose-Verfeinerung während der Suche geht dann deutlich schneller. Zu beachten ist allerdings, dass die Größe des Modells in diesem Fall deutlich anwachsen kann (typischerweise um weniger als 30 Prozent). Außerdem ist zu beachten, dass es bei `find_shape_model_3d` in Abhängigkeit von der Wahl von *'fast_pose_refinement'* zu leicht unterschiedlichen Ergebnis-Posen kommen kann, da bei der Wahl von *'true'* intern die Pose-Verfeinerung approximiert wird.

Werteliste: 'true', *'false'*,

Defaultwert: 'true'

'lowest_model_level': In manchen Fällen ist der Modellgenerierungsprozess sehr zeitaufwändig und der Speicherbedarf des resultierenden Modells sehr hoch. Der Grund hierfür ist, dass in diesen Fällen die Anzahl der Ansichten, die generiert und im Modell gespeichert werden müssen, sehr hoch ist. Je größer der Pose-Bereich gewählt wird und je größer die Objekte im Bild erscheinen (gemessen in Pixeln), desto mehr Ansichten werden benötigt. Vor allem die Verwendung von großen Bildern (z.B. größer als 640×480) kann folglich zu sehr großen Modellen führen. Da die Anzahl der Ansichten auf den untersten Pyramidenebene am größten ist, bietet der Parameter *'lowest_model_level'* die Möglichkeit, die unterste(n) Pyramidenebene(n) von der Ansichtengenerierung auszuschließen. Der Wert, der für *'lowest_model_level'* übergeben wird, bestimmt die unterste Pyramidenebene, bis zu der Ansichten generiert und im Modell gespeichert werden. Wird z.B. für *'lowest_model_level'* der Wert *2* übergeben, verringert sich bei großen Modellen die Modellgenerierungszeit und die Modellgröße auf ca. ein Drittel der ursprünglichen Werte. Wird *'lowest_model_level'* nicht angegeben, werden Ansichten bis zur untersten Pyramidenebene generiert, was dem Verhalten entspricht, wenn für *'lowest_model_level' 1* übergeben wird. Wird für *'lowest_model_level'* ein Wert größer als *1* übergeben, wird bei der späteren Suche mit `find_shape_model_3d` die Verfolgung der Matches durch die Pyramide zunächst auf der angegebenen Ebene abgebrochen. Wird bei `find_shape_model_3d` für die Verfeinerung der Posen allerdings ein Least-Squares Adjustment gewählt, werden die Matches anschließend mit Hilfe des Least-Squares Adjustments auf der untersten Ebene verfeinert. Zu beachten ist, dass für verschiedene Werte von *'lowest_model_level'* bei der Suche unterschiedliche Matches gefunden werden können. Des weiteren ist der Score der Matches abhängig von der bei der Suche gewählten Verfeinerungsmethode. Außerdem ist zu beachten, dass je größer *'lowest_model_level'* gewählt wird, desto größer wird der Anteil der Verfeinerung an der Gesamtlaufzeit von `find_shape_model_3d`. Daher wirkt sich auch der Wert des generischen Parameters *'fast_pose_refinement'* (siehe oben) mit größeren Werten für *'lowest_model_level'* stärker aus. Auf der einen Seite kann ein großer Wert für *'lowest_model_level'* bei ausgeschaltetem *'fast_pose_refinement'* (*'false'*) zu sehr langen Rechenzeiten während der Suche führen. Auf der anderen Seite kann ein großer Wert für *'lowest_model_level'* bei eingeschaltetem *'fast_pose_refinement'* (*'true'*) zu ungenaueren Ergebnissen führen, da intern die Pose-Verfeinerung nur approximiert wird. Der Wert für *'lowest_model_level'* sollte demnach so klein wie möglich gewählt werden. Außerdem sollte *'lowest_model_level'* so klein gewählt werden, dass der Großteil der Kanten des 3D-Objektmodells auf dieser Pyramidenebene noch erkennbar sind.

Wertevorschläge: 1, *2*, *3*

Defaultwert: 1

'optimization': Bei Modellen mit besonders großen Ansichten kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters *'optimization'* auf einen Wert ungleich *'none'* zu setzen. Falls *'optimization'* $=$ *'none'*, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter *'optimization'* reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei `find_shape_model_3d` notwendig werden, den Parameter `Greediness` auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei Modellen mit kleineren Ansichten bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen. Falls für *'optimization' 'auto'* übergeben wird, wählt `create_shape_model_3d` die Reduktion der Punkte automatisch.

Werteliste: 'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'

Default value: 'auto'

'metric': Dieser Parameter legt fest, unter welchen Bedingungen das Modell im Bild noch erkannt wird. Wenn *'metric' = 'ignore_part_polarity'*, dann darf sich die Kontrastpolarität nur zwischen verschiedenen Modellteilen ändern, wohingegen die Polarität von Modellpunkten innerhalb des selben Modellteils nicht wechseln darf. Zu beachten ist, dass der Begriff *'ignore_part_polarity'* missverstanden werden kann. Er bedeutet, dass Polaritätswechsel zwischen benachbarten Modellteilen den Score nicht beeinflussen und daher ignoriert werden. Geeignete Modellteile werden automatisch bestimmt. Die Größe der Teile kann mit dem generischen Parameter *'part_size'*, der weiter unten beschrieben ist, festgelegt werden. Zu beachten ist, dass diese Metrik nur mit einkanaligen Bildern verwendet werden kann. Wird ein mit dieser Metrik generiertes Modell mit `find_shape_model_3d` in einem mehrkanaligen Bild gesucht, wird ein Fehler zurückgegeben. Wenn *'metric' = 'ignore_local_polarity'*, dann können Objekte auch dann erkannt werden, wenn sich die Kontrastpolarität lokal für jeden Modellpunkt ändert. Diese Metrik kann sowohl für die Suche in einkanaligen als auch in mehrkanaligen Bildern verwendet werden. Die Metrik *'ignore_part_polarity'* sollte verwendet werden, wenn das Objekt in Bildern mit starker Hintergrundtextur oder mit Störobjekten gesucht werden soll, was ansonsten zu falschen Matches führen würde. Zu beachten ist, dass der von `find_shape_model_3d` berechnete Score im Allgemeinen für *'ignore_part_polarity'* kleiner ist als für *'ignore_local_polarity'*. Dies sollte bei der Wahl von `MinScore` des Operators `find_shape_model_3d` berücksichtigt werden.

Werteliste: 'ignore_local_polarity', 'ignore_part_polarity'

Defaultwert: 'ignore_local_polarity'

'part_size': Mit diesem Parameter wird die Größe der Objektteile festgelegt, die verwendet werden, wenn *'metric'* auf *'ignore_part_polarity'* (siehe oben) gesetzt wird. Die Größe muss in Pixeln angeben werden und sollte ungefähr zweimal so groß sein wie die Größe der Störtextur im Bild. Möchte man z.B. Objekte vor einem schachbrettartigen Hintergrund mit schwarzen und weißen Quadraten der Größe 5×5 Pixel finden, dann sollte *'part_size'* auf *10* gesetzt werden. Dabei ist zu beachten, dass größere Werte für *'part_size'* den Score von korrekten Instanzen ebenfalls verringern kann. Dies kann vor allem bei Objekten mit glänzenden oder reflektierenden Oberflächen auftreten. Folglich kann ein höherer Wert für *'part_size'* das Risiko erhöhen, richtige Instanzen nicht mehr finden zu können. Wurde für *'metric' 'ignore_local_polarity'* gewählt, wird der Wert von *'part_size'* ignoriert.

Wertevorschläge: 2, 3, 4, 6, 8, 10

Defaultwert: 4

'min_face_angle': Kanten des 3D-Objektmodells werden nur dann in die Formbeschreibung der Ansichten übernommen, wenn der Winkel zwischen den beiden benachbarten Flächen im 3D-Objektmodell, die diese Kante gemeinsam haben, größer ist als *'min_face_angle'*. Wird *'min_face_angle'* auf *0.0* gesetzt, werden alle Kanten übernommen. Wird *'min_face_angle'* auf π (entspricht 180 Grad) gesetzt, wird nur die Silhouette des 3D-Objektmodells übernommen. Dieser Parameter kann insbesondere dafür verwendet werden, um Kanten in gekrümmten Flächen, z.B. einer Zylinder- oder Kegeloberfläche, zu unterdrücken. Gekrümmte Flächen werden durch eine Vielzahl ebener Flächen approximiert. Die Kanten zwischen solchen Flächen sollten nicht in die Formbeschreibung aufgenommen werden, da sie auch in echten Bildern vom Modell nicht zu sehen sind. Deshalb sollte *'min_face_angle'* genügend hoch gesetzt werden um solche Kanten zu unterdrücken. Die Auswirkung unterschiedlicher Werte für *'min_face_angle'* kann vor dem Aufruf von `create_shape_model_3d` mit `project_object_model_3d` überprüft werden. Zu beachten ist, dass wenn Kanten in die Formbeschreibung aufgenommen werden, die im Suchbild nicht sichtbar sind, die Matchingperformance (Robustheit und Geschwindigkeit) deutlich absinken kann.

Wertevorschläge: 'rad(10)', 'rad(20)', 'rad(30)', 'rad(45)'

Defaultwert: 'rad(30)'

'min_size': Dieser Wert dient zur Selektion von signifikanten Komponenten des Modells basierend auf der Größe der Komponenten, d.h. zusammenhängende Komponenten, deren Anzahl von Punkten kleiner ist als diese Mindestgröße, werden unterdrückt. Dieser Schwellenwert wird pro Pyramidenstufe halbiert.

Wertevorschläge: 'auto', 0, 3, 5, 10, 20

Defaultwert: 'auto'

'model_tolerance': Der Parameter spezifiziert die Toleranz der projizierten Kanten des 3D-Objektmodells im Bild (in Pixel). Je größer der Wert gewählt wird, desto weniger Ansichten werden generiert. Ein größerer Wert hat demnach zur Folge, dass das Modell zum einen weniger Speicher benötigt und zum anderen schneller mit `find_shape_model_3d` gefunden werden kann. Andererseits kann sich die Robustheit des Matchings verringern, wenn für *'model_tolerance'* ein zu großer Wert gewählt wird. Daher sollte dieser Parameter nur

mit Sorgfalt verändert werden. Für die meisten Anwendungen stellt ein Wert von *1* einen guten Kompromiss zwischen Geschwindigkeit und Robustheit dar.

Wertevorschläge: 0, 1, 2

Defaultwert: 1

'union_adjacent_contours': Dieser Parameter legt fest, ob nach der Projektion mittels `project_shape_model_3d` benachbarte Konturen verbunden werden sollen. Diese Option is äquivalent zu einem anschließenden Aufruf von `union_adjacent_contours_xld`, ist allerdings deutlich schneller.

Werteliste: 'true', 'false'

Defaultwert: 'false'

Ist die Systemvariable (siehe `set_system`) *'opengl_hidden_surface_removal_enable'* auf *'true'* gesetzt (Standard, sofern es verfügbar ist), wird die Grafikkarte verwendet um die Berechnung der sichtbaren Flächen in den Modellansichten zu beschleunigen. Abhängig von der verwendeten Grafikkarte ist dies signifikant schneller als die analytische Sichtbarkeitsberechnung. Ist *'fast_pose_refinement'* auf *'true'* gesetzt, werden auch die Vorberechnungen für die Pose-Verfeinerung auf der Grafikkarte durchgeführt. Es ist zu beachten, dass die Ergebnisse der OpenGL Projektion leicht verschieden zu den Ergebnissen der analytischen Projektion sind.

Parameter

- ▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d ⇝ *handle*
  Handle des 3D-Objektmodells.
- ▷ **`CamParam`** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **`RefRotX`** (input_control) . . . . . angle.rad ⇝ *real*
  Referenzorientierung: Rotation um x-Achse bzw. x-Komponente des Rodriguez-Vektors (in Bogenmaß bzw. ohne Einheit).
  Defaultwert: 0
  Wertevorschläge: `RefRotX` ∈ {-1.57, -0.78, -0.17, 0., 0.17, 0.78, 1.57}
- ▷ **`RefRotY`** (input_control) . . . . . angle.rad ⇝ *real*
  Referenzorientierung: Rotation um y-Achse bzw. y-Komponente des Rodriguez-Vektors (in Bogenmaß bzw. ohne Einheit).
  Defaultwert: 0
  Wertevorschläge: `RefRotY` ∈ {-1.57, -0.78, -0.17, 0., 0.17, 0.78, 1.57}
- ▷ **`RefRotZ`** (input_control) . . . . . angle.rad ⇝ *real*
  Referenzorientierung: Rotation um z-Achse bzw. z-Komponente des Rodriguez-Vektors (in Bogenmaß bzw. ohne Einheit).
  Defaultwert: 0
  Wertevorschläge: `RefRotZ` ∈ {-1.57, -0.78, -0.17, 0., 0.17, 0.78, 1.57}
- ▷ **`OrderOfRotation`** (input_control) . . . . . string ⇝ *string*
  Bedeutung der Rotationswerte der Referenzorientierung.
  Defaultwert: 'gba'
  Werteliste: `OrderOfRotation` ∈ {'gba', 'abg', 'rodriguez'}
- ▷ **`LongitudeMin`** (input_control) . . . . . angle.rad ⇝ *real*
  Kleinste geographische Länge der Modellansichten.
  Defaultwert: -0.35
  Wertevorschläge: `LongitudeMin` ∈ {-0.78, -0.35, -0.17}
- ▷ **`LongitudeMax`** (input_control) . . . . . angle.rad ⇝ *real*
  Größte geographische Länge der Modellansichten.
  Defaultwert: 0.35
  Wertevorschläge: `LongitudeMax` ∈ {0.17, 0.35, 0.78}
  Restriktion: LongitudeMax >= LongitudeMin
- ▷ **`LatitudeMin`** (input_control) . . . . . angle.rad ⇝ *real*
  Kleinste geographische Breite der Modellansichten.
  Defaultwert: -0.35
  Wertevorschläge: `LatitudeMin` ∈ {-0.78, -0.35, -0.17}
  Restriktion: - pi / 2 <= LatitudeMin && LatitudeMin <= pi / 2

▷ **LatitudeMax** (input_control) . . . . . angle.rad ⇝ *real*
Größte geographische Breite der Modellansichten.
Defaultwert: 0.35
Wertevorschläge: `LatitudeMax` ∈ {0.17, 0.35, 0.78}
Restriktion: - pi / 2 <= LatitudeMax && LatitudeMax <= pi / 2 && LatitudeMax >= LatitudeMin

▷ **CamRollMin** (input_control) . . . . . angle.rad ⇝ *real*
Kleinster Kamerarollwinkel der Modellansichten.
Defaultwert: -3.1416
Wertevorschläge: `CamRollMin` ∈ {-3.14, -1.57, -0.39, 0.0, 0.39, 1.57, 3.14}

▷ **CamRollMax** (input_control) . . . . . angle.rad ⇝ *real*
Größter Kamerarollwinkel der Modellansichten.
Defaultwert: 3.1416
Wertevorschläge: `CamRollMax` ∈ {-3.14, -1.57, -0.39, 0.0, 0.39, 1.57, 3.14}
Restriktion: CamRollMax >= CamRollMin

▷ **DistMin** (input_control) . . . . . number ⇝ *real*
Kleinster Kamera-Objekt-Abstand der Modellansichten.
Defaultwert: 0.3
Wertevorschläge: `DistMin` ∈ {0.05, 0.1, 0.2, 0.5}
Restriktion: DistMin > 0

▷ **DistMax** (input_control) . . . . . number ⇝ *real*
Größter Kamera-Objekt-Abstand der Modellansichten.
Defaultwert: 0.4
Wertevorschläge: `DistMax` ∈ {0.1, 0.2, 0.5, 1.0}
Restriktion: DistMax >= DistMin

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer*
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 10
Wertevorschläge: `MinContrast` ∈ {1, 2, 3, 5, 7, 10, 20, 30, 1000, 2000, 5000}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen von (optionalen) Parametern für die Steuerung des Verhaltens des Operators.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'num_levels', 'fast_pose_refinement', 'lowest_model_level', 'optimization', 'metric', 'part_size', 'min_face_angle', 'min_size', 'model_tolerance', 'union_adjacent_contours'}

▷ **GenParamValue** (input_control) . . . . . attribute.name(-array) ⇝ *integer* / real / string
Die zu den optionalen generischen Parametern gehörenden Werte.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {0, 1, 2, 3, 4, 6, 8, 10, 'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high', 0.1, 0.2, 0.3, 'ignore_local_polarity', 'ignore_part_polarity', 'true', 'false'}

▷ **ShapeModel3DID** (output_control) . . . . . shape_model_3d ⇝ *handle*
Handle des 3D-Formmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_shape_model_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn die Parameter so gewählt worden sind, dass alle Modellansichten zu wenige Punkte besitzen, wird die Fehlermeldung 8510 zurückgeliefert. Für den Fall, dass das projizierte Modell in mindestens einer Ansicht größer als die doppelte Bildgröße ist, wird die Fehlermeldung 8910 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_object_model_3d, project_object_model_3d, get_object_model_3d_params

Nachfolger

find_shape_model_3d, write_shape_model_3d, project_shape_model_3d, get_shape_model_3d_params, get_shape_model_3d_contours

Siehe auch

convert_point_3d_cart_to_spher, convert_point_3d_spher_to_cart, create_cam_pose_look_at_point, trans_pose_shape_model_3d

Literatur

Markus Ulrich, Christian Wiedemann, Carsten Steger, „Combining Scale-Space and Similarity-Based Aspect Graphs for Fast 3D Object Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1902-1914, Oct., 2012.

Modul

3D Metrology

deserialize_shape_model_3d

```
deserialize_shape_model_3d ( : : SerializedItemHandle : ShapeModel3DID )
```

Deserialisieren eines 3D-Formmodells.

deserialize_shape_model_3d deserialisiert ein 3D-Formmodell, welches mit serialize_shape_model_3d serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte 3D-Formmodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten 3D-Formmodell mit dem Handle ShapeModel3DID gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **ShapeModel3DID** (output_control) ..... shape_model_3d ⇝ *handle*
  Handle des 3D-Formmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_shape_model_3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_shape_model_3d

Nachfolger

find_shape_model_3d, get_shape_model_3d_params

Siehe auch

create_shape_model_3d, clear_shape_model_3d

Modul

3D Metrology

find_shape_model_3d

```
find_shape_model_3d ( Image : : ShapeModel3DID, MinScore,
    Greediness, NumLevels, GenParamName, GenParamValue : Pose,
    CovPose, Score )
```

Suche der besten Matches eines 3D-Formmodells in einem Bild.

find_shape_model_3d findet die besten Matches des 3D-Formmodells ShapeModel3DID im Eingabebild Image. Das 3D-Formmodell muss zuvor mit create_shape_model_3d erzeugt oder mit read_shape_model_3d eingelesen worden sein.

Die 3D-Lage der gefundenen Modellinstanzen wird in Pose zurückgeliefert. Sie ist in der Form ${}^{ccs}\mathbf{P}_{mcs}$. Dabei steht *ccs* für das Kamerakoordinatensystem und *mcs* für das Modellkoordinatensystem (ein 3D Weltkoordinatensystem), siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". Dies bedeutet, sie beschreibt die Lage des 3D-Objektmodells im Kamerakoordinatensystem. Zu beachten ist, dass sich die resultierende Pose nicht auf das in create_shape_model_3d eingeführte Referenzkoordinatensystem bezieht, sondern auf das ursprüngliche durch die CAD-Datei definierte Koordinatensystem des 3D-Objektmodells. Falls eine 3D-Lagen-Verfeinerung durchgeführt wurde (siehe unten), werden zusätzlich die Genauigkeiten der sechs 3D-Lagen-Parameter in CovPose zurückgeliefert. Standardmäßig enthält CovPose die 6 Standardabweichungen der 3D-Lagen-Parameter pro Match. Wurde der generische Parameter *'cov_pose_mode'* (siehe unten) dagegen auf *'covariances'* gesetzt, enthält CovPose die 36 Werte der kompletten 6×6 Kovarianzmatrix der 6 Posenwerte. Zu beachten ist, dass es dabei nur um eine innere Genauigkeit der 3D-Lagen-Verfeinerung handelt. Die tatsächliche Genauigkeit der 3D-Lage kann davon abweichen. Schließlich wird in Score die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes Image gibt den Suchbereich für den Referenzpunkt des 3D-Objektmodells an.

MinScore: Der Parameter MinScore legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von MinScore gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, dass das Modell niemals verdeckt wird, kann MinScore auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden. Zu beachten ist, dass in Bildern mit vielen Störobjekten oder mit starker Hintergrundtextur der Wert für MinScore nicht deutlich niedriger als *0.7* gesetzt werden sollte, da ansonsten falsche Matches gefunden werden könnten.

Greediness: Der Parameter Greediness bestimmt, wie „gierig" die Suche durchgeführt werden soll. Für Greediness $= 0$ wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden, immer findet. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für Greediness $= 1$ wird eine unsicherere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, dass das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für Greediness $= 1$ wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten Fällen wird das 3D-Formmodell für Greediness $= 0.9$ immer sicher gefunden.

NumLevels: Mit NumLevels wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit create_shape_model_3d angegebenen Bereich beschnitten. Falls NumLevels als *0* angegeben wird, wird die mit create_shape_model_3d angegebene Anzahl verwendet. Optional kann NumLevels einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, um Laufzeit einzusparen. Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden.

GenParamName und GenParamValue: Zusätzlich zu den oben beschriebenen Parametern gibt es generische Parameter, die optional angegeben werden können, um das Matching zu beeinflussen. Für die meisten Anwendungen müssen diese Parameter nicht spezifiziert werden, sondern können mit den Default-Werten übernommen werden. Fall erwünscht, können die generischen Parameter und deren Werte in GenParamName und GenParamValue übergeben werden. Die folgenden Werte für GenParamName sind möglich:

- Für den Fall, dass der Pose-Bereich, in dem das Modell gesucht werden soll, kleiner ist als der Pose-Bereich, der bei der Modellgenerierung mit create_shape_model_3d angegeben wurde, kann der Pose-Bereich mit den folgenden Parametern eingeschränkt werden. Falls die Werte außerhalb des Pose-Bereichs des Modell liegen, werden diese automatisch auf den Pose-Bereich des Modells beschnitten.

'longitude_min': Setzt die kleinste geographische Länge des Pose-Bereichs.
Wertevorschläge: 'rad(-45)', 'rad(-30)', 'rad(-15)'
Defaultwert: 'rad(-180)'

'longitude_max': Setzt die größte geographische Länge des Pose-Bereichs.
Wertevorschläge: 'rad(15)', 'rad(30)', 'rad(45)'
Defaultwert: 'rad(180)'

'latitude_min': Setzt die kleinste geographische Breite des Pose-Bereichs.
Wertevorschläge: 'rad(-45)', 'rad(-30)', 'rad(-15)'
Defaultwert: 'rad(-90)'

'latitude_max': Setzt die größte geographische Breite des Pose-Bereichs.
Wertevorschläge: 'rad(15)', 'rad(30)', 'rad(45)'
Defaultwert: 'rad(90)'

'cam_roll_min': Setzt den kleinsten Kamerarollwinkel des Pose-Bereichs.
Wertevorschläge: 'rad(-45)', 'rad(-30)', 'rad(-15)'
Defaultwert: 'rad(-180)'

'cam_roll_max': Setzt den größten Kamerarollwinkel des Pose-Bereichs
Wertevorschläge: 'rad(15)', 'rad(30)', 'rad(45)'
Defaultwert: 'rad(180)'

'dist_min': Setzt den kleinsten Kamera-Objekt-Abstand des Pose-Bereichs.
Wertevorschläge: 0.05, 0.1, 0.5, 1.0
Defaultwert: 0

'dist_max': Setzt den größten Kamera-Objekt-Abstand des Pose-Bereichs.
Wertevorschläge: 0.05, 0.1, 0.5, 1.0
Defaultwert: (∞)

- Weitere generische Parameter, die nicht den Pose-Bereich betreffen, können wie folgt angegeben werden:

  'num_matches': Mit diesem Parameter kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als die angegebene Anzahl Instanzen eine Bewertung größer als `MinScore` erreichen, werden nur die besten *'num_matches'* Instanzen zurückgeliefert. Falls weniger als *'num_matches'* Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter `MinScore` hat Vorrang vor *'num_matches'*. Wird für *'num_matches'* der Wert *0* übergeben, werden alle Matches, die das Score-Kriterium erfüllen zurückgegeben. Zu beachten ist, dass je mehr Instanzen gefunden werden sollen, desto langsamer wird das Matching sein.
  Wertevorschläge: 0, 1, 2, 3
  Defaultwert: 1

  'max_overlap': Es kann vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Posen gefunden werden. Mit dem Parameter *'max_overlap'* kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als *'max_overlap'* überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe `smallest_rectangle2`). Wurde in `create_shape_model_3d` für *'lowest_model_level'* ein Wert größer als *1* übergeben, basiert die Berechnung anstatt auf den Konturen auf dem kleinsten umschließenden Rechteck der Projektion des kleinsten umschließenden achsenparallelen Quaders des 3D-Objektmodells. Da in diesem Fall die Überlappung überschätzt werden kann, ist es in manchen Fällen nötig, einen größeren Wert für *'max_overlap'* zu übergeben. Bei $'max_overlap' = 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei $'max_overlap' = 1$ werden alle gefundenen Instanzen zurückgeliefert.
  Wertevorschläge: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0
  Defaultwert: 0.5

  'pose_refinement': Dieser Parameter gibt an, ob die Posen der gefundenen Instanzen verfeinert werden sollen. Falls für *'pose_refinement' 'none'* gewählt wird, wird die Pose des Modells nur mit einer eingeschränkten Genauigkeit bestimmt. In diesem Fall hängt die Genauigkeit von verschiedenen Abtastschrittweiten ab, die während der Suche verwendet werden. Daher lässt sich die damit erzielbare Genauigkeit nur schlecht abschätzen. Aus diesem Grund sollte für *'pose_refinement'* nur dann *'none'* gewählt werden, wenn es in erster Linie auf die Geschwindigkeit ankommt und eine

Näherungspose ausreichend ist. In allen anderen Fällen sollte die Pose durch Ausgleichsrechnung, d.h., durch Minimierung der Abstände der Modellpunkte und der zugehörigen Bildpunkte (Least-Squares Adjustment), bestimmt werden. Um eine möglichst hohe Genauigkeit zu erzielen, wird die Pose-Verfeinerung direkt in 3D durchgeführt. Daher kostet dieser Modus auch zusätzliche Rechenzeit. Ist die Systemvariable (siehe `set_system`) *'opengl_hidden_surface_removal_enable'* auf *'true'* gesetzt (Standard, sofern es verfügbar ist) und wurde das Modell mit *'fast_pose_refinement'* auf *'false'* gesetzt erzeugt, wird die Modellprojektion für die Pose-Verfeinerung auf der Grafikkarte durchgeführt. Abhängig von der verwendeten Grafikkarte ist dies signifikant schneller als die analytische Projektion. Es ist zu beachten, dass die Ergebnisse der OpenGL Projektion leicht verschieden zu denen der analytischen Projektion sind. Mit den verschiedenen Modi (*'least_squares'*, *'least_squares_high'* und *'least_squares_very_high'*) kann dabei die Genauigkeit festgelegt werden, mit der der minimale Abstand gesucht wird. Je höher die Genauigkeit gewählt wird, desto länger dauert allerdings auch die Pose-Verfeinerung. Im Normalfall sollte *'pose_refinement'* auf *'least_squares_high'* gesetzt werden, da sich so der optimale Kompromiss zwischen Laufzeit und Genauigkeit ergibt. Zu beachten ist, dass die Pose-Verfeinerung durch die Wahl von *'fast_pose_refinement'* für den Parameter `GenParamName` des Operators `create_shape_model_3d` beschleunigt werden kann.
Werteliste: 'none', *'least_squares'*, *'least_squares_high'*, *'least_squares_very_high'*
Defaultwert: 'least_squares_high'

'recompute_score': Dieser Parameter gibt an, ob der Score der Matches nach der Pose-Verfeinerung neu berechnet wird. Wird für *'recompute_score' 'false'* gewählt, wird der Score zurückgegeben, der vor der Pose-Verfeinerung errechnet wurde. In manchen Fällen ändert sich die Objekt-Pose durch die Pose-Verfeinerung allerdings um mehr als nur ein Pixel im Bild. Folglich beschreibt der ursprüngliche Score unter Umständen den verfeinerten Match nicht mehr angemessen. Dies kann dazu führen, dass falsche Matches einen relativ hohen Score erhalten oder dass perfekte Matches einen relativ geringen Score erhalten. Um einen aussagekräftigeren Score zu erhalten, kann der Score nach der Pose-Verfeinerung neu berechnet werden indem der Parameter *'recompute_score'* auf *'true'* gesetzt wird. Zu beachten ist, dass sich dadurch die Reihenfolge der zurückgegebenen Matches ändern kann. Außerdem kann sich auch die Auswahl der zurückgegebenen Matches ändern. Des Weiteren ist zu beachten, dass die Neuberechnung der Score-Werte zusätzliche Rechenzeit beansprucht. Die zusätzliche Rechenzeit kann durch das Setzen des Parameters *'fast_pose_refinement'* des Operators `create_shape_model_3d` auf *'true'* reduziert werden.
Werteliste: 'false', *'true'*
Defaultwert: 'false'

'outlier_suppression': Dieser Parameter wirkt sich nur dann aus, wenn für *'pose_refinement'* ein anderer Wert als *'none'* gewählt wird, also eine Ausgleichsrechnung durchgeführt wird. Dann kann es in einigen Fällen sinnvoll sein, während der Ausgleichsrechnung eine robuste Ausreißerunterdrückung anzuwenden. Dies ist insbesondere dann hilfreich, wenn im Bild ein hoher Grad an Störobjekten vorhanden ist, die verhindern, dass die Ausgleichsrechnung die optimale Pose findet. In diesem Fall sollte *'outlier_suppression'* entweder auf *'medium'* (eliminiert eine mittlere Anzahl an Ausreißern) oder auf *'high'* (eliminiert eine hohe Anzahl an Ausreißern) gesetzt werden. Für die meisten Anwendungen ist jedoch keine Ausreißerunterdrückung notwendig, d.h. *'pose_refinement'* kann auf *'none'* gesetzt werden. Zu beachten ist auch, dass eine aktivierte Ausreißerunterdrückung mit einer deutlich längeren Rechenzeit verbunden ist.
Wertelist: 'none', *'medium'*, *'high'*
Defaultwert: 'none'

'cov_pose_mode': Dieser Parameter wirkt sich nur dann aus, wenn für *'pose_refinement'* ein anderer Wert als *'none'* gewählt wird, also eine Ausgleichsrechnung durchgeführt wird. *'cov_pose_mode'* legt fest, in welcher Form die Genauigkeiten `CovPose` der in der Ausgleichsrechnung berechneten Pose-Parameter zurückgegeben werden. Wird für *'cov_pose_mode' 'standard_deviations'* übergeben, so werden für die 6 Pose-Parameter eines Matches deren 6 Standardabweichungen zurückgegeben. Wird für *'cov_pose_mode'* dagegen *'covariances'* gewählt, enthält `CovPose` pro Match die 36 Werte der kompletten 6×6 Kovarianzmatrix der 6 Posenwerte.
Wertelist: 'standard_deviations', *'covariances'*
Defaultwert: 'standard_deviations'

'border_model': Im Normalfall wird das Modell innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als `MinScore` erreichen würde. Außerdem kann es sein, dass das Modell, wenn

es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann durch das Setzen von *'border_model'* auf *'true'* dahingehend verändert werden, dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als `MinScore` erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit `dilation_circle` zu vergrößern.
Werteliste: 'false', 'true'
Defaultwert: 'false'

Parameter

- ▷ **`Image`** (input_object) . . . . . (multichannel-)image ⇝ *object* : byte / uint2
  Eingabebild, in dem das Modell gefunden werden soll.
- ▷ **`ShapeModel3DID`** (input_control) . . . . . shape_model_3d ⇝ *handle*
  Handle des 3D-Formmodells.
- ▷ **`MinScore`** (input_control) . . . . . real ⇝ *real*
  Minimale Bewertung der zu findenden Instanzen des Modells.
  Defaultwert: 0.7
  Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
- ▷ **`Greediness`** (input_control) . . . . . real ⇝ *real*
  „Gierigkeit“ der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen“ werden).
  Defaultwert: 0.9
  Wertevorschläge: `Greediness` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich: $0 \leq$ `Greediness` ≤ 1
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
- ▷ **`NumLevels`** (input_control) . . . . . integer-array ⇝ *integer*
  Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls $|$`NumLevels`$| = 2$).
  Defaultwert: 0
  Werteliste: `NumLevels` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **`GenParamName`** (input_control) . . . . . attribute.name-array ⇝ *string*
  Namen von (optionalen) Parametern für die Steuerung des Verhaltens des Operators.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'longitude_min', 'longitude_max', 'latitude_min', 'latitude_max', 'cam_roll_min', 'cam_roll_max', 'dist_min', 'dist_max', 'num_matches', 'max_overlap', 'pose_refinement', 'cov_pose_mode', 'outlier_suppression', 'border_model', 'recompute_score'}
- ▷ **`GenParamValue`** (input_control) . . . . . attribute.name-array ⇝ *integer* / real / string
  Die zu den optionalen generischen Parametern gehörenden Werte.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {-0.78, -0.35, -0.17, 0.0, 0.17, 0.35, 0.78, 0.1, 0.2, 0.5, 'none', 'false', 'true', 'least_squares', 'least_squares_high', 'least_squares_very_high', 'standard_deviations', 'covariances', 'medium', 'high'}
- ▷ **`Pose`** (output_control) . . . . . pose(-array) ⇝ *real* / integer
  3D-Lage des 3D-Formmodells.
- ▷ **`CovPose`** (output_control) . . . . . real-array ⇝ *real*
  6 Standardabweichungen bzw. 36 Kovarianzen der Poseparameter.
- ▷ **`Score`** (output_control) . . . . . real-array ⇝ *real*
  Bewertung der gefundenen Instanzen des 3D-Formmodells.

Beispiel

```
read_object_model_3d (DXFModelFileName, 'm', [], [], ObjectModel3D, \
                      DxfStatus)
CamParam := ['area_scan_division',0.01221,2791,7.3958e-6,7.4e-6,\
             308.21,245.92,640,480]
create_shape_model_3d (ObjectModel3D, CamParam, 0, 0, 0, 'gba', \
                       -rad(20), rad(20), -rad(20), rad(20), 0, \
                       rad(360), 0.15, 0.2, 10, [], [], ShapeModel3DID)
grab_image_async (Image, AcqHandle, -1)
find_shape_model_3d (Image, ShapeModel3DID, 0.6, 0.9, 0, [], [], \
                     Pose, CovPose, Score)
project_shape_model_3d (ModelContours, ShapeModel3DID, CamParam, \
                        Pose, 'true', rad(30))
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_shape_model_3d` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wurde bei der Modellgenerierung mit `find_shape_model_3d` für *'metric'* der Wert *'ignore_part_polarity'* gewählt und in `Image` ein mehrkanaliges Bild übergeben, wird die Fehlermeldung 3359 zurückgegeben.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_shape_model_3d`, `read_shape_model_3d`

Nachfolger

`project_shape_model_3d`

Siehe auch

`convert_point_3d_cart_to_spher`, `convert_point_3d_spher_to_cart`, `create_cam_pose_look_at_point`, `trans_pose_shape_model_3d`

Modul

3D Metrology

`get_shape_model_3d_contours` `( : ModelContours : ShapeModel3DID, Level, View : ViewPose )`

Auslesen der Konturrepräsentation einer Ansicht eines 3D-Formmodells.

Der Operator `get_shape_model_3d_contours` gibt in `ModelContours` eine Repräsentation einer Modellansicht des 3D-Formmodells `ShapeModel3DID` als XLD-Konturen zurück. Die Parameter `Level` und `View` geben an, für welche Modellansicht die Konturrepräsentation zurückgegeben werden soll. Dabei bezeichnet `Level` die Pyramidenstufe und `View` die Ansicht auf dieser Pyramidenstufe.

Die erlaubten Wertebereiche für `Level` und `View` können zuvor mit dem Operator `get_shape_model_3d_params` ermittelt werden, indem man für `GenParamName` *'num_views_per_level'* übergibt.

Die Konturen können zur Visualisierung und Beurteilung des mit `create_shape_model_3d` erstellten Modells verwendet werden. Dies ermöglicht es abzuschätzen, ob die Anzahl der Pyramidenebenen korrekt gewählt wurde. Zeigen z.B. die Konturen auf der obersten Ebene nicht genügend Details, um für die Modellansicht noch repräsentativ zu sein, sollte bei der Suche mit `find_shape_model_3d` die Anzahl der Pyramidenebenen mit `NumLevels` entsprechend nach unten korrigiert werden. Für den Fall, dass die Konturen selbst auf der obersten Pyramidenebene noch zu viele Details enthalten, sollte bereits beim Erstellen des 3D-Formmodells mit `create_shape_model_3d` eine größere Anzahl an Pyramidenebenen gewählt werden.

Zusätzlich wird in ViewPose die Pose der gewählte Ansicht zurückgegeben. Diese kann z.B. verwendet werden, um das 3D-Formmodell in der der Ansicht entsprechenden Pose mit project_shape_model_3d zu projizieren. Das projizierte Modell kann dann als Grundlage für die oben beschriebene Bewertung der Modellkonturen ModelContours dienen. Zu beachten ist, dass aufgrund von radialen Verzeichnungen die Position der Konturen der Projektion leicht von der Position der Modellkonturen abweichen kann.

Parameter

- ▷ **ModelContours** (output_object) ..... xld_cont-array ⇝ *object*
  Konturrepräsentation der Modellansicht.
- ▷ **ShapeModel3DID** (input_control) ..... shape_model_3d ⇝ *handle*
  Handle des 3D-Formmodells.
- ▷ **Level** (input_control) ..... integer ⇝ *integer*
  Pyramidenebene, für die die Konturrepräsentation zurückgegeben werden soll.
  Defaultwert: 1
  Wertevorschläge: Level ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Restriktion: Level >= 1
- ▷ **View** (input_control) ..... integer ⇝ *integer*
  Ansicht, für die die Konturrepräsentation zurückgegeben werden soll.
  Defaultwert: 1
  Wertevorschläge: View ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Restriktion: View >= 1
- ▷ **ViewPose** (output_control) ..... pose ⇝ *real* / integer
  3D-Lage (Pose) des 3D-Formmodells in der Ansicht.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_shape_model_3d_contours den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_shape_model_3d, read_shape_model_3d, get_shape_model_3d_params

Nachfolger

create_shape_model_3d

Modul

3D Metrology

get_shape_model_3d_params (: : ShapeModel3DID,
GenParamName : GenParamValue)

Auslesen der Parameter eines 3D-Formmodells.

Mit Hilfe des Operators get_shape_model_3d_params lassen sich Parameter des 3D-Objektmodells abfragen.

Die Namen der Parameter, die abgefragt werden sollen, werden in GenParamName übergeben, die korrespondierenden Werte werden in GenParamValue zurückgeliefert.

Im Einzelnen kann auf die folgenden Parameter zugegriffen werden:

'cam_param': Interne Parameter der für das Matching zu verwendenden Kamera.

'ref_rot_x': Referenzorientierung: Rotation um x-Achse bzw. x-Komponente des Rodriguez-Vektors (in Bogenmaß bzw. ohne Einheit).

'ref_rot_y': Referenzorientierung: Rotation um y-Achse bzw. y-Komponente des Rodriguez-Vektors (in Bogenmaß bzw. ohne Einheit).

'ref_rot_z': Referenzorientierung: Rotation um z-Achse bzw. z-Komponente des Rodriguez-Vektors (in Bogenmaß bzw. ohne Einheit).

'order_of_rotation': Bedeutung der Rotationswerte der Referenzorientierung.

'longitude_min': Kleinste geographische Länge der Modellansichten.

'longitude_max': Größte geographische Länge der Modellansichten.

'latitude_min': Kleinste geographische Breite der Modellansichten.

'latitude_max': Größte geographische Breite der Modellansichten.

'cam_roll_min': Kleinster Kamerarollwinkel der Modellansichten.

'cam_roll_max': Größter Kamerarollwinkel der Modellansichten.

'dist_min': Kleinster Kamera-Objekt-Abstand der Modellansichten.

'dist_max': Größter Kamera-Objekt-Abstand der Modellansichten.

'min_contrast': Minimaler Kontrast des Objektes in den Suchbildern.

'num_levels': Vom Benutzer vorgegebene Anzahl von Pyramidenebenen.

'num_levels_max': Maximale verwendete Anzahl von Pyramidenebenen über alle Modellansichten.

'optimization': Art der Optimierung durch Reduzierung der Modellpunkte.

'metric': Art der zum Matchen verwendeten Metrik.

'part_size': Größe der verwendeten Modellteile, wenn *'metric'* auf *'ignore_part_polarity'* gesetzt wurde.

'min_face_angle': Kleinster 3D-Flächenwinkel für den die 3D-Objektmodellkante ins 3D-Formmodell einfließt.

'min_size': Mindestgröße der projizierten 3D-Objektmodellkante (in Anzahl Pixel) damit sie ins 3D-Formmodell einfließt.

'model_tolerance': Größte zulässige Abweichung der projizierten Modellkanten (in Pixel).

'num_views_per_level': Anzahl der Modellansichten pro Pyramidenebene. Für jede Pyramidenebene wird die Anzahl der im Modell gespeicherten Ansichten zurückgegeben. Die Länge des zurückgelieferten Tupels entspricht somit der Anzahl der verwendeten Pyramidenebenen, die mit *'num_levels_max'* abgefragt werden kann. Für Pyramidenebenen unterhalb von *'lowest_model_level'* (siehe Dokumentation zu `create_shape_model_3d`) wird der Wert *0* zurückgegeben.

'reference_pose': Referenzlage und -orientierung des 3D-Formmodells. Die zurückgelieferte 3D-Lage ist in der Form ${}^{rcs}\mathbf{P}_{mcs}$. Dabei steht *rcs* für das Referenzkoordinatensystem und *mcs* für das Modellkoordinatensystem (ein 3D Weltkoordinatensystem), siehe auch Transformationen / Posen und „`Solution Guide III-C - 3D Vision`". D.h., sie beschreibt die Lage des intern verwendeten Referenzkoordinatensystems des 3D-Formmodells bezüglich des im 3D-Objektmodell gespeicherten Koordinatensystems. Mit dieser 3D-Lage können Punkte aus dem Objektkoordinatensystem ins Referenzkoordinatensystem transformiert werden.

'reference_point': 3D-Koordinaten des Referenzpunktes des zu Grunde liegenden 3D-Objektmodells.

'bounding_box1': Kleinster umschließender achsenparalleler Quader des zu Grunde liegenden 3D-Objektmodells in der Form: [min_x, min_y, min_z, max_x, max_y, max_z].

'fast_pose_refinement': Beschreibt, ob die 3D-Lagen-Verfeinerung während der Suche im beschleunigten Modus (*'true'*) oder im herkömmlichen Modus (*'false'*) durchgeführt wird.

'lowest_model_level': Unterste Pyramidenebene, bis zu der Ansichten im Modell gespeichert sind.

'union_adjacent_contours': Beschreibt, ob in `project_shape_model_3d` benachbarte Konturen verbunden werden sollen.

Ausführlichere Informationen zu den einzelnen Parametern können beim Operator `create_shape_model_3d` nachgelesen werden.

Es ist möglich, in einem Operatoraufruf mehrere Parameter zu kombinieren und gemeinsam in einem Tupel an `GenParamName` zu übergeben. Als Ergebnis wird in `GenParamValue` ein Tupel gleicher Länge zurückgeliefert, dessen Werte mit den entsprechenden Parameternamen korrespondieren. Dies ist allerdings ausschließlich für solche Parameter möglich, die nur einen einzelnen Wert zurückliefern.

Parameter

▷ **ShapeModel3DID** (input_control) . . . . . shape_model_3d ⇝ *handle*
Handle des 3D-Formmodells.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für das 3D-Formmodell abgefragt werden sollen.
Defaultwert: 'num_levels_max'
Werteliste: GenParamName ∈ {'cam_param', 'ref_rot_x', 'ref_rot_y', 'ref_rot_z', 'order_of_rotation', 'longitude_min', 'longitude_max', 'latitude_min', 'latitude_max', 'cam_roll_min', 'cam_roll_max', 'dist_min', 'dist_max', 'min_contrast', 'num_levels', 'num_levels_max', 'optimization', 'metric', 'part_size', 'min_face_angle', 'min_size', 'model_tolerance', 'num_views_per_level', 'reference_pose', 'reference_point', 'bounding_box1', 'fast_pose_refinement', 'lowest_model_level', 'union_adjacent_contours'}

▷ **GenParamValue** (output_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Werte der generischen Parameter.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_shape_model_3d_params den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_shape_model_3d, read_shape_model_3d

Nachfolger

find_shape_model_3d

Siehe auch

convert_point_3d_cart_to_spher, convert_point_3d_spher_to_cart, create_cam_pose_look_at_point, trans_pose_shape_model_3d

Modul

3D Metrology

project_shape_model_3d (: ModelContours : ShapeModel3DID, CamParam, Pose, HiddenSurfaceRemoval, MinFaceAngle :)

Projizieren der Kanten eines 3D-Formmodells in Bildkoordinaten.

Der Operator project_shape_model_3d projiziert die Kanten des 3D-Objektmodells, das zur Erstellung des 3D-Formmodells ShapeModel3DID verwendet wurde, in das Bildkoordinatensystem und liefert die projizierten Kanten in ModelContours zurück. Die Koordinaten des 3D-Objektmodells sind im 3D-Weltkoordinatensystem (*mcs*) gegeben. Sie werden zuerst mittels der gegebenen 3D-Lage Pose in Kamerakoordinaten (*ccs*) transformiert und dann basierend auf den internen Kameraparametern CamParam in das Bildkoordinatensystem projiziert.

Die internen Kameraparameter CamParam beschreiben dabei die Abbildungseigenschaften der Kamera (siehe Kalibrierung). Die 3D-Lage Pose wird in der Form ${}^{ccs}\mathbf{P}_{mcs}$ erwartet, siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". Dies bedeutet, sie beschreibt die Position und Lage des Modellkoordinatensystems, welches über das 3D Objektmodel definiert ist, im Kamerakoordinatensystem.

Mit dem Parameter HiddenSurfaceRemoval kann die Verdeckungsberechnung ein- bzw. ausgeschaltet werden. Wenn der Parameter HiddenSurfaceRemoval auf *'true'* gesetzt ist, werden nur die projizierten Kanten zurückgegeben, die nicht durch Flächen des 3D-Objektmodells verdeckt sind. Ist HiddenSurfaceRemoval auf *'false'* gesetzt, werden alle projizierten Kanten des 3D-Objektmodells zurückgegeben. Dieser Modus ist schneller als wenn HiddenSurfaceRemoval auf *'true'* gesetzt ist.

Ist die Systemvariable (siehe set_system) *'opengl_hidden_surface_removal_enable'* auf *'true'* gesetzt (Standard, sofern es verfügbar ist) und HiddenSurfaceRemoval auf *'true'* gesetzt, wird die Modellprojektion auf

der Grafikkarte durchgeführt. Abhängig von der verwendeten Grafikkarte ist dies signifikant schneller als die analytische Projektion. Es ist zu beachten, dass die Ergebnisse der OpenGL Projektion leicht verschieden zu denen der analytischen Projektion sind. Insbesondere werden in diesem Modus nur die Kanten projiziert die im Bild der Kamera (CamParam) sichtbar sind.

Mit dem Parameter MinFaceAngle wird der minimale Winkel angegeben, den zwei benachbarte Flächen einschließen müssen, damit die Kante zwischen ihnen projiziert wird. Wird MinFaceAngle auf *0.0* gesetzt, werden alle Kanten projiziert. Wird MinFaceAngle auf π (entspricht 180 Grad) gesetzt, wird nur die Silhouette des 3D-Objektmodells zurückgegeben. Dieser Parameter kann insbesondere dafür verwendet werden, um Kanten in gekrümmten Flächen, z.B. einer Zylinder- oder Kegeloberfläche, zu unterdrücken.

Wenn bei der Erzeugung mittels create_shape_model_3d der Parameter *'union_adjacent_contours'* aktiviert wurde, dann werden benachbarte Konturen verbunden.

project_shape_model_3d und project_object_model_3d liefern das selbe Ergebnis, wenn an project_object_model_3d das 3D-Objektmodell übergeben wird, das zur Erstellung des 3D-Formmodells verwendet wurde.

project_shape_model_3d ist insbesondere dazu nützlich, mit find_shape_model_3d gefundene Matches zu visualisieren, wenn das zu Grunde liegende 3D-Objektmodell nicht mehr verfügbar ist.

Parameter

- ▷ **ModelContours** (output_object) ..... xld_cont-array ⇝ *object*
  Konturrepräsentation der Modellansicht.
- ▷ **ShapeModel3DID** (input_control) ..... shape_model_3d ⇝ *handle*
  Handle des 3D-Formmodells.
- ▷ **CamParam** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **Pose** (input_control) ..... pose ⇝ *real* / integer
  3D-Lage (Pose) des 3D-Formmodells im Weltkoordinatensystem.
- ▷ **HiddenSurfaceRemoval** (input_control) ..... string ⇝ *string*
  Verdeckte Flächen eliminieren?
  Defaultwert: 'true'
  Werteliste: HiddenSurfaceRemoval ∈ {'true', 'false'}
- ▷ **MinFaceAngle** (input_control) ..... angle.rad ⇝ *real* / integer
  Kleinster Flächenwinkel für den die Kante dargestellt wird
  Defaultwert: 0.523599
  Wertevorschläge: MinFaceAngle ∈ {0.17, 0.26, 0.35, 0.52}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert project_shape_model_3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_shape_model_3d, read_shape_model_3d, get_shape_model_3d_params, find_shape_model_3d

Alternativen

project_object_model_3d

Siehe auch

convert_point_3d_cart_to_spher, convert_point_3d_spher_to_cart, create_cam_pose_look_at_point, trans_pose_shape_model_3d

Modul

3D Metrology

read_shape_model_3d (: : FileName : ShapeModel3DID)

Einlesen eines 3D-Formmodells von Datei.

read_shape_model_3d liest ein 3D-Formmodell, das mit write_shape_model_3d geschrieben wurde, aus der Datei FileName ein. Die Dateiendung für das 3D-Formmodell ist in HALCON 'sm3'.

Parameter

- ▷ **FileName** (input_control) ..... filename.read ⇝ *string*
  Name der Datei.
  Dateiendung: .sm3
- ▷ **ShapeModel3DID** (output_control) ..... shape_model_3d ⇝ *handle*
  Handle des 3D-Formmodells.

Ergebnis

Ist der Dateiname korrekt, dann liefert read_shape_model_3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

find_shape_model_3d, get_shape_model_3d_params

Siehe auch

create_shape_model_3d, clear_shape_model_3d

Modul

3D Metrology

serialize_shape_model_3d (
: : ShapeModel3DID : SerializedItemHandle)

Serialisieren eines 3D-Formmodells.

serialize_shape_model_3d serialisiert die Daten eines 3D-Formmodells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_shape_model_3d in eine Datei geschrieben werden. Das 3D-Formmodell wird in dem Handle ShapeModel3DID übergeben. Das serialisierte 3D-Formmodell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_shape_model_3d wieder deserialisiert werden.

Parameter

- ▷ **ShapeModel3DID** (input_control) ..... shape_model_3d ⇝ *handle*
  Handle des 3D-Formmodells.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_shape_model_3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_shape_model_3d`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_shape_model_3d`

Modul

3D Metrology

`trans_pose_shape_model_3d` `( : : ShapeModel3DID, PoseIn, Transformation : PoseOut )`

Transformieren einer Pose, die sich auf das 3D-Objektmodellkoordinatensystem bezieht, in eine Pose, die sich auf das Referenzkoordinatensystem eines 3D-Formmodells bezieht und umgekehrt.

Der Operator `trans_pose_shape_model_3d` transformiert die Pose `PoseIn` in die Pose `PoseOut` unter Verwendung der in `Transformation` angegebenen Transformationsrichtung. In den meisten Fällen wird dieser Operator für die Transformation einer Kamera-Pose, die im ersten Koordinatensystem gegeben ist, in eine Kamera-Pose bezüglich des zweiten Koordinatensystems verwendet.

Die Pose kann zwischen zwei Koordinatensystemen transformiert werden. Das erste Koordinatensystem ist das Referenzkoordinatsystem (*ref*) des in `ShapeModel3DID` übergebenen 3D-Formmodells. Der Ursprung des Referenzkoordinatensystems liegt im Referenzpunkt des zu Grunde liegenden 3D-Objektmodells. Die Orientierung des Referenzkoordinatensystems ist durch die in `create_shape_model_3d` angegebene Referenzorientierung festgelegt.

Das zweite Koordinatensystem ist das Weltkoordinatensystem (*mcs*), d.h. das Koordinatensystems des 3D-Objektmodells, das dem 3D-Formmodell zu Grunde liegt. Dieses Koordinatensystem ist implizit durch die Koordinaten festgelegt, die in der mit `read_object_model_3d` eingelesenen CAD-Datei gespeichert sind.

Wenn `Transformation` auf *'ref_to_model'* gesetzt ist, wird angenommen, dass sich die 3D-Lage `PoseIn` auf das Referenzkoordinatensystem des 3D-Formmodells bezieht. Die 3D-Lage wird somit in der Form ${}^{cs}\mathbf{P}_{rcs}$ erwartet. Dabei steht *cs* für das Koordinatensystem in welches die Eingabe-3D-Lage transformiert (z.B. das Kamerakoordinatensystem). Für weitere Informationen Transformationen / Posen und „`Solution Guide III-C - 3D Vision`". Die transformierte Ausgabe-Pose bezieht sich in diesem Fall auf das Koordinatensystem des 3D-Objektmodells, sprich ${}^{cs}\mathbf{P}_{mcs}$.

Wenn `Transformation` auf *'model_to_ref'* gesetzt ist, wird angenommen, dass sich `PoseIn` auf das Koordinatensystem des 3D-Objektmodells bezieht. Die 3D-Lage ist in der Form ${}^{cs}\mathbf{P}_{rcs}$, siehe auch Transformationen / Posen und „`Solution Guide III-C - 3D Vision`". Die transformierte Ausgabe-Pose bezieht sich in diesem Fall auf das Referenzkoordinatensystem des 3D-Formmodells, sprich ${}^{cs}\mathbf{P}_{rcs}$.

Die relative 3D-Lage der beiden Koordinatensysteme zueinander kann mit dem Operator `get_shape_model_3d_params` ermittelt werden, indem man für `GenParamName` *'reference_pose'* übergibt.

Parameter

▷ **`ShapeModel3DID`** (input_control) . . . . . shape_model_3d ⇝ *handle*
Handle des 3D-Formmodells.

▷ **`PoseIn`** (input_control) . . . . . pose ⇝ *real* / integer
Zu transformierende Pose im Ausgangssystem.

▷ **`Transformation`** (input_control) . . . . . string ⇝ *string*
Richtung der Transformation.
Defaultwert: 'ref_to_model'
Werteliste: `Transformation` ∈ {'ref_to_model', 'model_to_ref'}

▷ **`PoseOut`** (output_control) . . . . . pose ⇝ *real* / integer
Transformierte Pose im Zielsystem.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `trans_pose_shape_model_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_shape_model_3d`

Alternativen

`hom_mat3d_translate`, `hom_mat3d_rotate`

Modul

3D Metrology

write_shape_model_3d (: : ShapeModel3DID, FileName :)

Schreiben eines 3D-Formmodells in eine Datei.

`write_shape_model_3d` schreibt ein 3D-Formmodell in die Datei `FileName`. Das Modell kann mit `read_shape_model_3d` wieder eingelesen werden. Die Dateiendung für das 3D-Formmodell ist in HALCON 'sm3'.

Parameter

▷ **ShapeModel3DID** (input_control) ..... shape_model_3d ⇝ *handle*
Handle des 3D-Formmodells.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.sm3`

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert `write_shape_model_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_shape_model_3d`

Modul

3D Metrology

3.4 Oberflächenbasiert

clear_surface_matching_result (: : SurfaceMatchingResultID :)

Freigabe eines Ergebnisses des oberflächenbasierten Matchings.

`clear_surface_matching_result` gibt den Speicher eines Ergebnisses des oberflächenbasierten Matchings frei, welches mit `find_surface_model` oder `refine_surface_model_pose` angelegt wurde.

Das Ergebnis kann nach dem Aufruf nicht mehr verwendet werden. Das Handle SurfaceMatchingResultID ist nach dem Aufruf ungültig.

Parameter

▷ **SurfaceMatchingResultID** (input_control) . . . . . surface_matching_result(-array) ⇝ *handle*
Ergebnishandle des oberflächenbasierten Matchings.

Ergebnis

Ist der Handle des Modells gültig, dann liefert clear_surface_matching_result den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SurfaceMatchingResultID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

find_surface_model, refine_surface_model_pose

Siehe auch

find_surface_model, refine_surface_model_pose

Modul

3D Metrology

clear_surface_model (: : SurfaceModelID :)

Freigabe des Speichers eines Oberflächenmodells.

clear_surface_model gibt den Speicher eines Oberflächenmodells, welches mit create_surface_model oder read_surface_model angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle SurfaceModelID ist nach dem Aufruf ungültig.

Parameter

▷ **SurfaceModelID** (input_control) . . . . . surface_model(-array) ⇝ *handle*
Handle des 3D-Oberflächenmodells

Ergebnis

Ist der Handle des Modells gültig, dann liefert clear_surface_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SurfaceModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`read_surface_model`, `create_surface_model`

Siehe auch

`read_surface_model`, `create_surface_model`

Modul

3D Metrology

`create_surface_model` (: : `ObjectModel3D`, `RelSamplingDistance`, `GenParamName`, `GenParamValue` : `SurfaceModelID`)

Erstellt die für oberflächenbasiertes Matching notwendigen Datenstrukturen.

`create_surface_model` erstellt aus dem 3D-Objektmodell `ObjectModel3D` ein Modell für oberflächenbasiertes Matching. Das 3D-Oberflächenmodell kann beispielsweise mittels `read_object_model_3d` aus einer Datei gelesen worden sein oder mittels `xyz_to_object_model_3d` erstellt worden sein. Das Handle des erstellten Oberflächenmodells wird in `SurfaceModelID` zurückgegeben.

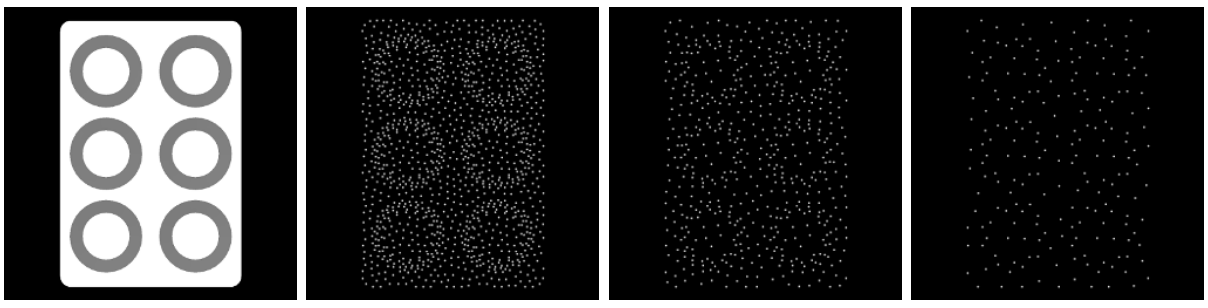
Weitere Parameter des Modells können mit `set_surface_model_param` gesetzt werden, nachdem das Modell erzeugt wurde.

Für die Erstellung des Oberflächenmodells aus dem 3D-Objektmodell werden Punkte und Normalen aus dem 3D-Objektmodell benötigt. Hier sind die folgenden Kombinationen möglich:

- Punkte und Punktnormalen;
- Punkte und ein Dreiecks- oder Polygongitter, z.B. aus einer CAD-Datei;
- Punkte und ein 2D-Mapping, z.B. für 3D-Objektmodelle, die mittels `xyz_to_object_model_3d` erstellt wurden.

Es ist zu beachten, dass die Richtung und Ausrichtung (einwärts oder auswärts) der Normalen des Modells wichtig für das Matching sind. Für das kantenunterstützte oberflächenbasierte Matching müssen die Normalen einwärts gerichtet sein und zusätzlich muss das Modell ein Dreiecks- oder Polygongitter enthalten (siehe unten).

Für das Oberflächenmodell wird die Oberfläche des 3D-Objektmodells mit einem bestimmten Abstand abgetastet. Die Abtastdistanz wird im Parameter `RelSamplingDistance` spezifiziert, und wird relativ zum Durchmesser des umschließenden achsenparallelen Quaders angegeben. Wird etwa `RelSamplingDistance` auf *0.05* gesetzt und ist der Durchmesser von `ObjectModel3D` gleich *10 cm*, dann haben die Punkte der abgetasteten Oberfläche einen Abstand von etwa *5 mm*. Die abgetasteten Punkte werden für das näherungsweise Matching in `find_surface_model` verwendet (siehe unten). Die abgetasteten Punkte können mit dem Operator `get_surface_model_param` als *'sampled_model'* abgerufen werden. Es ist zu beachten, dass einzelne Ausreißerpunkte den Durchmesser des umschließenden Quaders deutlich verändern können, was aufgrund der relativen Parametrisierung die Abtastdistanz stark stören kann. Ausreißerpunkte im 3D-Objektmodell sind daher vor dem Aufruf des Operators zu entfernen. Verringert man `RelSamplingDistance`, so werden mehr Punkte abgetastet, was zu einem stabileren aber langsameren Matching führt. Erhöht man `RelSamplingDistance`, so werden weniger Punkte abgetastet, was zu einem schnelleren aber weniger stabilen Matching führt.



(1) (2) (3) (4)

(1) Ursprüngliches 3D-Modell. (2) 3D-Modell abgetastet mit `RelSamplingDistance` = *0.02*. (3) `RelSamplingDistance` = *0.03*. (4) `RelSamplingDistance` = *0.05*.

Die abgetasteten Punkte werden für das Auffinden des Objekts in einer Szene im Operator `find_surface_model` verwendet. Dafür werden aus den abgetasteten Punkten alle möglichen Punktpaare gebildet und für jedes Paar der Abstand und die relative Orientierung der Oberflächen berechnet. Beide Werte werden diskretisiert und für das Matching gespeichert. Die generischen Parameter *'feat_step_size_rel'* und *'feat_angle_resolution'* können verwendet werden um die Diskretisierung von Distanz und Orientierung zu steuern (siehe unten).

Das 3D-Objektmodell wird ein zweites Mal für die Pose-Verfeinerung abgetastet. Diese zweite Abtastung wird mit einer kleineren Abtastdistanz durchgeführt, was zu mehr Punkten führt. Der generische Parameter *'pose_ref_rel_sampling_distance'* setzt die Abtastdistanz relativ zum Durchmesser des umschliessenden Quaders des 3D-Objektmodells fest. Verringert man den Wert so wird die Pose-Verfeinerung genauer, jedoch wird auch das Modell größer und die Modellgenerierung und das Matching langsamer. Erhöht man den Wert so wird die Pose-Verfeinerung ungenauer, das Modell kleiner und die Modellgenerierung und das Matching schneller.

Das oberflächenbasierte Matching kann zusätzlich zu den 3D-Oberflächenpunkten auch 3D-Kanten verwenden, um die Ausrichtung zu verbessern. Dies ist besonders für Objekte mit größeren planaren Seiten hilfreich, die andernfalls in einer Hintergrundebene oder falsch rotiert gefunden werden können. Um das kantenunterstützte oberflächenbasierte Matching zu verwenden, muss das Modell durch setzen des generischen Parameters *'train_3d_edges'* auf *'true'* entsprechend vorbereitet werden. Dabei muss das Modell ein Dreiecks- oder Polygongitter enthalten, bei dem die Reihenfolge der Punkte bei der Normalenberechnung zu einwärts gerichteten Normalen führt. Weiterhin benötigt das Training in diesem Fall OpenGL 2.1, GLSL 1.2 und die OpenGL Extensions GL_EXT_framebuffer_object und GL_EXT_framebuffer_blit. Es ist zu beachten, dass das Training deutlich länger dauern kann als das Training ohne Kantenunterstützung.

Zusätzlich kann das Modell für oberflächenbasiertes Matching zur Berechnung der ansichtsbasierten Bewertung vorbereitet werden. Dies ist besonders für Szenen hilfreich, in denen Objekte nur zu einem kleinen Teil sichtbar sind, da diese mit der herkömmlichen Bewertung nur einen kleinen Wert erhalten würden. Um in diesen Fällen einen höheren und aussagekräftigeren Wert zu bekommen, entspricht der Wert bei der ansichtsbasierten Bewertung dem Verhältnis zwischen der Anzahl der Punkte, die dem Modell zugeordneten werden konnten und der Anzahl an Modellpunkten die von einer bestimmten Ansicht aus sichtbar sind. Um das ansichtsbasierte oberflächenbasierte Matching zu verwenden, muss das Modell durch Setzen des generischen Parameters *'train_view_based'* auf *'true'* entsprechend vorbereitet werden. Wie bei *'train_3d_edges'*, muss das 3D-Objektmodell eine Vermaschung aufweisen, d.h. Dreiecke oder Polygone enthalten. Auch für *'train_view_based'* ist es wichtig, dass die Normalenvektoren einwärts gerichtet sind.

Es ist zu beachten, dass verrauschte Daten bei der Erzeugung eines 3D-Objektmodells zur Berechnung unzureichender Normalen führt. Insbesondere wenn das Modell für die Verwendung von 3D-Kanten oder zur Berechnung der ansichtsbasierten Bewertung vorbereitet wurde kann dies zu unzuverlässigen Bewertungen führen. Um die Anzahl verrauschter Punkte zu reduzieren können beispielsweise die Operatoren `smooth_object_model_3d` oder `simplify_object_model_3d` genutzt werden.

In den Parametern `GenParamName` und `GenParamValue` können generische Parameter und deren Werte übergeben werden. `GenParamName` enthält ein Tupel von Parameternamen die gesetzt werden sollen, und `GenParamValue` die dazugehörigen Werte. Die folgenden Werte sind für `GenParamName` möglich:

'model_invert_normals': Invertieren der Normalenausrichtung des 3D-Objektmodells. Die Normalenausrichtungen (einwärts oder auswärts) müssen für die Erstellung des Oberflächenmodells bekannt sein und mit der Ausrichtung in der Szene übereinstimmen. Wenn sowohl das Modell als auch die Szene mit demselben Aufbau aufgenommen werden, so zeigen die Normalen bereits in die gleiche Richtung. Wenn das Modell aus einer CAD-Datei geladen wurde dann könnten die Normalen in unterschiedliche Richtungen zeigen. Wenn beim Matching der Effekt auftritt, dass das Modell auf der 'Außenseite' der Szene gefunden wird und das Modell aus einer CAD-Datei stammt, so sollte versucht werden diesen Parameter auf *'true'* zu setzen. Es ist auch darauf zu achten, dass die Normalen in der CAD-Datei alle entweder nach innen oder nach außen zeigen, d.h. konsistent orientiert sind. Für die Pose-Verfeinerung ist die Normalenausrichtung irrelevant. Wird das Oberflächenmodell nur mit `refine_surface_model_pose` verwendet, so hat die Wahl von *'model_invert_normals'* keinen Einfluss auf das Ergebnis.

Mögliche Werte: 'false', 'true'

Standardwert: 'false'

'pose_ref_rel_sampling_distance': Setzen der Abtastdistanz für die Pose-Verfeinerung relativ zum Durchmesser des 3D-Objektmodells. Verringert man den Wert, so wird die Pose-Verfeinerung genauer, jedoch wird auch das Modell größer und die Modellgenerierung und das Matching langsamer. Erhöht man den Wert, so wird die Pose-Verfeinerung ungenauer, das Modell kleiner und die Modellgenerierung und das Matching schneller.

Vorgeschlagene Werte: 0.05, 0.02, 0.01, 0.005
Standardwert: 0.01
Restriktion: 0 < *'pose_ref_rel_sampling_distance'* < 1

'feat_step_size_rel': Setzen der Diskretisierung des Abstandes von Punktpaaren relativ zum Durchmesser des 3D-Objektmodells. Standardmäßig wird dies auf den Wert von `RelSamplingDistance` gesetzt. Es wird nicht empfohlen diesen Wert zu ändern. Für stark verrauschte Szenen kann der Wert erhöht werden um die Stabilität des Matchings gegen verrauschte Punkte zu erhöhen.
Vorgeschlagene Werte: 0.1, 0.05, 0.03
Standardwert: Wert von `RelSamplingDistance`
Restriktion: 0 < *'feat_step_size_rel'* < 1

'feat_angle_resolution': Setzen der Diskretisierung der Orientierung von Punktepaaren, als Anzahl der Unterteilungen eines Winkels. Es wird nicht empfohlen diesen Wert zu ändern. Erhöht man diesen Wert, so erhöht sich die Genauigkeit des näherungsweisen Matchings in `find_surface_model`, jedoch reduziert sich die Stabilität gegen verrauschte Normalenrichtungen. Verringert man diesen Wert, so verringert sich auch die Genauigkeit des näherungsweisen Matchings, jedoch erhöht sich die Stabilität gegen verrauschte Normalenrichtungen. Für stark verrauschte Szenen in denen die Normalenrichtungen nicht genau bestimmt werden können kann dieser Wert auf *25* oder *20* gesetzt werden.
Vorgeschlagene Werte: 20, 25, 30
Standardwert: 30
Restriktion: 'feat_angle_resolution' > 1

'train_3d_edges': Trainiert das Oberflächenmodell für die kantenunterstützte Objekterkennung und Posenverfeinerung. Das Modell muss in diesem Fall eine Vermaschung aufweisen, d.h. Dreiecke oder Polygone enthalten. Zusätzlich ist es wichtig, dass die Normalenvektoren einwärts gerichtet sind. Weiterhin benötigt das Training in diesem Fall OpenGL.
Mögliche Werte: 'false', 'true'
Standardwert: 'false'

'train_view_based': Trainiert das Oberflächenmodell zur Berechnung einer ansichtsbasierten Bewertung für die Objekterkennung und Posenverfeinerung. Das Modell muss in diesem Fall eine Vermaschung aufweisen, d.h. Dreiecke oder Polygone enthalten. Zusätzlich ist es wichtig, dass die Normalenvektoren einwärts gerichtet sind. Weiterhin benötigt das Training in diesem Fall OpenGL.
Mögliche Werte: 'false', 'true'
Standardwert: 'false'

'train_self_similar_poses': Bereitet das Oberflächenmodell für eine Optimierung bezüglich selbstähnlicher, fastsymmetrischer Lagen vor. Dabei werden während der Modellgenerierung Lagen des Objekts gesucht, in welchen das Objekts sehr ähnlich aussieht, d.h. Lagen, in denen sich das Objekt nur durch kleine Eigenschaften unterscheidet (z.B. die Lage von Bohrlöchern) und die von `find_surface_model` leicht verwechselt werden können. Bei der Suche wird `find_surface_model` automatisch die Beste dieser Lagen ermitteln, um eine solche Verwechslung zu vermeiden.
Mögliche Werte: 'false', 'true'
Standardwert: 'false'

Parameter

▷ **`ObjectModel3D`** (input_control) ..... object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells.

▷ **`RelSamplingDistance`** (input_control) ..... real ⇝ *real*
Abtastdistanz relativ zum Durchmesser des Objekts.
Defaultwert: 0.03
Wertevorschläge: `RelSamplingDistance` ∈ {0.1, 0.05, 0.03, 0.02, 0.01}
Restriktion: 0 < RelSamplingDistance < 1

▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamName` ∈ {'model_invert_normals', 'pose_ref_rel_sampling_distance', 'feat_step_size_rel', 'feat_angle_resolution', 'train_3d_edges', 'train_view_based', 'train_self_similar_poses'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / real / integer
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {0, 1, 'true', 'false', 0.005, 0.01, 0.02, 0.05, 0.1}

▷ **SurfaceModelID** (output_control) . . . . . surface_model ⇝ *handle*
Handle des Oberflächenmodells.

Ergebnis

Wenn alle Parameter korrekt sind liefert `create_surface_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`, `get_object_model_3d_params`, `surface_normals_object_model_3d`

Nachfolger

`find_surface_model`, `refine_surface_model_pose`, `get_surface_model_param`, `write_surface_model`, `clear_surface_model`, `set_surface_model_param`

Alternativen

`read_surface_model`

Siehe auch

`find_surface_model`, `refine_surface_model_pose`, `read_surface_model`, `write_surface_model`, `clear_surface_model`, `set_surface_model_param`

Literatur

Bertram Drost, Markus Ulrich, Nassir Navab, Slobodan Ilic: „Model Globally, Match Locally: Efficient and Robust 3D Object Recognition.“ Computer Vision and Pattern Recognition, pp. 998-1005, 2010.

Modul

3D Metrology

deserialize_surface_model (: : SerializedItemHandle : SurfaceModelID)

Deserialisieren eines Oberflächenmodells.

`deserialize_surface_model` deserialisiert ein Oberflächenmodell, welches mit `serialize_surface_model` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Oberflächenmodell wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Oberflächenmodell mit dem Handle `SurfaceModelID` gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **SurfaceModelID** (output_control) . . . . . surface_model ⇝ *handle*
Handle des gelesenen Oberflächenmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_surface_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`, `fread_serialized_item`, `receive_serialized_item`, `serialize_surface_model`

Nachfolger

`find_surface_model`, `refine_surface_model_pose`, `get_surface_model_param`, `clear_surface_model`, `find_surface_model_image`, `refine_surface_model_pose_image`

Alternativen

`create_surface_model`

Siehe auch

`create_surface_model`, `read_surface_model`, `write_surface_model`

Modul

3D Metrology

```
find_surface_model ( : : SurfaceModelID, ObjectModel3D,
    RelSamplingDistance, KeyPointFraction, MinScore,
    ReturnResultHandle, GenParamName, GenParamValue : Pose, Score,
    SurfaceMatchingResultID )
```

Suche der besten Matches eines Oberflächenmodells in einer Szene.

Der Operator `find_surface_model` sucht die besten Matches des Oberflächenmodells `SurfaceModelID` in der 3D-Szene `ObjectModel3D` und gibt ihre Posen in `Pose` zurück.

Der Matching-Vorgang ist in drei Schritte unterteilt:

1. `Näherungsweises Matching`
2. `Grobe Pose-Verfeinerung`
3. `Exakte Pose-Verfeinerung`

Die einzelnen Schritte werden detailliert in der Technical Note `Surface-Based Matching` erläutert. Die generischen Parameter die zur Steuerung dieser Schritte verwendet werden können werden in den entsprechenden Abschnitten weiter unten beschrieben. In den weiteren Abschnitten werden die Parameter beschrieben und zusätzlich zu beachtenden Punkte angegeben.

Die HDevelop-Prozedur *debug_find_surface_model* kann verwendet werden, um den Matchingprozess und die eingestellten Parameter zu visualisieren und zu überprüfen.

Zu beachtende Punkte

Zum Auffinden des Oberflächenmodells werden Punkte und Normalen der 3D-Szene verwendet. Die Szene sollte eine der folgenden Optionen zur Verfügung stellen:

- Punkte und Punktnormalen.
- Punkte und ein 2D-Mapping, z.B. für 3D-Objektmodelle die mittels `xyz_to_object_model_3d` erstellt wurden. In diesem Fall werden die Normalen mit dem 2D-Mapping berechnet.
- Nur Punkte. Die Normalen werden aufgrund der 3D Umgebung abgeschätzt. Es gilt zu beachten, dass diese Option nicht empfohlen wird, da sie generell zu längeren Verarbeitungszeiten führt und die berechneten Normalen üblicherweise weniger genau sind, was in ungenaueren Ergebnissen resultiert.

Für eine genaue `Pose` ist es wichtig, dass die Normalen der Szene und des Modells in die gleiche Richtung zeigen (siehe *'scene_invert_normals'*).

Wenn das Oberflächenmodell für kantenunterstütztes oberflächenbasiertes Matching vorbereitet wurde und das kantenunterstützte Matching nicht durch *'use_3d_edges'* abgeschaltet wurde, ist lediglich die zweite Kombination möglich, d.h. die Szene muss ein 2D-Mapping beinhalten.

Wenn das Oberflächenmodell für kantenunterstütztes oberflächenbasiertes Matching vorbereitet wurde und die Szene ein 2D-Mapping beinhaltet, werden Punktnormalen aus der Szene nicht benutzt (siehe *'scene_normal_computation'* unten).

Für Oberflächenmodelle, die für kantenunterstütztes oberflächenbasiertes Matching vorbereitet wurden, ist es notwendig, dass die Normalen einwärts ausgerichtet sind.

Wird eine vermaschte Szene mit Dreiecken oder Polygonen übergeben, etwa ein CAD-Modell, so werden diese ignoriert und nur die Punkte der Szene verwendet. In der Regel sollte daher nicht in einer vermaschten Szene gesucht werden. Stattdessen sollte eine solche Szene zuvor etwa mit `sample_object_model_3d` in Punkte mit Normalen umgewandelt werden.

Werden verrauschte Daten verwendet, z.B. von Time-of-Flight-Kameras, kann der generische Parameter *'scene_normal_computation'* auf *'mls'* gesetzt werden um robustere Ergebnisse zu erhalten (siehe unten).

Parameterbeschreibung

`SurfaceModelID` ist das Handle des Oberflächenmodells. Das Modell muss zuvor mittels `create_surface_model` oder `read_surface_model` erstellt, bzw eingelesen worden sein. Gewisse Oberflächenmodell-Parameter, welche das Auffinden beeinflussen, können über den Operator `set_surface_model_param` gesetzt werden, z.B. *'pose_restriction_max_angle_diff'* zum Einschränken des Rotationsbereichs.

`ObjectModel3D` ist das Handle des 3D-Objektmodells mit der Szene in der die Matches gesucht werden. In den meisten Fällen wird angenommen, dass die Kamera entlang der z-Achse ausgerichtet ist. Dies ist wichtig bei der Ausrichtung der Normalen der Szene, falls diese neu berechnet werden (siehe *'scene_normal_computation'* unten). Falls das Oberflächenmodell für kantenunterstütztes oberflächenbasiertes Matching vorbereitet wurde und die Szene ein 2D-Mapping beinhaltet, werden die Normalen der Szene automatisch konsistent ausgerichtet.

Der Parameter `RelSamplingDistance` kontrolliert die Abtastdistanz während des Schrittes `Näherungsweises Matching` und bei der Berechnung des `Score` im Schritt `Grobe Pose-Verfeinerung`. Sein Wert wird relativ zum Durchmesser des Oberflächenmodells angegeben. Verkleinert man `RelSamplingDistance`, so werden mehr Punkte der Szene abgetastet, was zu einem stabileren aber langsameren Matching führt. Erhöht man `RelSamplingDistance`, so werden weniger Punkte abgetastet, was zu einem weniger stabilen aber schnelleren Matching führt. Eine Illustration zur `RelSamplingDistance` befindet sich bei dem Operator `create_surface_model`. Die abgetasteten Punkte können mit dem Operator `get_surface_matching_result` abgerufen werden. Für ein robustes Matching sollten mindestens 50-100 Punkte auf jeder zu findenden Objektinstanz liegen.

Der Parameter `KeyPointFraction` kontrolliert wie viele der abgetasteten Szenenpunkte als Schlüsselpunkte ausgewählt werden. Wird dieser Wert etwa auf *0.1* gesetzt, so werden 10% der abgetasteten Szenepunkte als Schlüsselpunkte verwendet. Für robuste Ergebnisse ist es wichtig, dass auf jeder zu findenden Objektinstanz mehrere Schlüsselpunkte liegen. Erhöht man `KeyPointFraction`, so werden mehr Punkte als Schlüsselpunkte ausgewählt, was zu einem robusteren aber langsameren Matching führt. Verringert man den Wert so werden weniger Schlüsselpunkte ausgewählt, was zu einem weniger robusten aber schnelleren Matching führt. Für eine visuelle Überprüfung können die ausgewählten Schlüsselpunkte mit dem Operator `get_surface_matching_result` abgerufen werden.

Der Parameter `MinScore` kann verwendet werden um die Ergebnisse zu filtern. Nur Instanzen deren Bewertung größer als `MinScore` ist werden zurückgegeben. Wird `MinScore` auf *0* gesetzt, so werden alle gefundenen Instanzen zurückgegeben.

Für das kantenunterstützte oberflächenbasierte Matching (siehe `create_surface_model`) werden vier Einzelbewertungen ermittelt (siehe weiter unten für ihre Beschreibung). Für Modelle die zur Berechnung einer ansichtsbasierten Bewertung trainiert wurden (siehe `create_surface_model`), wird eine weitere, fünfte Einzelbewertungen ermittelt. Entsprechend können die Resultate mit jeder dieser Bewertungen gefiltert werden, indem `MinScore` ein Tupel mit bis zu fünf Schwellenwerten übergeben wird. Diese Schwellenwerte sind in der gleichen Reihenfolge wie die Bewertungen geordnet (siehe weiter unten) und fehlende Einträge werden als *0* betrachtet, was bedeutet, dass keine Ergebnisse aufgrund dieser Bewertung herausgefiltert wird. Zum Finden geeigneter Schwellenwerte können die Einzelbewertungen gefundener Posen mit dem Operator `get_surface_matching_result` erhalten werden. Je nach Einstellung des Verfahrens sind möglicherweise

nicht alle Bewertungen verfügbar. Für nicht verfügbare Bewertungen wird der in MinScore eingestellte Schwellenwert ignoriert. Die fünf Einzelbewertungen, deren Schwellenwerte in dieser Reihenfolge in MinScore übergeben werden müssen, sind:

1. Die Gesamtbewertung wie in Score und durch *'score'* in get_surface_matching_result zurückgegeben.
2. Der Oberflächenanteil der Ergebnisbewertung, d.h. welcher Anteil der Objektoberfläche in der Szene gefunden wurde, zurückgegeben durch *'score_surface'* in get_surface_matching_result.
3. Der Anteil der 3D-Kanten der Ergebnisbewertung, d.h. wie gut die 3D-Kanten des Objekts mit den in der Szene gefundenen 3D-Kanten übereinstimmen, zurückgegeben durch *'score_3d_edges'* in get_surface_matching_result.
4. Der Anteil der 2D-Kanten der Ergebnisbewertung, d.h. wie gut die projizierten Objektsilhouetten mit den in den Bildern gefundenen Kanten übereinstimmen (nur für die Operatoren find_surface_model_image und refine_surface_model_pose_image verfügbar), zurückgegeben durch *'score_2d_edges'* in get_surface_matching_result.
5. Der ansichtsbasierte Anteil der Ergebnisbewertung, d.h. wie viele Punkte der Szene einem Modellpunkt zugeordnet werden konnten im Verhältnis zu den Modellpunkten, die in dieser Ansicht sichtbar sind, zurückgegeben durch *'score_view_based'* in get_surface_matching_result.

Der Parameter ReturnResultHandle bestimmt, ob ein Handle für das Ergebnis des oberflächenbasierten Matchings zurückgegeben werden soll oder nicht. Falls der Parameter den Wert *'true'* erhält, wird das Handle unter dem Parameter SurfaceMatchingResultID zurückgegeben. Weitere Details des Matchingverfahrens können dann mittels get_surface_matching_result abgerufen werden.

Die Parameter GenParamName und GenParamValue werden zum Setzen generischer Parameter verwendet. Beide erhalten ein Tupel gleicher Länge, wobei das Tupel von GenParamName die Namen der zu setzenden Parameter enthält und das Tupel GenParamValue die entsprechenden Werte. Die möglichen Parameternamen und Parameterwerte sind im Abschnitt Die drei Schritte des Matching-Vorgangs beschrieben.

Der Ausgabeparameter Pose gibt die Pose der gefundenen Instanzen zurück. Er beschreibt die Pose des Oberflächenmodells im Szenenkoordinatensystem (*scs*) und bezieht sich auf das Modellkoordinatensystem des 3D-Objektmodells (*mcs*), welches für create_surface_model verwendet wurde. Somit ist die Pose in der Form ${}^{scs}\mathbf{P}_{mcs}$ (siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision").

Der Ausgabeparameter Score gibt für jede übereinstimmende Instanz eine Bewertung zurück. Der Wert und die Interpretation der Bewertung sind, wie im folgenden Abschnitt erläutert, für verschiedene Fälle unterschiedlich.

- **Mit Pose-Verfeinerung**
  Bei einem Matching mit Pose-Verfeinerung hängt die Bewertung davon ab, ob das Matching mit Kantenunterstützung durchgeführt wurde:
  - Ohne Kantenunterstützung wird für die Bewertung der Oberflächenanteil berechnet, d.h. der ungefähre Anteil der in der Szene sichtbaren Fläche des Objekts. Dies erfolgt durch Zählen der gefundenen Modellpunkte mit entsprechendem Szenepunkt (wie bei *'num_model_points'*) und dividieren dieser Anzahl entweder durch:
    * die Gesamtzahl abgetasteter Modellpunkte, falls das Modell nicht für eine ansichtsbasierte Bewertung trainiert worden ist.
      oder durch
    * die Anzahl aller Modellpunkte die von einem einzigen Blickpunkt aus maximal sichtbar sein können, falls das Modell für eine ansichtsbasierte Bewertung trainiert worden ist

    $0 \leq Score \leq 1$
  - Mit Kantenunterstützung ist die Bewertung das geometrische Mittel des Oberflächenanteils und des Kantenanteils. Der Kantenanteil wird davon beeinflusst, ob das Modell für eine ansichtsbasierte Bewertung trainiert worden ist oder nicht. Ermittelt wird dieser aus der Anzahl an abgetasteten Kantenpunkten des Modells die mit Kanten der Szene ausgerichtet sind, geteilt durch die maximale Anzahl an Modellkantenpunkten, die von einem einzigen Blickpunkt aus sichtbar sein können. Wenn die 3D-Kanten aus mehreren Sichtrichtungen extrahiert wurden, kann dies zu einem Score größer 1 führen.
    $0 \leq Score \leq 1$ (wenn die Szene von einem einzigen Blickpunkt aus gesehen wurde)
    $0 \leq Score \leq N$ (wenn die Gesamtszene aus Einzelszenen zusammengesetzt wurde, die von insgesamt N verschiedenen Blickpunkten gesehen wurde)

Es ist zu beachten, dass für die Berechnung der Bewertung nach der groben Pose-Verfeinerung die abgetasteten Szenepunkte verwendet werden, während für die Berechnung nach der exakten Pose-Verfeinerung alle Szenepunkte verwendet werden. Die Bewertung nach der exakten Pose-Verfeinerung ist daher unabhängig von der Abtastdistanz der Szene.

- **Ohne Pose-Verfeinerung**
  Falls von den drei Schritten des Matching-Vorgangs nur der erste, `Näherungsweises Matching`, ausgeführt wird, hängen der mögliche Wert und die Interpretation der Bewertung nur von einer möglichen Kantenunterstützung ab:
  - Ohne Kantenunterstützung
    Die Bewertung ist die ungefähre Anzahl an Szenenpunkten der abgetasteten Szene die auf dem gefundenen Objekt liegen.
    $Score \geq 0$
  - Mit Kantenunterstützung
    Die Bewertung ist die ungefähre Anzahl an Szenenpunkten der abgetasteten Szene die auf dem gefundenen Objekt liegen, multipliziert mit der Anzahl an Punkten der abgetasteten 3D-Kanten die mit Kanten des Modells ausgerichtet sind.
    $Score \geq 0$

Der Ausgabeparameter `SurfaceMatchingResultID` gibt ein Ergebnishandle für das Oberflächenmatching zurück. Mittels dieses Handles können weitere Details des Matchingverfahrens unter Verwendung des Operators `get_surface_matching_result` abgerufen werden. Das Handle wird nur zurückgegeben, wenn `ReturnResultHandle` auf *'true'* gesetzt ist.

Die drei Schritte des Matching-Vorgangs

Der Matching-Vorgang ist in drei Schritte unterteilt:

1. **Näherungsweises Matching** Die ungefähren Posen der Instanzen des Oberflächenmodells in der Szene werden gesucht.
   Die folgenden generischen Parameter steuern das näherungsweise Matching, und werden mit `GenParamName` und `GenParamValue` gesetzt:

   'num_matches': Setzt die maximale Anzahl an gefundenen Instanzen die zurückgegeben werden.
   Vorgeschlagene Werte: 1, 2, 5
   Standardwert: 1
   Restriktion: 'num_matches' > 0

   'max_overlap_dist_rel': Aus Laufzeitgründen kann die maximale Überlappung zweier Matches nicht in 3D definiert werden. Stattdessen kann mit *'max_overlap_dist_rel'* der minimale Abstand zwischen den Mittelpunkten der kleinsten achsparallelen, die Matches umschließenden Quader angegeben werden. Der Wert wird relativ zum Objektdurchmesser angegeben. Wenn ein Objekt mit einem hohen `Score` gefunden wurde, werden alle anderen Matches unterdrückt, für die der Mittelpunkt ihres kleinsten umschließenden Quaders zu nahe am Mittelpunkt des kleinsten umschließenden Quaders des ersten Matches liegt. Wenn sich die gefundenen Matches nicht überlappen sollen, muss der Wert für *'max_overlap_dist_rel'* auf *1.0* gesetzt werden. Nur einer der beiden Parameter *'max_overlap_dist_rel'* und *'max_overlap_dist_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
   Vorgeschlagene Werte: 0.1, 0.5, 1
   Standardwert: 0.5
   Restriktion: 'max_overlap_dist_rel' >= 0

   'max_overlap_dist_abs': Dieser Parameter hat den gleichen Effekt wie der Parameter *'max_overlap_dist_rel'*. Allerdings wird der Wert im Gegensatz zum Parameter *'max_overlap_dist_rel'* bei *'max_overlap_dist_abs'* als absoluter Wert angegeben. Eine Beschreibung des Effekts dieses Parameters ist weiter oben bei dem Parameter *'max_overlap_dist_rel'* zu finden. Nur einer der beiden Parameter *'max_overlap_dist_rel'* und *'max_overlap_dist_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
   Vorgeschlagene Werte: 1, 2, 3
   Restriktion: 'max_overlap_dist_abs' >= 0

   'scene_normal_computation': Dieser Parameter kontrolliert die Normalenberechnung der abgetasteten 3D-Szene.

Im Standard-Modus *'fast'* werden in den meisten Fällen die Normalen der 3D-Szene benutzt (wenn sie bereits Normalen enthält) oder anhand einer relativ kleinen Nachbarschaft berechnet (wenn nicht). Die berechneten Normalen n werden dann so ausgerichtet, dass $n_z \geq 0$ falls keine Normalen in der Szene vorhanden sind. Die Ausrichtung $n_z \geq 0$ stammt von der Annahme, dass die Kamera entlang der z-Achse ausgerichtet ist.
Im Standard-Modus *'fast'*, im Fall dass das Oberflächenmodell für kantenunterstütztes oberflächenbasiertes Matching vorbereitet wurde und die Szene ein 2D-Mapping beinhaltet, werden Punktnormalen aus der Szene nicht benutzt, sondern immer anhand des 2D-Mapping berechnet. Außerdem werden die berechneten Normalen automatisch konsistent ausgerichtet.
Im Modus *'mls'* wird eine größere Nachbarschaft einbezogen und das komplexere, aber oft genauere *'mls'*-Verfahren für die Normalenberechnung verwendet. Eine genauere Beschreibung des *'mls'*-Verfahrens findet sich beim Operator `surface_normals_object_model_3d`. Der *'mls'*-Modus ist vor allem für Daten mit hohem Rauschen gedacht, etwa für Bilder aus Time-of-Flight-Kameras. Die berechneten Normalen werden wie im *'fast'*-Modus ausgerichtet.
Mögliche Werte: 'fast', 'mls'
Standardwert: 'fast'

'scene_invert_normals': Dieser Parameter invertiert die Normalenausrichtung der 3D-Szene. Die Normalenausrichtungen (einwärts oder auswärts) der Szene müssen mit den Ausrichtungen des Oberflächenmodells übereinstimmen. Wenn sowohl das Modell als auch die Szene mit demselben Aufbau aufgenommen werden, so zeigen die Normalen bereits in die gleiche Richtung. Wenn beim Matching der Effekt auftritt, dass das Modell auf der 'Außenseite' der Szene gefunden wird, so sollte versucht werden, diesen Parameter auf *'true'* zu setzen. Es ist auch darauf zu achten, dass die Normalen in der Szene alle entweder nach innen oder nach außen zeigen, d.h. konsistent orientiert sind. Für kantenunterstütztes oberflächenbasiertes Matching müssen die Normalen einwärts ausgerichtet sein, werden aber typischerweise automatisch konsistent zum 2D-Mapping ausgerichtet. Die Ausrichtung der Normalen kann mit der Prozedur `debug_find_surface_model` überprüft werden.
Mögliche Werte: 'false', 'true'
Standardwert: 'false'

'3d_edges': Erlaubt das manuelle Setzen der 3D-Kanten der Szene für das kantenunterstützte oberflächenbasierte Matching, wenn das Oberflächenmodell für kantenunterstütztes oberflächenbasiertes Matching vorbereitet wurde. Der Wert dieses Parameters muss ein 3D-Objektmodell-Handle sein. Die Kanten sind üblicherweise das Ergebnis des Operators `edges_object_model_3d`, können aber weiter gefiltert werden um Ausreißer zu entfernen. Wird dieser Parameter nicht angegeben, dann extrahiert `find_surface_model` die Kanten intern, analog zum Operator `edges_object_model_3d`.

'3d_edge_min_amplitude_rel': Setzt den Amplitudenschwellenwert bei der automatischen Extraktion von 3D-Kanten für das kantenunterstützte oberflächenbasierte Matching, d.h. wenn das Oberflächenmodell mit aktiviertem *'train_3d_edges'* erzeugt wurde und keine Kanten mittels *'3d_edges'* manuell übergeben wurden. Der Schwellenwert wird relativ zum Durchmesser des Modells angegeben und verhält sich analog zum Parameter `MinAmplitude` des Operators `edges_object_model_3d`.
Vorgeschlagene Werte: 0.05, 0.1, 0.5
Standardwert: 0.05
Restriktion: '3d_edge_min_amplitude_rel' >= 0

'3d_edge_min_amplitude_abs': Analog zu *'3d_edge_min_amplitude_rel'*, allerdings wird der Schwellenwert nicht relativ zum Modelldurchmesser sondern als absoluter Wert angegeben.
Restriktion: '3d_edge_min_amplitude_abs' >= 0

'viewpoint': Dieser Parameter gibt die 3D-Position des Blickpunktes an, von dem aus die 3D-Punkte in `ObjectModel3D` gesehen werden. Der Blickpunkt wird bei Modellen mit ansichtsbasierter Bewertung benutzt (d.h. mit aktiviertem *'train_view_based'*), um die Anzahl der Punkte des Modells abzufragen die von diesem Blickpunkt aus sichtbar sind. Die 3D-Koordinaten des Blickpunktes (x, y und z) werden als Strings, getrennt durch Leerzeichen, übergeben. Der Blickpunkt ist im gleichen Koordinatensystem wie `ObjectModel3D` definiert und sollte in etwa der Sensorposition bei der Aufnahme der 3D-Szene entsprechen. Mit der Prozedur `debug_find_surface_model` kann die Lage des Blickpunktes visualisiert und überprüft werden.
Standardwert: '0 0 0'

'max_gap': Dieser Parameter gibt die maximale Größe einer Lücke in Pixeln in einer 3D-Szene an. Dies gilt nur sofern das Oberflächenmodell mit aktiviertem *'train_3d_edges'* erstellt wurde. Für Lücken, die einen größeren Durchmesser haben als *'max_gap'*, werden Kanten an den Rändern der Lücke erzeugt, kleinere Lücken werden geschlossen. Dadurch können Kanten unterdrückt werden, welche an Lücken

in den 3D-Daten entstehen, die der Sensor nicht rekonstruieren konnte. Für besonders hochauflösende Sensoren sollte der Wert erhöht werden.
Werden Kanten manuell mit dem generischen Parameter *'3d_edges'* übergeben, so ist zu beachten, dass *'max_gap'* in diesem Fall ignoriert wird. Andernfalls verhält sich *'max_gap'* analog zum entsprechenden generischen Parameter des Operators `edges_object_model_3d`.
Inwiefern sich *'max_gap'* auf das Modell auswirkt kann mit Hilfe der Prozedur `debug_find_surface_model` visualisiert werden.
Standardwert: 30

'use_3d_edges': Schaltet die Kantenunterstützung ein oder aus. Er kann verwendet werden, um ein Matching ohne Kantenunterstützung durchzuführen, obwohl das Modell mit Kantenunterstützung erstellt wurde. Wenn das Oberflächenmodell nicht mit aktiviertem *'train_3d_edges'* erstellt wurde, wird ein Fehler zurückgegeben.
Mögliche Werte: 'true', 'false'
Standardwert: 'true'

2. Grobe Pose-Verfeinerung In diesem zweiten Schritt werden die im vorherigen Schritt gefundenen näherungsweisen Posen verfeinert. Dies erhöht die Genauigkeit der Posen sowie die Aussagekraft der Bewertung.

Die folgenden generischen Parameter steuern die grobe Pose-Verfeinerung und werden mit `GenParamName` und `GenParamValue` gesetzt:

'sparse_pose_refinement': Aktiviert oder deaktiviert die grobe Pose-Verfeinerung.
Mögliche Werte: 'true', 'false'
Standardwert: 'true'

'pose_ref_use_scene_normals': Aktiviert oder deaktiviert die Verwendung von Szenennormalen für die Pose-Verfeinerung. Wenn dieser Parameter aktiviert wird und die Szene Normalenvektoren enthält, dann werden diese Normalenvektoren verwendet um die Genauigkeit der Pose-Verfeinerung zu steigern. Dazu wird der Einfluss derer Szenenpunkte verringert, deren Normalenvektor in eine andere Richtung als der entsprechende Objektnormalenvektor zeigt. Enthält die Szene keine Punktnormalen, so wird dieser Parameter ignoriert.
Mögliche Werte: 'true', 'false'
Standardwert: 'false'

'use_view_based': Schaltet die ansichtsbasierte Bewertung beim oberflächenbasierten Matching ein oder aus. Dies ist vor allem dann sinnvoll, wenn ein Matching ohne Verwendung der ansichtsbasierten Bewertung durchgeführt wird, obwohl das Modell speziell dafür trainiert wurde. Der Einfluss von *'use_view_based'* auf die Bewertung ist in der Beschreibung zu `Score` weiter oben erklärt. Wenn das Oberflächenmodell nicht mit aktiviertem *'train_view_based'* erstellt wurde, wird bei aktiviertem *'use_view_based'* ein Fehler zurückgegeben.
Mögliche Werte: 'true', 'false'
Standardwert: 'false', falls *'train_view_based'* bei Modellerstellung deaktiviert, andernfalls *'true'*.

3. Exakte Pose-Verfeinerung Verfeinert die in den vorherigen Schritten gefundenen Posen.

Die folgenden generischen Parameter steuern die exakte Pose-Verfeinerung und beeinflussen ihre Genauigkeit und Geschwindigkeit. Die Werte werden über `GenParamName` und `GenParamValue` gesetzt:

'dense_pose_refinement': Aktiviert oder deaktiviert die exakte Pose-Verfeinerung.
Mögliche Werte: 'true', 'false'
Standardwert: 'true'

'pose_ref_num_steps': Anzahl der Iterationen der exakten Pose-Verfeinerung. Erhöht man diesen Wert, so wird die Verfeinerung genauer aber langsamer. Verringert man den Wert, so wird sie entsprechend ungenauer und schneller. Ab einer gewissen Anzahl an Iterationen konvergiert das Verfahren, so dass eine weitere Erhöhung keinen Effekt mehr auf die Genauigkeit hat.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist.
Vorgeschlagene Werte: 1, 3, 5, 20
Standardwert: 5
Restriktion: 'pose_ref_num_steps' > 0

'pose_ref_sub_sampling': Setzt, wie viele der Szenepunkte für die exakte Pose-Verfeinerung verwendet werden. Wird dieser Wert z.B. auf *5* gesetzt, so wird jeder fünfte Punkt der Szene verwendet. Dieser Parameter erlaubt einen einfachen Kompromiss zwischen Genauigkeit und Geschwindigkeit der Verfeinerung. Erhöht man den Wert, so werden weniger Punkte verwendet, was zu einer schnelleren aber weniger genauen Verfeinerung führt. Eine Verringerung des Wertes hat den umgekehrten Effekt.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist.

Vorgeschlagene Werte: 1, *2*, *5*, *10*
Standardwert: 2
Restriktion: 'pose_ref_sub_sampling' > 0

'pose_ref_dist_threshold_rel': Setzt den Abstandsschwellenwert für die exakte Pose-Verfeinerung relativ zum Durchmesser des Modells. Nur Punkte deren Abstand zur Objektinstanz diesen Schwellenwert unterschreiten gehen in die Pose-Verfeinerung ein. Punkte, die weiter von der Instanz entfernt sind, werden ignoriert. Nur einer der beiden Parameter *'pose_ref_dist_threshold_rel'* und *'pose_ref_dist_threshold_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist.
Vorgeschlagene Werte: 0.03, *0.05*, *0.1*, *0.2*
Standardwert: 0.1
Restriktion: 'pose_ref_dist_threshold_rel' > 0

'pose_ref_dist_threshold_abs': Setzt den Abstandsschwellenwert für die exakte Pose-Verfeinerung als absoluten Wert. Für Details siehe den Parameter *'pose_ref_dist_threshold_rel'*. Nur einer der beiden Parameter *'pose_ref_dist_threshold_rel'* und *'pose_ref_dist_threshold_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist.
Restriktion: 'pose_ref_dist_threshold_abs' > 0

'pose_ref_scoring_dist_rel': Setzt den Abstandsschwellenwert für die Bewertung der Pose relativ zum Durchmesser des Modells. Für Details siehe den Parameter *'pose_ref_scoring_dist_abs'*. Nur einer der beiden Parameter *'pose_ref_scoring_dist_rel'* und *'pose_ref_scoring_dist_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist.
Vorgeschlagene Werte: 0.2, *0.01*, *0.005*, *0.0001*
Standardwert: 0.005
Restriktion: 'pose_ref_scoring_dist_rel' > 0

'pose_ref_scoring_dist_abs': Setzt den Abstandsschwellenwert für die Bewertung der Pose als absoluten Wert. Nur Szenepunkte deren Abstand zur gefundenen Objektinstanz diesen Schwellenwert unterschreiten gehen in die Berechnung der Bewertung ein und werden als 'auf dem Objekt liegend' klassifiziert. Alle anderen Szenepunkte werden ignoriert. Der Schwellenwert sollte entsprechend dem zu erwartenden Rauschen und der Genauigkeit der Pose-Verfeinerung gesetzt werden. Nur einer der beiden Parameter *'pose_ref_scoring_dist_rel'* und *'pose_ref_scoring_dist_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist.

'pose_ref_use_scene_normals': Aktiviert oder deaktiviert die Verwendung von Szenennormalen für die Pose-Verfeinerung. Eine genauere Beschreibung dieses Parameters findet sich weiter oben im Abschnitt `Grope Pose-Verfeinerung`.
Mögliche Werte: 'true', *'false'*
Standardwert: 'false'

'pose_ref_dist_threshold_edges_rel': Setzt den Abstandsschwellenwert von Kanten für die exakte Pose-Verfeinerung relativ zum Durchmesser des Modells. Nur Kanten deren Abstand zur Objektinstanz diesen Schwellenwert unterschreiten gehen in die Pose-Verfeinerung ein. Kanten, die weiter von der Instanz entfernt sind, werden ignoriert. Nur einer der beiden Parameter *'pose_ref_dist_threshold_edges_rel'* und *'pose_ref_dist_threshold_edges_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist oder wenn kein kantenunterstütztes Matching durchgeführt wird.
Vorgeschlagene Werte: 0.03, *0.05*, *0.1*, *0.2*
Standardwert: 0.1
Restriktion: 'pose_ref_dist_threshold_edges_rel' > 0

'pose_ref_dist_threshold_edges_abs': Setzt den Abstandsschwellenwert von Kanten für die exakte Pose-Verfeinerung als absoluten Wert. Für Details siehe den Parameter *'pose_ref_dist_threshold_edges_rel'*. Nur einer der beiden Parameter *'pose_ref_dist_threshold_edges_rel'* und *'pose_ref_dist_threshold_edges_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist oder wenn kein kantenunterstütztes Matching durchgeführt wird.

Restriktion: 'pose_ref_dist_threshold_edges_abs' > 0

'pose_ref_scoring_dist_edges_rel': Setzt den Abstandsschwellenwert von Kanten für die Bewertung der Pose relativ zum Durchmesser des Modells. Für Details siehe den Parameter *'pose_ref_scoring_dist_edges_abs'*. Nur einer der beiden Parameter *'pose_ref_scoring_dist_edges_rel'* und *'pose_ref_scoring_dist_edges_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist oder wenn kein kantenunterstütztes Matching durchgeführt wird.
Vorgeschlagene Werte: 0.2, *0.01*, *0.005*, *0.0001*
Standardwert: 0.005
Restriktion: 'pose_ref_scoring_dist_edges_rel' > 0

'pose_ref_scoring_dist_edges_abs': Setzt den Abstandsschwellenwert von Kanten für die Bewertung der Pose als absoluten Wert. Nur Szenenkanten deren Abstand zu den Kanten der gefundenen Objektinstanz diesen Schwellenwert unterschreiten gehen in die Berechnung der Bewertung ein und werden als 'auf dem Objekt liegend' klassifiziert. Alle anderen Szenenkanten werden ignoriert. Der Schwellenwert sollte entsprechend der zu erwartenden Genauigkeit der extrahierten Kanten und der Genauigkeit der Pose-Verfeinerung gesetzt werden. Nur einer der beiden Parameter *'pose_ref_scoring_dist_edges_rel'* und *'pose_ref_scoring_dist_edges_abs'* sollte gesetzt werden. Werden beide gesetzt, so wird nur der zuletzt gesetzte Wert verwendet.
Dieser Parameter wird ignoriert, wenn die exakte Pose-Verfeinerung deaktiviert ist oder wenn kein kantenunterstütztes Matching durchgeführt wird.
Restriktion: 'pose_ref_scoring_dist_edges_abs' > 0

'use_view_based': Schaltet die ansichtsbasierte Bewertung beim oberflächenbasierten Matching ein oder aus. Für weitere Informationen siehe die entsprechende Beschreibung im Abschnitt zur groben Pose-Verfeinerung.
Wenn das Oberflächenmodell nicht mit aktiviertem *'train_view_based'* erstellt wurde, wird bei aktiviertem *'use_view_based'* ein Fehler zurückgegeben.
Mögliche Werte: 'true', *'false'*
Standardwert: 'false', falls *'train_view_based'* bei Modellerstellung deaktiviert, andernfalls *'true'*.

'use_self_similar_poses': Schaltet die Optimierung bezüglich selbstähnlicher, fast-symmetrischer Posen ein oder aus.
Wenn das Oberflächenmodell nicht mit aktiviertem *'train_self_similarities'* erstellt wurde, wird bei aktiviertem *'use_self_similar_poses'* ein Fehler zurückgegeben.
Mögliche Werte: 'true', *'false'*
Standardwert: 'false', falls *'train_self_similarities'* bei Modellerstellung deaktiviert, andernfalls *'true'*.

Parameter

▷ **SurfaceModelID** (input_control) . . . . . surface_model ⇝ *handle*
Handle des Oberflächenmodells.

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells, welches die Szene enthält.

▷ **RelSamplingDistance** (input_control) . . . . . real ⇝ *real*
Abtastdistanz der Szene relativ zum Durchmesser des Oberflächenmodells.
Defaultwert: 0.05
Wertevorschläge: `RelSamplingDistance` ∈ {0.1, 0.07, 0.05, 0.04, 0.03}
Restriktion: 0 < RelSamplingDistance < 1

▷ **KeyPointFraction** (input_control) . . . . . real ⇝ *real*
Anteil der abgetasteten Szenepunkte, die als Schlüsselpunkte verwendet werden.
Defaultwert: 0.2
Wertevorschläge: `KeyPointFraction` ∈ {0.3, 0.2, 0.1, 0.05}
Restriktion: 0 < KeyPointFraction <= 1

▷ **MinScore** (input_control) . . . . . real(-array) ⇝ *real* / integer
Mindestbewertung der zurückgegebenen Posen.
Defaultwert: 0
Restriktion: MinScore >= 0

▷ **ReturnResultHandle** (input_control) . . . . . string ⇝ *string*
Aktiviert die Rückgabe eines Ergebnishandels in SurfaceMatchingResultID.
Defaultwert: 'false'
Wertevorschläge: ReturnResultHandle ∈ {'true', 'false'}

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'num_matches', 'max_overlap_dist_rel', 'max_overlap_dist_abs', 'sparse_pose_refinement', 'dense_pose_refinement', 'pose_ref_num_steps', 'pose_ref_sub_sampling', 'pose_ref_dist_threshold_rel', 'pose_ref_dist_threshold_abs', 'pose_ref_scoring_dist_rel', 'pose_ref_scoring_dist_abs', 'pose_ref_use_scene_normals', 'scene_normal_computation', 'scene_invert_normals', '3d_edge_min_amplitude_rel', '3d_edge_min_amplitude_abs', 'viewpoint', 'max_gap', '3d_edges', 'pose_ref_dist_threshold_edges_rel', 'pose_ref_dist_threshold_edges_abs', 'pose_ref_scoring_dist_edges_rel', 'pose_ref_scoring_dist_edges_abs', 'use_3d_edges', 'use_view_based', 'use_self_similar_poses'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *string* / real / integer
Werte der generischen Parameter
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0, 1, 'true', 'false', 0.005, 0.01, 0.03, 0.05, 0.1, 'num_scene_points', 'model_point_fraction', 'num_model_points', 'fast', 'mls'}

▷ **Pose** (output_control) . . . . . pose(-array) ⇝ *real* / integer
3D-Lage des Oberflächenmodells in der Szene.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Objektinstanzen.

▷ **SurfaceMatchingResultID** (output_control) . . . . . surface_matching_result(-array) ⇝ *handle*
Ergebnishandle, wenn ReturnResultHandle aktiviert ist.

Ergebnis

Wenn alle Parameter korrekt sind liefert find_surface_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d, get_object_model_3d_params, read_surface_model, create_surface_model, get_surface_model_param, edges_object_model_3d

Nachfolger

refine_surface_model_pose, get_surface_matching_result, clear_surface_matching_result, clear_object_model_3d

Alternativen

refine_surface_model_pose, find_surface_model_image, refine_surface_model_pose_image

Siehe auch

refine_surface_model_pose, find_surface_model_image

Modul

3D Metrology

find_surface_model_image (Image : : SurfaceModelID, ObjectModel3D, RelSamplingDistance, KeyPointFraction, MinScore, ReturnResultHandle, GenParamName, GenParamValue : Pose, Score, SurfaceMatchingResultID)

Suche der besten Matches eines Oberflächenmodells in einer 3D-Szene und Bildern.

find_surface_model_image sucht die besten Matches des Oberflächenmodells SurfaceModelID in der Szene, die durch das 3D-Objektmodell ObjectModel3D und Bildern der Szene Image dargestellt wird. Die Anzahl der in Image übergebenen Bilder muss mit der Anzahl der mit set_surface_model_param gesetzten Kameras übereinstimmen. Weiterhin muss bei der Erzeugung des Oberflächenmodells in create_surface_model der Parameter *'train_3d_edges'* aktiviert worden sein.

Die Bilder werden nur für die grobe und feine Pose-Verfeinerung verwendet, nicht für das näherungsweise Matching. Dazu wird gleichzeitig die Übereinstimmung des Modells mit den 3D-Daten der Szene sowie die Übereinstimmung der rückprojizierten Modellkanten mit den Kanten in den übergebenen Bildern optimiert. Der Definitionsbereich der Bilder wird hierbei ignoriert.

Zusätzlich zu den in find_surface_model dokumentierten Parametern unterstützt find_surface_model_image auch die folgenden generischen Parameter:

'min_contrast': Setzt den Mindestkontrast des Objekts im Suchbild. Kanten deren Kontrast unter diesem Schwellenwert liegen werden bei der Verfeinerung ignoriert.
Vorgeschlagene Werte: 5, 10, 20
Standardwert: 10
Restriktion: 'min_contrast' >= 0

'max_deformation': Setzt den Suchbereich in Pixeln für korrespondierenden Bildkanten. Dieser Parameter kann verwendet werden, wenn das Objekt gegenüber den in create_surface_model trainierten Modell leicht verformt ist. Eine Erhöhung dieses Parameters kann einen deutlichen Einfluss auf die Laufzeit des Operators haben.
Vorgeschlagene Werte: 0, 1, 5
Standardwert: 1
Restriktion: 'max_deformation' >= 0

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object* : byte / uint2
Bilder der Szene.

▷ **SurfaceModelID** (input_control) ..... surface_model ⇝ *handle*
Handle des Oberflächenmodells.

▷ **ObjectModel3D** (input_control) ..... object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells, welches die Szene enthält.

▷ **RelSamplingDistance** (input_control) ..... real ⇝ *real*
Abtastdistanz der Szene relativ zum Durchmesser des Oberflächenmodells.
Defaultwert: 0.05
Wertevorschläge: RelSamplingDistance ∈ {0.1, 0.07, 0.05, 0.04, 0.03}
Restriktion: 0 < RelSamplingDistance < 1

▷ **KeyPointFraction** (input_control) ..... real ⇝ *real*
Anteil der abgetasteten Szenepunkte, die als Schlüsselpunkte verwendet werden.
Defaultwert: 0.2
Wertevorschläge: KeyPointFraction ∈ {0.3, 0.2, 0.1, 0.05}
Restriktion: 0 < KeyPointFraction <= 1

▷ **MinScore** (input_control) ..... real(-array) ⇝ *real* / integer
Mindestbewertung der zurückgegebenen Posen.
Defaultwert: 0
Restriktion: MinScore >= 0

▷ **ReturnResultHandle** (input_control) ..... string ⇝ *string*
Aktiviert die Rückgabe eines Ergebnishandels in SurfaceMatchingResultID.
Defaultwert: 'false'
Wertevorschläge: ReturnResultHandle ∈ {'true', 'false'}

▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'num_matches', 'max_overlap_dist_rel', 'max_overlap_dist_abs', 'sparse_pose_refinement', 'dense_pose_refinement', 'pose_ref_num_steps', 'pose_ref_sub_sampling', 'pose_ref_dist_threshold_rel', 'pose_ref_dist_threshold_abs', 'pose_ref_scoring_dist_rel', 'pose_ref_scoring_dist_abs', 'pose_ref_use_scene_normals', 'scene_normal_computation', 'scene_invert_normals', '3d_edge_min_amplitude_rel', '3d_edge_min_amplitude_abs', 'viewpoint', 'max_gap', '3d_edges', 'max_deformation', 'min_contrast', 'use_3d_edges', 'use_view_based', 'use_self_similar_poses'}

▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *string* / real / integer
Werte der generischen Parameter
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0, 1, 'true', 'false', 0.005, 0.01, 0.03, 0.05, 0.1, 'num_scene_points', 'model_point_fraction', 'num_model_points', 'fast', 'mls'}

▷ **Pose** (output_control) ..... pose(-array) ⇝ *real* / integer
3D-Lagen des Oberflächenmodells in der Szene.

▷ **Score** (output_control) ..... real-array ⇝ *real*
Bewertung der gefundenen Objektinstanzen.

▷ **SurfaceMatchingResultID** (output_control) ..... surface_matching_result(-array) ⇝ *handle*
Ergebnishandle, wenn ReturnResultHandle aktiviert ist.

Ergebnis

Wenn alle Parameter korrekt sind liefert find_surface_model_image den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d, get_object_model_3d_params, read_surface_model, create_surface_model, get_surface_model_param, edges_object_model_3d

Nachfolger

refine_surface_model_pose, get_surface_matching_result, clear_surface_matching_result, clear_object_model_3d

Alternativen

refine_surface_model_pose, find_surface_model, refine_surface_model_pose_image

Siehe auch

refine_surface_model_pose, find_surface_model

Modul

3D Metrology

get_surface_matching_result (: : SurfaceMatchingResultID, ResultName, ResultIndex : ResultValue)

Auslesen von Ergebnissen des oberflächenbasierten Matchings.

Der Operator get_surface_matching_result liefert detaillierte Ergebnisse des oberflächenbasierten Matchings oder der oberflächenbasierten Pose-Verfeinerung. Die Ergebnisse sind in SurfaceMatchingResultID gespeichert, das mit find_surface_model oder refine_surface_model_pose angelegt wurde.

Der Parameter ResultName wählt das abzurufende Detailergebnis aus. Bezieht sich das Detailergebnis auf eine der zurückgelieferten Posen, so wählt ResultIndex diese aus, wobei mittels *0* das erste Ergebnis selektiert wird. Für einige Werte von ResultName wird ResultIndex ignoriert.

Die folgenden Werte sind für ResultName möglich, wenn SurfaceMatchingResultID von find_surface_model oder find_surface_model_image erzeugt wurde:

'sampled_scene': Es wird ein 3D-Objektmodell zurückgegeben, welches die für das näherungsweise Matching abgetasteten Szenepunkte enthält. Dies ist hilfreich um die Abtastdistanz einzustellen (siehe Parameter RelSamplingDistance von find_surface_model). Der Parameter ResultIndex wird ignoriert.

'key_points': Es wird ein 3D-Objektmodell zurückgegeben, welches die Schlüsselpunkte enthält, die für das Matching verwendet wurden. Dies ist hilfreich um die Abtastdistanz und die Anzahl an Schlüsselpunkten einzustellen (siehe Parameter KeyPointFraction von find_surface_model). Der Parameter ResultIndex wird ignoriert.

'score_unrefined': Es wird die Bewertung des Ergebnisses vor der exakten Pose-Verfeinerung zurückgegeben. Je nachdem, ob die grobe Pose-Verfeinerung aktiviert war ist dies die Bewertung des näherungsweisen Matchings oder der groben Pose-Verfeinerung. Eine detaillierte Beschreibung der Bewertung findet sich in find_surface_model. Der Index des Ergebnisses wird in ResultIndex übergeben. Wenn SurfaceMatchingResultID mittels refine_surface_model_pose erstellt wurde, wird 0 zurückgegeben.

'sampled_3d_edges': Wenn das Oberflächenmodell für kantenunterstütztes oberflächenbasiertes Matching vorbereitet wurde, wird ein 3D-Objektmodell zurückgegeben, welches die abgetasteten 3D-Kanten enthält die für das näherungsweise Matching verwendet wurden. Der Parameter ResultIndex wird ignoriert.

Die folgenden Werte für ResultName sind unabhängig davon, von welchem Operator SurfaceMatchingResultID erzeugt wurde, möglich:

'pose': Gibt die Pose eines der Ergebnisse zurück. Der Index des Ergebnisses wird in ResultIndex übergeben.

'score_refined': Gibt die Bewertung des Ergebnisses nach der exakten Pose-Verfeinerung zurück. Eine detaillierte Beschreibung der Bewertung findet sich in find_surface_model. Der Index des Ergebnisses wird in ResultIndex übergeben. Wenn SurfaceMatchingResultID mittels find_surface_model erstellt wurde und die exakte Pose-Verfeinerung deaktiviert war, wird 0 zurückgegeben.

'score': Gibt die kombinierte Bewertung des in ResultIndex indizierten Ergebnisses zurück, d.h., diese Bewertung ist gleich der Bewertung Score des Operators find_surface_model.

'score_surface': Gibt die oberflächenbasierte Bewertung des in ResultIndex indizierten Ergebnisses zurück. Falls nicht speziell anders gesetzt, ist diese Bewertung und *'score_refined'* gleich.

'score_3d_edges': Gibt die Bewertung der 3D-Kanten des in ResultIndex indizierten Ergebnisses zurück. Diese Bewertung ist nur bei oberflächenbasiertem Matching mit Kantenunterstützung anwendbar.

'score_2d_edges': Gibt die bildkantenbasierte Bewertung des in ResultIndex indizierten Ergebnisses zurück. Diese Bewertung ist nur bei oberflächenbasiertem Matching mit Kantenunterstützung anwendbar.

'score_view_based': Gibt die ansichtsbasierte Bewertung des in ResultIndex indizierten Ergebnisses zurück. Diese Bewertung ist nur bei oberflächenbasiertem Matching mit einem Modell, das speziell für diese Berechnung trainiert wurde, anwendbar.

'all_scores': Gibt für das in ResultIndex indizierte Ergebnis die fünf Bewertungen *'score'*, *'score_surface'*, *'score_3d_edges'*, *'score_2d_edges'* und *'score_view_based'* zurück. Dabei haben die Bewertungen die gleiche Reihenfolge wie die Schwellenwerte, die über den Parameter MinScore der Matching- und Verfeinerungsoperatoren übergeben werden können.

Parameter

▷ **SurfaceMatchingResultID** (input_control) ..... surface_matching_result ⇝ *handle*
Ergebnishandle des oberflächenbasierten Matchings.

▷ **ResultName** (input_control) ..... string(-array) ⇝ *string*
Name des Ergebnisdetails.
Defaultwert: 'pose'
Werteliste: ResultName ∈ {'sampled_scene', 'key_points', 'pose', 'score_unrefined', 'score_refined', 'sampled_3d_edges', 'score', 'score_surface', 'score_3d_edges', 'score_2d_edges', 'score_view_based', 'all_scores'}

▷ **ResultIndex** (input_control) . . . . . integer ⇝ *integer*
Index des Ergebnisses, beginnend mit 0.
Defaultwert: 0
Wertevorschläge: ResultIndex ∈ {0, 1, 2, 3}
Restriktion: ResultIndex >= 0

▷ **ResultValue** (output_control) . . . . . integer(-array) ⇝ *integer* / string / real / handle
Wert des Ergebnisses.

Ergebnis

Ist der Handle des Modells gültig, dann liefert get_surface_matching_result den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

find_surface_model, refine_surface_model_pose

Nachfolger

clear_surface_model

Siehe auch

find_surface_model, refine_surface_model_pose, read_surface_model, write_surface_model, clear_surface_model

Modul

3D Metrology

get_surface_model_param (: : SurfaceModelID, GenParamName : GenParamValue)

Auslesen von Einstellungen und Eigenschaften eines Oberflächenmodells.

get_surface_model_param liefert Einstellungen und Eigenschaften des Oberflächenmodells SurfaceModelID. Das Oberflächenmodell muss mittels create_surface_model oder read_surface_model erstellt worden sein. Die Namen der abzurufenden Werte werden im generischen Parameter GenParamName übergeben, die entsprechenden Werte werden in GenParamValue zurückgegeben.

Die folgenden Werte sind für GenParamName möglich:

'diameter': Durchmesser der Modellpunktwolke. Der Durchmesser ist die Länge der Diagonalen des umschließenden achsenparallelen Quaders (siehe *'bounding_box1'*).

'center': Mittelpunkt des Modells. Der Mittelpunkt ist das Zentrum des umschließenden achsenparallelen Quaders (siehe *'bounding_box1'*).

'bounding_box1': Kleinster umschließender achsenparalleler Quader, zurückgegeben als (min_x, min_y, min_z, max_x, max_y, max_z).

'sampled_model': Die für das Matching abgetasteten 3D-Punkte des Modells. Zurückgegeben wird das Handle eines 3D-Objektmodells, welches alle Punkte enthält, die für das näherungsweise Matching abgetastet wurden (siehe create_surface_model).

'sampled_pose_refinement': Die für die Pose-Verfeinerung abgetasteten 3D-Punkte des Modells. Zurückgegeben wird das Handle eines 3D-Objektmodells, welches alle Punkte enthält die für die Pose-Verfeinerung abgetastet wurden (siehe create_surface_model).

'3d_edges_trained': Gibt zurück ob das Oberflächenmodell für kantenunterstützte Matching vorbereitet wurde, d.h., ob der Parameter *'train_3d_edges'* in create_surface_model für dieses Modell aktiviert wurde. Zurückgegeben wird der String *'true'* oder *'false'*.

'view_based_trained': Gibt zurück ob das Oberflächenmodell zur Berechnung einer ansichtsbasierten Bewertung während des Matchings vorbereitet wurde, d.h., ob der Parameter *'train_view_based'* in `create_surface_model` für dieses Modell aktiviert wurde. Zurückgegeben wird der String *'true'* oder *'false'*.

'camera_parameter':

'camera_parameter X': Gibt die Kameraparameter der Kamera mit der Nummer X zurück, wobei X ein 0-basierter Index ist. Falls nicht angegeben, ist X standardmäßig 0 (erste Kamera). Die Kameraparameter müssen vorher mittels `set_surface_model_param` gesetzt worden sein.

'camera_pose':

'camera_pose X': Gibt die Kamerapose der Kamera mit der Nummer X zurück, wobei X ein 0-basierter Index ist. Falls nicht angegeben, ist X standardmäßig 0 (erste Kamera).

'symmetry_axis_direction':

'symmetry_axis_origin': Gibt die Richtung der vorher mit `set_surface_model_param` gesetzten Symmetrieachse bzw. einen auf ihr liegenden Punkt zurück. Wenn keine Achse gesetzt wurde, wird ein leeres Tupel zurückgegeben.

'symmetry_poses': Gibt die mit `set_surface_model_param` gesetzten Symmetrieposen zurück.

'symmetry_poses_all': Gibt alle Symmetrieposen zurück, die `set_surface_model_param` auf Grundlage der gesetzten Symmetrieposen gefunden hat.

'pose_restriction_reference_pose': Gibt die mit `set_surface_model_param` gesetzte Referenzpose des Modells zurück. Wenn keine Referenzpose gesetzt ist, wird ein leeres Tupel zurückgegeben.

'pose_restriction_max_angle_diff': Gibt die mittels `set_surface_model_param` gesetzte maximale Winkeldifferenz zwischen der Referenzpose und einer gefundenen Pose zurück. Der Winkel wird im Bogenmaß zurückgegeben. Wenn keine Winkeldifferenz gesetzt ist, wird ein leeres Tupel zurückgegeben.

'pose_restriction_allowed_axis_direction':

'pose_restriction_allowed_axis_origin': Gibt die Richtung der vorher mit `set_surface_model_param` gesetzten zu ignorierenden Rotationsachse zurück. Wenn keine Achse gesetzt wurde, wird ein leeres Tupel zurückgegeben.

'pose_restriction_filter_final_poses_only': Gibt *'true'* zurück, wenn nur die finalen Posen gefiltert werden sollen, oder *'false'*, wenn die Filterung bereits während des Matchingprozesses geschehen soll (Standardwert).

'self_similar_poses_trained': Gibt zurück ob das Oberflächenmodell zur Optimierung bezüglich selbstähnlicher, fast-symmetrischer Lagen vorbereitet wurde, d.h. ob der Parameter *'train_self_similar_poses'* in `create_surface_model` für dieses Modell aktiviert wurde. Zurückgegeben wird der String *'true'* oder *'false'*.

'sampled_self_similarity': Gibt das Handle eines 3D-Objektmodells zurück welches die 3D-Punkte des Modells enthält, die für Suche nach selbstähnlichen Posen abgetastet wurden.

'self_similar_poses': Gibt die Posen zurück, unter denen das Objekt selbstähnlich, d.h. fast-symmetrisch, ist. Diese stehen nur zur Verfügung, wenn das Oberflächenmodell in `create_surface_model` mit aktiviertem Parameter *'train_self_similar_poses'* trainiert wurde.

'self_similar_poses_models': Gibt das Handle von 3D-Objektmodellen zurück, welche das Originalmodell enthalten, das in die mit *'self_similar_poses'* abrufbaren Posen transformiert wurden. Dies erlaubt eine Untersuchung der gefundenen selbstähnlichen Lagen. Dieser Parameter steht nur zur Verfügung, wenn das Oberflächenmodell in `create_surface_model` mit aktiviertem Parameter *'train_self_similar_poses'* trainiert wurde.

Parameter

▷ **`SurfaceModelID`** (input_control) . . . . . surface_model ⇝ *handle*
Handle des Oberflächenmodells.

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name der abzurufenden Eigenschaft.
Defaultwert: 'diameter'
Werteliste: `GenParamName` ∈ {'diameter', 'center', 'bounding_box1', 'sampled_model', 'sampled_pose_refinement', '3d_edges_trained', 'camera_parameter', 'camera_pose', 'symmetry_axis_direction', 'symmetry_axis_origin', 'symmetry_poses', 'symmetry_poses_all',

'pose_restriction_reference_pose', 'pose_restriction_max_angle_diff',
'pose_restriction_allowed_axis_direction', 'pose_restriction_allowed_axis_origin',
'pose_restriction_filter_final_poses_only', 'view_based_trained', 'self_similar_poses_trained',
'sampled_self_similarity', 'self_similar_poses', 'self_similar_poses_models' }

▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *real* / string / integer / handle
Wert der abzurufenden Eigenschaft.

Ergebnis

Wenn alle Parameter korrekt sind liefert `get_surface_model_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_surface_model`, `read_surface_model`

Nachfolger

`find_surface_model`, `refine_surface_model_pose`, `write_surface_model`

Siehe auch

`create_surface_model`, `set_surface_model_param`

Modul

3D Metrology

read_surface_model (: : FileName : SurfaceModelID)

Liest ein Oberflächenmodell aus einer Datei.

`read_surface_model` liest ein Oberflächenmodell welches zuvor mittels `write_surface_model` geschrieben wurde aus der Datei `FileName` ein. Das Handle des eingelesenen Oberflächenmodells wird in `SurfaceModelID` zurückgegeben. Enthält `FileName` keine absolute Pfadangabe, so wird die Datei im aktuellen Verzeichnis des HALCON-Prozesses gesucht. Die Standard-Dateiendung für Oberflächenmodelle in HALCON ist 'sfm'. Existiert die Datei `FileName` nicht, so wird diese Standard-Dateiendung angehängt und erneut gesucht.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname der SFM-Datei.
Dateiendung: `.sfm`

▷ **SurfaceModelID** (output_control) . . . . . surface_model ⇝ *handle*
Handle des gelesenen Oberflächenmodells.

Ergebnis

Wenn alle Parameter korrekt sind und die Datei gelesen werden konnte liefert `read_surface_model` den Wert TRUE. Wenn die Datei kein Oberflächenmodell enthält wird der Fehler 9506 zurückgegeben. Enthält die Datei ein Oberflächenmodell in einer Version, die nicht mit dieser HALCON-Version gelesen werden kann, wird der Fehler 9507 zurückgegeben. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d

Nachfolger

find_surface_model, refine_surface_model_pose, get_surface_model_param, clear_surface_model, find_surface_model_image, refine_surface_model_pose_image

Alternativen

create_surface_model

Siehe auch

create_surface_model, write_surface_model

Modul

3D Metrology

refine_surface_model_pose (: : SurfaceModelID, ObjectModel3D, InitialPose, MinScore, ReturnResultHandle, GenParamName, GenParamValue : Pose, Score, SurfaceMatchingResultID)

Verfeinern der Pose eines Oberflächenmodells in einer 3D-Szene.

refine_surface_model_pose verfeinert die näherungsweise Pose InitialPose des Oberflächenmodells SurfaceModelID in der 3D-Szene ObjectModel3D. Das Oberflächenmodell SurfaceModelID muss zuvor mittels create_surface_model oder read_surface_model erstellt worden sein. Das 3D-Objektmodell der Szene kann beispielsweise mittels read_object_model_3d aus einer Datei gelesen worden sein oder mittels xyz_to_object_model_3d erstellt worden sein. Weiterhin können einige mittels set_surface_model_param gesetzte Parameter des Oberflächenmodells die Pose-Verfeinerung beeinflussen, wie etwa das Einschränken des Rotationsbereichs.

refine_surface_model_pose ist nützlich wenn die Lage eines 3D-Objekts in einer 3D-Szene ungefähr bekannt ist und lediglich verfeinert werden soll. Die verfeinerte Pose wird in Pose zurückgegeben, zusammen mit einer Bewertung in Score. Es können auch mehrere Posen zur Verfeinerung übergeben werden. Es ist zu beachten, dass anders als bei find_surface_model die resultierenden Posen nicht nach ihrer Bewertung sortiert werden, sondern in der selben Reihenfolge zurückgegeben werden, in der sie übergeben wurden.

Die maximal mögliche Abweichung der Pose, bis zu der die Verfeinerung funktioniert, hängt stark vom Objekt und der Szene ab. Im allgemeinen können Abweichungen in der Rotation von bis zu 15° und Abweichungen in der Lage von bis zu 10% des Modelldurchmessers verfeinert werden.

Die numerisch maximal erreichbare Positionsgenauigkeit nach der Posenverfeinerung liegt bei etwa 0.1% der Modellgröße. Die Genauigkeit hängt weiter vom Rauschen der Daten, von der Anzahl der Datenpunkte und von der Modellform ab.

Details über die Verfeinerung und die Parameter finden sich in der Dokumentation zu find_surface_model im Abschnitt über die exakte Pose-Verfeinerung. Die folgenden generischen Parameter, die ebenfalls in find_surface_model dokumentiert sind, können für refine_surface_model_pose gesetzt werden: *'pose_ref_num_steps'*, *'pose_ref_sub_sampling'*, *'pose_ref_dist_threshold_rel'*, *'pose_ref_dist_threshold_abs'*, *'pose_ref_scoring_dist_rel'*, *'pose_ref_scoring_dist_abs'*, *'pose_ref_use_scene_normals'*, *'3d_edge_min_amplitude_rel'*, *'3d_edge_min_amplitude_abs'*, *'3d_edges'*, *'use_3d_edges'*, *'use_view_based'*, *'use_self_similar_poses'*, *'pose_ref_dist_threshold_edges_rel'*, *'pose_ref_dist_threshold_edges_abs'*, *'pose_ref_scoring_dist_edges_rel'* und *'pose_ref_scoring_dist_edges_abs'*.

Parameter

- ▷ **SurfaceModelID** (input_control) ..... surface_model ⇝ *handle*
  Handle des Oberflächenmodells.
- ▷ **ObjectModel3D** (input_control) ..... object_model_3d ⇝ *handle*
  Handle des 3D-Objektmodells, welches die Szene enthält.
- ▷ **InitialPose** (input_control) ..... pose(-array) ⇝ *real* / integer
  Zu verfeinernde 3D-Lagen des Oberflächenmodells in der Szene.

▷ **MinScore** (input_control) . . . . . real(-array) ⇝ *real* / integer
Mindestbewertung der zurückgegebenen Posen.
Defaultwert: 0
Restriktion: MinScore >= 0

▷ **ReturnResultHandle** (input_control) . . . . . string ⇝ *string*
Aktiviert die Rückgabe eines Ergebnishandels in SurfaceMatchingResultID.
Defaultwert: 'false'
Werteliste: ReturnResultHandle ∈ {'true', 'false'}

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'pose_ref_num_steps', 'pose_ref_sub_sampling', 'pose_ref_dist_threshold_rel', 'pose_ref_dist_threshold_abs', 'pose_ref_scoring_dist_rel', 'pose_ref_scoring_dist_abs', 'pose_ref_use_scene_normals', '3d_edge_min_amplitude_rel', '3d_edge_min_amplitude_abs', 'viewpoint', '3d_edges', 'use_3d_edges', 'use_view_based', 'use_self_similar_poses'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *string* / real / integer
Werte der generischen Parameter
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0, 1, 'true', 'false', 0.005, 0.01, 0.03, 0.05, 0.1}

▷ **Pose** (output_control) . . . . . pose(-array) ⇝ *real* / integer
3D-Lagen des Oberflächenmodells in der Szene.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Objektinstanzen.

▷ **SurfaceMatchingResultID** (output_control) . . . . . surface_matching_result(-array) ⇝ *handle*
Ergebnishandle, wenn ReturnResultHandle aktiviert ist.

Ergebnis

Wenn alle Parameter korrekt sind liefert refine_surface_model_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d, get_object_model_3d_params, read_surface_model, create_surface_model, get_surface_model_param, find_surface_model, edges_object_model_3d

Nachfolger

get_surface_matching_result, clear_surface_matching_result, clear_object_model_3d

Alternativen

find_surface_model, refine_surface_model_pose_image, find_surface_model_image

Siehe auch

create_surface_model, find_surface_model, refine_surface_model_pose_image

Modul

3D Metrology

refine_surface_model_pose_image (Image : : SurfaceModelID, ObjectModel3D, InitialPose, MinScore, ReturnResultHandle, GenParamName, GenParamValue : Pose, Score, SurfaceMatchingResultID)

Verfeinern der Pose eines Oberflächenmodells in einer 3D-Szene und Bildern.

refine_surface_model_pose_image verfeinert die näherungsweise Pose InitialPose des Oberflächenmodells SurfaceModelID in der Szene, die durch das 3D-Objektmodell ObjectModel3D und Bildern der Szene Image dargestellt wird. Die Anzahl der in Image übergebenen Bilder muss mit der Anzahl der mit set_surface_model_param gesetzten Kameras übereinstimmen. Weiterhin muss bei der Erzeugung des Oberflächenmodells in create_surface_model der Parameter *'train_3d_edges'* aktiviert worden sein.

refine_surface_model_pose_image optimiert gleichzeitig die Übereinstimmung des Modells mit den 3D-Daten der Szene sowie die Übereinstimmung der rückprojizierten Modellkanten mit den Kanten in den übergebenen Bildern. Der Definitionsbereich der Bilder wird hierbei ignoriert.

Zusätzlich zu den in refine_surface_model_pose dokumentierten Parametern unterstützt der Operator auch die in find_surface_model_image dokumentierten generischen Parameter *'min_contrast'* und *'max_deformation'*.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object* : byte / uint2
  Bilder der Szene.
- ▷ **SurfaceModelID** (input_control) . . . . . surface_model ⇝ *handle*
  Handle des Oberflächenmodells.
- ▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
  Handle des 3D-Objektmodells, welches die Szene enthält.
- ▷ **InitialPose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
  Zu verfeinernde 3D-Lagen des Oberflächenmodells in der Szene.
- ▷ **MinScore** (input_control) . . . . . real(-array) ⇝ *real* / integer
  Mindestbewertung der zurückgegebenen 3D-Lagen.
  Defaultwert: 0
  Restriktion: MinScore >= 0
- ▷ **ReturnResultHandle** (input_control) . . . . . string ⇝ *string*
  Aktiviert die Rückgabe eines Ergebnishandels in SurfaceMatchingResultID.
  Defaultwert: 'false'
  Werteliste: ReturnResultHandle ∈ {'true', 'false'}
- ▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'pose_ref_num_steps', 'pose_ref_sub_sampling', 'pose_ref_dist_threshold_rel', 'pose_ref_dist_threshold_abs', 'pose_ref_scoring_dist_rel', 'pose_ref_scoring_dist_abs', 'pose_ref_use_scene_normals', 'max_deformation', 'min_contrast', '3d_edge_min_amplitude_rel', '3d_edge_min_amplitude_abs', 'viewpoint', '3d_edges', 'use_3d_edges', 'use_view_based', 'use_self_similar_poses'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *string* / real / integer
  Werte der generischen Parameter
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {0, 1, 'true', 'false', 0.005, 0.01, 0.03, 0.05, 0.1}
- ▷ **Pose** (output_control) . . . . . pose(-array) ⇝ *real* / integer
  3D-Lage des Oberflächenmodells in der Szene.
- ▷ **Score** (output_control) . . . . . real-array ⇝ *real*
  Bewertung der gefundenen Objektinstanzen.
- ▷ **SurfaceMatchingResultID** (output_control) . . . . . surface_matching_result(-array) ⇝ *handle*
  Ergebnishandle, wenn ReturnResultHandle aktiviert ist.

Ergebnis

Wenn alle Parameter korrekt sind liefert refine_surface_model_pose_image den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`, `get_object_model_3d_params`, `read_surface_model`, `create_surface_model`, `get_surface_model_param`, `find_surface_model`, `edges_object_model_3d`

Nachfolger

`get_surface_matching_result`, `clear_surface_matching_result`, `clear_object_model_3d`

Alternativen

`find_surface_model`, `refine_surface_model_pose`, `find_surface_model_image`

Siehe auch

`create_surface_model`, `find_surface_model`, `refine_surface_model_pose`

Modul

3D Metrology

`serialize_surface_model` (: : SurfaceModelID : SerializedItemHandle)

Serialisieren eines Oberflächenmodells.

`serialize_surface_model` serialisiert die Daten eines Oberflächenmodells (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_surface_model` in eine Datei geschrieben werden. Das Oberflächenmodell wird in dem Handle `SurfaceModelID` übergeben. Das serialisierte Oberflächenmodell wird in dem Handle `SerializedItemHandle` zurückgegeben. und kann mit `deserialize_surface_model` wieder deserialisiert werden.

Parameter

▷ **`SurfaceModelID`** (input_control) . . . . . surface_model ⇝ *handle*
Handle des Oberflächenmodells.

▷ **`SerializedItemHandle`** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_surface_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_surface_model`, `create_surface_model`, `get_surface_model_param`

Nachfolger

`clear_surface_model`, `fwrite_serialized_item`, `send_serialized_item`, `deserialize_surface_model`

Siehe auch

create_surface_model, read_surface_model, write_surface_model

Modul

3D Metrology

set_surface_model_param (: : SurfaceModelID, GenParamName, GenParamValue :)

Setzen von Parametern eines Oberflächenmodells.

Der Operator set_surface_model_param setzt Parameter und Eigenschaften des Oberflächenmodells SurfaceModelID. Das Oberflächenmodell muss vorher mittels create_surface_model oder read_surface_model erstellt worden sein. Die Namen der zu verändernden Parameter werden in GenParamName übergeben, die entsprechenden neuen Werte in GenParamValue.

Die möglichen Werte für GenParamName sind unten aufgeführt.

- **Kameras für die bildunterstützte Posenverfeinerung.** Die folgenden Parameter erlauben das Setzen und Löschen von Kameraparametern und -posen. Diese werden von find_surface_model_image und refine_surface_model_pose_image verwendet, um das Modell in die dort übergebenen Bilder zu projizieren.

  Die Kameraparameter müssen vor der Kamerapose eingestellt werden.

  'camera_parameter':

  'camera_parameter X': Setzt die Kameraparameter der Kamera mit Nummer X. X ist ein 0-basierter Index und standardmäßig 0 (erste Kamera). Die Kameraparameter werden von den Operatoren find_surface_model_image und refine_surface_model_pose_image verwendet, welche die entsprechenden Kamerabilder für die Pose-Verfeinerung verwenden. Kameras müssen in aufsteigender Reihenfolge hinzugefügt werden.

  'camera_pose':

  'camera_pose X': Setzt die Kamerapose der Kamera mit Nummer X. X ist ein 0-basierter Index und standardmäßig 0 (erste Kamera). Beim Hinzufügen einer neuen Kamera wird die Pose standardmäßig auf die Nullpose *[0,0,0,0,0,0,0]* gesetzt, was bedeutet, dass die Kamera und der 3D-Sensor den gleichen Ursprung haben.

  'clear_cameras': Löscht alle vorher im Oberflächenmodell gesetzten Kameras.

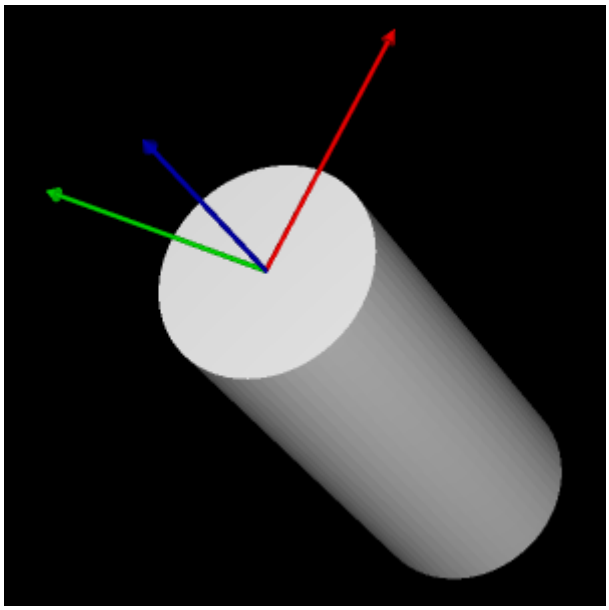
- **Objektsymmetrien definieren.** Die folgenden Parameter können verwendet werden, um im Oberflächenmodell Symmetrien des Objekts zu setzen. Diese Information kann verwendet werden, um das oberflächenbasierte Matching zu beschleunigen.

  Für mit *'train_3d_edges'* erstellte Oberflächenmodelle können keine Symmetrien gesetzt werden.

  Standardmäßig sind keine Symmetrien aktiviert.

  Zu beachten ist, dass set_surface_model_param beim Ändern der Symmetrien einige interne Datenstrukturen neu aufbaut, damit das oberflächenbasierte Matching schneller wird. Dies kann einige Sekunden dauern.

  'symmetry_axis_direction': Setzt die Richtung der Symmetrieachse des Modells. GenParamValue muss ein Tupel mit drei Zahlen sein, welche die x-, y- und z-Werte der Achsenrichtung sind. Das Modell wird modifiziert, um diese Information zum Beschleunigen des oberflächenbasierten Matchings zu verwenden.
  Um die Symmetrieachse zu löschen, kann in GenParamValue ein leeres Tupel übergeben werden.



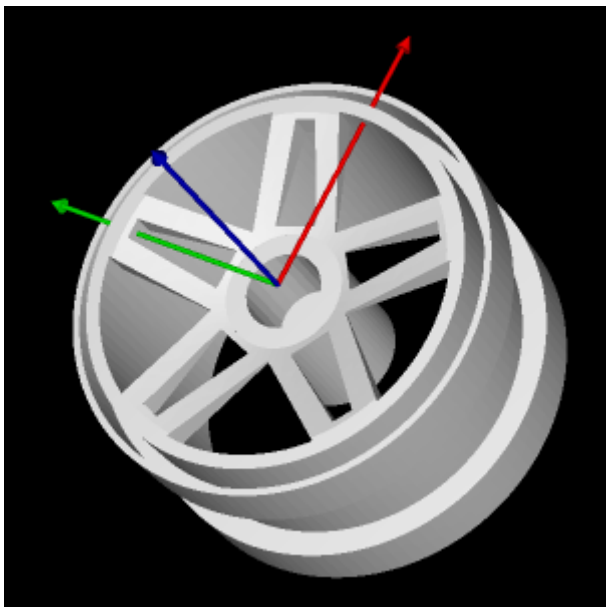
Ein Objekt (Zylinder), wobei *[0,0,1]* die Richtung der Symmetrieachse ist.

'symmetry_axis_origin': Setzt einen Punkt auf der Symmetrieachse des Modells. `GenParamValue` muss ein Tupel mit drei Zahlen sein, welche einen beliebigen Punkt in Modellkoordinaten darstellen, der auf der Symmetrieachse liegt. Dieser Parameter ist optional. Standardmäßig wird er auf den wie von `get_surface_model_param` zurückgegebenen Mittelpunkt des Modells gesetzt.

'symmetry_poses': Setzt eine oder mehr Symmetrieposen des Modells (siehe `create_pose`). Das Modell muss, wenn es mit einer dieser Posen transfomiert wird, zu sich selbst identisch sein. Das Modell wird modifiziert, um diese Information zum Beschleunigen des oberflächenbasierten Matchings zu verwenden.

Wenn eine oder mehr Symmetrieposen gesetzt werden, erzeugt `set_surface_model_param` intern alle durch Invertierung und Verkettung dieser Posen erzeugbaren weiteren Posen. Um alle derart intern erstellten Posen zu erhalten, kann `get_surface_model_param` mit dem Parameter *'symmetry_poses_all'* verwendet werden. Wenn intern mehr als 100 Posen gefunden werden, wird ein Fehler zurückgegeben, da dies auf ungültige Symmetrieposen hindeutet.

Um die Symmetrieposen zu löschen, kann ein leeres Tupel in `GenParamValue` übergeben werden. Es kann nur entweder eine Symmetrieachse oder Symmetrieposen gesetzt werden, aber nicht beides.



Ein Objekt mit diskontinuierlicher Symmetrie. Die Symmetriepose für dieses Objekt ist *[0,0,0, 0,0,360.0/5, 0]*.

- **Einschränkung des Posenbereichs.** Die folgenden Parameter können verwendet werden, um den Rotationsbereich einzuschränken, in dem `find_surface_model` nach dem Oberflächenmodell sucht, oder um den

erlaubten Rotationsbereich für refine_surface_model_pose zu setzen.

Standardmäßig gibt es keine Poseneinschränkungen, d.h. das Modell wird in allen möglichen Rotationen gesucht.

Zu beachten ist, dass set_surface_model_param beim Ändern der Poseneinschränkungen einige interne Datenstrukturen neu aufbaut, damit das oberflächenbasierte Matching schneller wird. Dies kann einige Sekunden dauern.

'pose_restriction_reference_pose': Setzt die Referenzpose des Modells. Die Referenzpose kann zusammen mit *'pose_restriction_max_angle_diff'* verwendet werden, um den erlaubten Rotationsbereichs des Modells einzuschränken.
Wenn GenParamValue ein leeres Tupel ist, wird eine vorher gesetzte Referenzpose gelöscht und die Poseneinschränkung des Modells damit aufgehoben.
Andernfalls muss GenParamValue eine Pose sein (siehe create_pose). Der Translationsanteil der Pose wird ignoriert. Zu beachten ist auch, dass sowohl *'pose_restriction_reference_pose'* als auch *'pose_restriction_max_angle_diff'* gesetzt werden müssen, damit eine Poseneinschränkung aktiv ist.

'pose_restriction_max_angle_diff': Legt fest, um wie viel die Rotation eine beim oberflächenbasierten Matching gefundenen Pose von der Rotation der mit *'pose_restriction_reference_pose'* gesetzten Referenzpose abweichen darf. Der Winkel wird im Bogenmaß angegeben.
Wenn GenParamValue ein leeres Tupel ist, wird ein vorher gesetzter Winkel gelöscht und die Poseneinschränkung aufgehoben.
Andernfalls muss GenParamValue ein Winkel sein, der angibt, um wie viel die Rotation einer gefundenen Pose *'P'* von der Referenzpose *'R'* abweiche darf. Der Vergleich erfolgt für jeden der Modellpunkte über die Gleichung $\angle(R\mathbf{v}, P\mathbf{v}) \leq \text{max_angle_diff}$, wobei $\mathbf{v}$ der Vektor des 3D-Punktes ist.

'pose_restriction_allowed_axis_direction': Rotation um die mit diesem Parameter gesetzte Achse werden beim Vergleichen von gefundenen Posen mit der mit *'pose_restriction_reference_pose'* gesetzten Referenzpose ignoriert.
Wenn GenParamValue ein leeres Tupel ist, wird eine vorher gesetzte Achse gelöscht.
Andernfalls muss GenParamValue drei Werte enthalten, die die Richtung der zu ignorierenden Achse in Modellkoordinaten enthält. Wenn eine solche Achse gesetzt ist, so darf sich ihre Richtung in einer gefundenen Pose von der Richtung der Achse in der Referenzpose höchstens um den mit *'pose_restriction_max_angle_diff'* gesetzten Winkel unterscheiden.
Der Vergleich der Rotation einer gefundenen Pose mit der Referenzpose erfolgt dann mit der Gleichung $\angle(R\,\text{axis}, P\,\text{axis}) \leq \text{max_angle_diff}$.

'pose_restriction_allowed_axis_origin': Definiert einen Punkt auf der mit *'pose_restriction_allowed_axis_direction'* gesetzten Achse. GenParamValue muss ein Tupel mit drei Werten sein, die einen beliebigen Punkt auf dieser Achse darstellen. Dieser Parameter ist optional. Standardmäßig wird er auf den wie von get_surface_model_param zurückgegebenen Mittelpunkt des Modells gesetzt.

'pose_restriction_filter_final_poses_only': Mittels dieses Flags kann zwischen zwei verschiedenen Modi für die Poseneinschränkung gewechselt werden.
Wenn GenParamValue auf *'false'* gesetzt ist (Standardwert), werden Posen mit Rotationen außerhalb des erlaubten Bereichs bereits früh im Matchingprozess unterdrückt. Diese Einstellung sollte verwendet werden, wenn das Objekt in der Szene stets in einer gewissen Orientierung liegt, find_surface_model aber teilweise eine falsche Orientierung zurückliefert. Mit dieser Einstellung liefert find_surface_model evtl. Posen zurück, die der Operator eigentlich als lokal suboptimal erachtet, wenn alle besseren Posen Rotationen außerhalb des erlaubten Bereichs haben. Wenn an refine_surface_model_pose eine Pose übergeben wird, deren Rotation außerhalb des erlaubten Bereichs liegt, wird diese in den erlaubten Bereich projiziert.
Wenn GenParamValue auf *'true'* gesetzt ist, werden nur die endgültigen von find_surface_model gefundenen Posen gefiltert und die außerhalb des erlaubten Bereichs liegenden entfernt. Diese Einstellung sollte verwendet werden, wenn das Objekt von find_surface_model zwar immer korrekt gefunden wird, aber für die Anwendung nicht alle Posen interessant sind, da einige z.B. nicht vom Roboter gegriffen werden können. In dieser Einstellung liefert find_surface_model möglicherweise weniger Posen als angefordert zurück, wenn eine oder mehr der gefundenen Posen außerhalb des erlaubten Bereichs liegen.

- **Setzen der Selbstähnlichkeit.** Die folgenden Parameter können verwendet werden, um die Optimierung bezüglich selbstähnlicher Posen zu steuern. Die Parameter können nur gesetzt werden, wenn create_surface_model mit aktiviertem Parameter *'train_self_similar_poses'* aufgerufen wurde.

Zu beachten ist, dass `set_surface_model_param` beim Ändern der Selbstähnlichkeitssuche einige interne Datenstrukturen neu aufbaut, damit das oberflächenbasierte Matching schneller wird. Dies kann einige Sekunden dauern.

'self_similar_poses': Setzt die selbstähnlichen Posen des Modells. Dies sind Posen, unter denen das Objekt sehr ähnlich aussieht, und die daher bei der Suche verwechselt werden können.
`find_surface_model` findet solche Posen automatisch, wenn *'use_self_similar_poses'* aktiviert ist. Die Posen können mit `get_surface_model_param` wieder abgerufen werden. Sollten diese automatisch gefundenen Posen nicht ausreichen, um die Selbstähnlichkeiten aufzulösen, können mit *'self_similarity_poses'* alternative Posen gesetzt werden. In der Regel wird dies nicht empfohlen.
`GenParamValue` muss eine Liste von Posen enthalten (siehe `create_pose`). Die Identitätspose wird der Liste automatisch hinzugefügt, wenn sie nicht bereits darin enthalten ist.

Achtung

In einigen Fällen kann dieser Operator bereits am Modell durchgeführte Änderungen nicht mehr rückgängig machen, wenn während der Aktualisierung der internen Datenstrukturen ein Fehler wie z.B. zu wenig Speicher auftritt. In diesem Fall kann das Modell in einem inkonsistenten, nur teilweise geänderten Zustand bleiben. Das Modell sollte dann gelöscht und neu erzeugt werden.

Dies betrifft nicht Fehler, die aufgrund ungültiger Parameterwerte auftreten, da diese schon vor der Aktualisierung des Modells geprüft werden.

Es ist auch zu beachten, dass die Neuerstellung der internen Datenstrukturen beim Ändern von Symmetrien oder der Poseneinschränkung so lange dauern kann wie das ursprüngliche `create_surface_model`.

Parameter

- ▷ **`SurfaceModelID`** (input_control) . . . . . surface_model ⇝ *handle*
  Handle des Oberflächenmodells.
- ▷ **`GenParamName`** (input_control) . . . . . attribute.name ⇝ *string*
  Name der zu setzenden Eigenschaft.
  Defaultwert: 'camera_parameter'
  Werteliste: `GenParamName` ∈ {'camera_parameter', 'camera_pose', 'clear_cameras', 'symmetry_axis_direction', 'symmetry_axis_origin', 'symmetry_poses', 'pose_restriction_reference_pose', 'pose_restriction_max_angle_diff', 'pose_restriction_allowed_axis_direction', 'pose_restriction_allowed_axis_origin', 'pose_restriction_filter_final_poses_only', 'self_similar_poses'}
- ▷ **`GenParamValue`** (input_control) . . . . . attribute.value(-array) ⇝ *real* / string / integer
  Wert der zu setzenden Eigenschaft.
  Wertevorschläge: `GenParamValue` ∈ {'true', 'false', [], [0,0,0,0,0,0,0,0], [0,0,1]}

Ergebnis

Wenn alle Parameter korrekt sind liefert `set_surface_model_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SurfaceModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_surface_model`, `read_surface_model`, `get_surface_model_param`

Nachfolger

`find_surface_model`, `refine_surface_model_pose`, `write_surface_model`, `find_surface_model_image`, `refine_surface_model_pose_image`

Siehe auch

`create_surface_model`, `get_surface_model_param`

Modul

3D Metrology

write_surface_model (: : SurfaceModelID, FileName :)

Schreibt ein Oberflächenmodell in eine Datei.

write_surface_model schreibt das Oberflächenmodell SurfaceModelID in die Datei FileName. Die Datei kann mit read_surface_model wieder gelesen werden. Die Standard-Dateiendung für Oberflächenmodelle in HALCON ist 'sfm'.

Parameter

▷ **SurfaceModelID** (input_control) . . . . . surface_model ⇝ *handle*
Handle des Oberflächenmodells.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Dateiname.
Dateiendung: .sfm

Ergebnis

Wenn alle Parameter korrekt sind und die Datei geschrieben werden konnte liefert write_surface_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_surface_model, create_surface_model, get_surface_model_param

Nachfolger

clear_surface_model

Siehe auch

create_surface_model, read_surface_model

Modul

3D Metrology

Kapitel 4

3D-Objektmodell

4.1 Generierung

clear_object_model_3d (: : ObjectModel3D :)

Freigeben des Speichers eines 3D-Objektmodells.

clear_object_model_3d gibt den Speicher eines 3D-Objektmodells, das z.B. mit read_object_model_3d angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle ObjectModel3D ist nach dem Aufruf ungültig.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert clear_object_model_3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ObjectModel3D

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d

Modul

3D Metrology

copy_object_model_3d (: : ObjectModel3D,
Attributes : CopiedObjectModel3D)

Kopieren eines 3D-Objektmodells.

Ein 3D-Objektmodell besteht aus einer Menge von Attributen. Der Operator copy_object_model_3d erstellt ein neues 3D-Objektmodell und kopiert die ausgewählten Attribute des 3D-Eingabeobjektmodells in das neue

3D-Ausgabeobjektmodell. Das 3D-Eingabeobjektmodell ist durch das Handle `ObjectModel3D` definiert. Der Operator gibt das Handle `CopiedObjectModel3D` des neuen 3D-Objektmodells zurück. Der Operator kann genutzt werden, um Speicherplatz zu sparen wenn nicht benötigte Attribute entfernt werden. Zugriff auf die Attribute des 3D-Objektmodells ist z.B. mit dem Operator `get_object_model_3d_params` möglich.

Mit dem Parameter `Attributes` wird festgelegt, welche Attribute kopiert werden sollen. Zusätzlich ist es auch möglich, Attribute, die nicht kopiert werden sollen, über ein vorgesetztes '~' Zeichen im String auszuschließen. Zum Entfernen von Attributen kann alternativ der Operator `remove_object_model_3d_attrib` verwendet werden. Es ist zu beachten, dass das eigentliche 3D-Objektmodell selbst aus einer Menge von Attributen besteht. So sind auch die Punktkoordinaten des Objektmodells ein Attribut. Dies bedeutet, dass mindestens dieses Attribut für `copy_object_model_3d` ausgewählt werden muss, damit ein 3D-Objektmodell kopiert werden kann. Ansonsten wäre das zu kopierende Objektmodell leer. Falls also nur ein 3D-Objektmodell kopiert werden soll ohne weitere Attribute, so muss `Attributes` auf den Wert *'point_coord'* gesetzt werden. Wenn das zu kopierende Attribut nicht vorhanden ist oder kein Attribut ausgewählt wurde, wird eine Fehlerbehandlung durchgeführt.

Die folgenden Werte für den Parameter `Attributes` sind möglich:

'point_coord': Dieser Wert legt fest, dass das Attribut mit den Punktkoordinaten kopiert wird.

'point_normal': Dieser Wert legt fest, dass das Attribut mit den Punktnormalen und das Attribut mit den Punktkoordinaten kopiert werden.

'triangles': Dieser Wert legt fest, dass das Attribut mit den Oberflächendreiecken und das Attribut mit den Punktkoordinaten kopiert werden.

'polygons': Dieser Wert legt fest, dass das Attribut mit den Oberflächenpolygonen und das Attribut mit den Punktkoordinaten kopiert werden.

'lines': Dieser Wert legt fest, dass das Attribut mit den Linien und das Attribut mit den Punktkoordinaten kopiert werden.

'xyz_mapping': Dieser Wert legt fest, dass das Attribut mit der Zuordnung der Bildkoordinaten und das Attribut mit den Punktkoordinaten kopiert werden.

'extended_attribute': Dieser Wert legt fest, dass alle erweiterten Attribute kopiert werden. Ist es notwendig, weitere Attribute zu kopieren, die mit den erweiterten Attributen zusammenhängen, werden diese ebenfalls kopiert. Diese Attribute können z.B. Punktkoordinaten, Oberflächendreiecke, Oberflächenpolygone oder Linienzüge sein.

'primitives_all': Dieser Wert legt fest, dass das Attribut mit den Parametern des 3D-Primitivs (eingeschlossen leerer 3D-Primitive) kopiert wird (z.B. generiert vom Operator `fit_primitives_object_model_3d`).

'primitive_plane': Dieser Wert legt fest, dass das Attribut mit einem 3D-Primitiv "Ebene"kopiert wird (z.B. generiert vom Operator `fit_primitives_object_model_3d`).

'primitive_sphere': Dieser Wert legt fest, dass das Attribut mit einem 3D-Primitiv "Kugel"kopiert wird (z.B. generiert vom Operator `fit_primitives_object_model_3d`)

'primitive_cylinder': Dieser Wert legt fest, dass das Attribut mit einem 3D-Primitiv "Zylinder"kopiert wird (z.B. generiert vom Operator `fit_primitives_object_model_3d`).

'primitive_box': Dieser Wert legt fest, dass das Attribut mit einem 3D-Primitiv "Quader"kopiert wird.

'shape_based_matching_3d_data': Dieser Wert legt fest, dass das Attribut mit dem vorbereiteten 3D-Formmodell für das formbasierte 3D-Matching kopiert wird.

'distance_computation_data': Dieser Wert legt fest, dass das Attribut mit der Datenstruktur für die 3D Distanzberechnung kopiert wird. Diese Datenstruktur kann mit `prepare_object_model_3d` erstellt und mit `distance_object_model_3d` verwendet werden. Wenn dieses Attribut ausgewählt wurde, so werden auch die Attribute kopiert, zu denen der Abstand berechnet werden soll. Wurde die Datenstruktur der Abstandsberechnung beispielsweise für Dreiecke vorbereitet, so werden Dreiecke und Punkte mit kopiert.

'surface_based_matching_data': Dieser Wert legt fest, dass die Daten für das oberflächenbasierte 3D-Matching kopiert werden. Die Attribute sind die 3D Punktkoordinaten und die Punktnormalen. Ist das Attribut mit den Punktnormalen nicht vorhanden, wird das Attribut mit der Zuordnung der 3D Punktkoordinaten zu den Bildkoordinaten kopiert. Ist das Attribut mit der Zuordnung der 3D Punktkoordinaten zu den Bildkoordinaten nicht vorhanden, wird das Attribut mit den Oberflächendreiecken kopiert. Ist das Attribut mit den Oberflächendreiecken auch nicht vorhanden, wird das Attribute mit den Oberflächenpolygonen kopiert. Sind keine dieser Attribute vorhanden, wird eine Fehlerbehandlung durchgeführt.

'segmentation_data': Dieser Wert legt fest, dass die Daten für die 3D-Segmentierung kopiert werden. Es werden die Attribute mit den 3D Punktkoordinaten und die Attribute mit den Oberflächendreiecken kopiert. Sind die Attribute mit den Oberflächendreiecken nicht vorhanden wird das Attribut mit der Zuordnung der Punktkoordinaten zu den Bildkoordinaten kopiert. Sind keine dieser Attribute vorhanden, wird eine Fehlerbehandlung durchgeführt.

'score': Dieser Wert legt fest, dass die Daten für das Qualitätsmaß und das Attribut mit den Punktkoordinaten kopiert werden. Das Qualitätsmaß wird in der Regel durch den Operator `reconstruct_surface_stereo` berechnet.

'red': Dieser Wert legt fest, dass die Daten für den roten Kanal und das Attribut mit den Punktkoordinaten kopiert werden.

'green': Dieser Wert legt fest, dass die Daten für den grünen Kanal und das Attribut mit den Punktkoordinaten kopiert werden.

'blue': Dieser Wert legt fest, dass die Daten für den blauen Kanal und das Attribut mit den Punktkoordinaten kopiert werden.

'original_point_indices': Dieser Wert legt fest, dass das Attribut mit den originalen Punktindizes und das Attribut mit den 3D-Punktkoordinaten kopiert werden. Die originalen Punktindizes können vom Operator `triangulate_object_model_3d` generiert werden.

'all': Dieser Wert legt fest, dass alle verfügbaren Attribute kopiert werden. Die Attribute sind die Punktkoordinaten, die Punktnormalen, die Oberflächendreiecke, die Oberflächenpolygone, das Mapping, das Formmodell für das Matching, die Parameter eines 3D-Primitivs und die erweiterten Attribute.

Parameter

▷ **`ObjectModel3D`** (input_control) ..... object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells.

▷ **`Attributes`** (input_control) ..... string(-array) ⇝ *string* / real / integer
Zu kopierende Attribute.
Defaultwert: 'all'
Werteliste: `Attributes` ∈ {'point_coord', 'point_normal', 'triangles', 'polygons', 'xyz_mapping', 'extended_attribute', 'shape_based_matching_3d_data', 'primitives_all', 'primitive_plane', 'primitive_sphere', 'primitive_cylinder', 'primitive_box', 'surface_based_matching_data', 'segmentation_data', 'distance_computation_data', 'score', 'red', 'green', 'blue', 'all', 'original_point_indices'}

▷ **`CopiedObjectModel3D`** (output_control) ..... object_model_3d ⇝ *handle*
Handle des kopierten 3D-Objektmodells.

Ergebnis

`copy_object_model_3d` liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`get_object_model_3d_params`

Siehe auch

`remove_object_model_3d_attrib`, `set_object_model_3d_attrib`

Modul

3D Metrology

deserialize_object_model_3d (: : SerializedItemHandle : ObjectModel3D)

Deserialisieren eines 3D-Objektmodells.

deserialize_object_model_3d deserialisiert ein 3D-Objektmodell, welches mit serialize_object_model_3d serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte 3D-Objektmodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten 3D-Objektmodell mit dem Handle ObjectModel3D gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **ObjectModel3D** (output_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_object_model_3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

write_object_model_3d, fread_serialized_item, receive_serialized_item, serialize_object_model_3d

Nachfolger

affine_trans_object_model_3d, object_model_3d_to_xyz, prepare_object_model_3d

Alternativen

xyz_to_object_model_3d

Siehe auch

write_object_model_3d, clear_object_model_3d

Modul

3D Metrology

gen_box_object_model_3d (: : Pose, LengthX, LengthY, LengthZ : ObjectModel3D)

Erzeugen eines 3D-Objektmodells, das einen Quader beschreibt.

gen_box_object_model_3d erzeugt ein 3D-Objektmodell, das einen Quader beschreibt. Der Mittelpunkt des Quaders wird über ein Koordinatensystem im Raum bestimmt (Pose). Die Längen der jeweiligen Seiten können mit den Parametern LengthX, LengthX und LengthZ für die jeweiligen Achsen spezifiziert werden. Das Handle des resultierenden 3D-Objektmodells wird im Parameter ObjectModel3D zurückgegeben.

Parameter

▷ **Pose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
Die Lage des Quaders im Raum (Position und Orientierung).

▷ **LengthX** (input_control) . . . . . number(-array) ⇝ *real*
Die Länge des Quaders entlang der x-Achse.

▷ **LengthY** (input_control) . . . . . number(-array) ⇝ *real*
Die Länge des Quaders entlang der y-Achse.
Parameteranzahl: LengthY == LengthX

▷ **LengthZ** (input_control) . . . . . number(-array) ⇝ *real*
Die Länge des Quaders entlang der z-Achse.
Parameteranzahl: LengthZ == LengthX

▷ **ObjectModel3D** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells, das den Quader beschreibt.

Ergebnis

`gen_box_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`smallest_bounding_box_object_model_3d`

Nachfolger

`get_object_model_3d_params`, `sample_object_model_3d`, `clear_object_model_3d`

Siehe auch

`gen_cylinder_object_model_3d`, `gen_sphere_object_model_3d`, `gen_sphere_object_model_3d_center`, `gen_plane_object_model_3d`

Modul

3D Metrology

gen_cylinder_object_model_3d (: : Pose, Radius, MinExtent, MaxExtent : ObjectModel3D)

Erzeugen eines 3D-Objektmodells, das einen Zylinder beschreibt.

`gen_cylinder_object_model_3d` erzeugt ein zylinderförmiges 3D-Primitiv, d.h. ein 3D-Objektmodell, das einen Zylinder beschreibt. Ein Zylinder wird über seinen Mittelpunkt und die Rotationsachse (`Pose`) sowie den Radius (`Radius`) beschrieben. Die Pose hat ihren Ursprung auf der Rotationsachse des Zylinders. Die Z-Achse ist entlang der Rotationsachse des Zylinders orientiert. Zusätzlich kann ein Zylinder durch Angabe der Parameter `MinExtent` und `MaxExtent` begrenzt werden. `MinExtent` und `MaxExtent` geben die Z-Koordinaten des niedrigsten bzw. höchsten Punktes des Zylinders auf der Rotationsachse an. Das Handle des resultierenden 3D-Objektmodells wird im Parameter `ObjectModel3D` zurückgegeben.

Parameter

▷ **Pose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
Die Lage und Orientierung des Mittelpunkts des Zylinders im Raum.

▷ **Radius** (input_control) . . . . . number(-array) ⇝ *real*
Der Radius des Zylinders.

▷ **MinExtent** (input_control) . . . . . number(-array) ⇝ *real*
Niedrigste Z-Koordinate des Zylinders in der Richtung der Rotationsachse.

▷ **MaxExtent** (input_control) . . . . . number(-array) ⇝ *real*
Höchste Z-Koordinate des Zylinders in der Richtung der Rotationsachse.
Restriktion: MinExtent < MaxExtent

▷ **ObjectModel3D** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells, das den Zylinder beschreibt.

Ergebnis

`gen_cylinder_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`get_object_model_3d_params`, `sample_object_model_3d`, `clear_object_model_3d`

Siehe auch

`gen_sphere_object_model_3d`, `gen_sphere_object_model_3d_center`, `gen_plane_object_model_3d`, `gen_box_object_model_3d`

Modul

3D Metrology

gen_empty_object_model_3d (: : : EmptyObjectModel3D)

Erzeugen eines leeren 3D-Objektmodells.

`gen_empty_object_model_3d` erzeugt ein leeres 3D-Objektmodell dessen Handle im Parameter `EmptyObjectModel3D` zurückgegeben wird. Attribute können mittels der Operatoren `set_object_model_3d_attrib` oder `set_object_model_3d_attrib_mod` hinzugefügt werden.

Parameter

▷ **EmptyObjectModel3D** (output_control) ..... object_model_3d ⇝ *handle*
Handle des neuen 3D-Objektmodells.

Ergebnis

`gen_empty_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_object_model_3d_attrib`, `set_object_model_3d_attrib_mod`

Siehe auch

`gen_box_object_model_3d`, `gen_cylinder_object_model_3d`, `gen_sphere_object_model_3d`, `gen_sphere_object_model_3d_center`, `gen_plane_object_model_3d`

Modul

3D Metrology

gen_object_model_3d_from_points (: : X, Y, Z : ObjectModel3D)

Erzeugen eines 3D-Objektmodells aus 3D-Punkten.

gen_object_model_3d_from_points erzeugt ein 3D-Objektmodell, das die durch Y, Y und Z angegebenen Punkte enthält.

Parameter

- ▷ **X** (input_control) . . . . . point3d.x(-array) ⇝ *real*
  Die x-Koordinaten der Punktwolke.
- ▷ **Y** (input_control) . . . . . point3d.y(-array) ⇝ *real*
  Die y-Koordinaten der Punktwolke.
- ▷ **Z** (input_control) . . . . . point3d.z(-array) ⇝ *real*
  Die z-Koordinaten der Punktwolke.
- ▷ **ObjectModel3D** (output_control) . . . . . object_model_3d ⇝ *handle*
  Handle des 3D-Objektmodells, das die gegebenen Punkte enthält.

Ergebnis

gen_object_model_3d_from_points liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

get_object_model_3d_params

Nachfolger

connection_object_model_3d, convex_hull_object_model_3d

Alternativen

xyz_to_object_model_3d

Siehe auch

gen_box_object_model_3d, gen_sphere_object_model_3d, gen_cylinder_object_model_3d

Modul

3D Metrology

gen_plane_object_model_3d (: : Pose, XExtent, YExtent : ObjectModel3D)

Erzeugen eines 3D-Objektmodells, welches eine Ebene beschreibt.

gen_plane_object_model_3d erzeugt ein ebenes 3D Primitiv, d.h. ein 3D-Objektmodell, welches eine Ebene beschreibt. Die Ebene wird über ihren Mittelpunkt und die Rotation angegeben, wobei die Normalenrichtung der Ebene in Richtung der mitgedrehten Z-Achse zeigt. Der Mittelpunkt und die Rotation wird mittels des Parameters Pose angegeben. Zusätzlich kann die Ebene durch ein Polygon begrenzt werden, das durch die Punkte mit den Koordinaten XExtent und YExtent definiert ist. Das Handle des resultierenden 3D-Objektmodells wird im Parameter ObjectModel3D zurückgegeben.

Parameter

- ▷ **Pose** (input_control) . . . . . pose ⇝ *real* / integer
  Der Mittelpunkt und die Rotation der Ebene im Raum.
  Parameteranzahl: Pose == 7
- ▷ **XExtent** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  X-Koordinaten zur Begrenzung der Ebene.

▷ **YExtent** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
Y-Koordinaten zur Begrenzung der Ebene.
Parameteranzahl: XExtent == YExtent

▷ **ObjectModel3D** (output_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells, das die Ebene beschreibt.

Ergebnis

`gen_plane_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`get_object_model_3d_params`, `sample_object_model_3d`, `clear_object_model_3d`

Siehe auch

`gen_cylinder_object_model_3d`, `gen_sphere_object_model_3d`, `gen_sphere_object_model_3d_center`, `gen_box_object_model_3d`

Modul

3D Metrology

gen_sphere_object_model_3d (: : Pose, Radius : ObjectModel3D)

Erzeugen eines 3D-Objektmodells, welches eine Kugel beschreibt.

`gen_sphere_object_model_3d` erzeugt ein kugelförmiges 3D Primitiv, d.h. ein 3D-Objektmodell, das eine Kugel beschreibt. Eine Kugel wird über ihren Mittelpunkt in `Pose` und ihren Radius in `Radius` beschrieben. Falls mehrere Kugeln erzeugt werden und die Anzahl der Werte des Parameters `Radius` 1 ist, werden alle Kugeln mit dem gleichen Radius erzeugt. Die Handle des resultierenden 3D-Objektmodells wird im Parameter `ObjectModel3D` zurückgegeben.

Parameter

▷ **Pose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
Die Lage des Mittelpunkts der Kugel im Raum.

▷ **Radius** (input_control) . . . . . number(-array) ⇝ *real*
Der Radius der Kugel.

▷ **ObjectModel3D** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells, das die Kugel beschreibt.

Ergebnis

`gen_sphere_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

smallest_sphere_object_model_3d

Nachfolger

get_object_model_3d_params, sample_object_model_3d, clear_object_model_3d

Alternativen

gen_sphere_object_model_3d_center

Siehe auch

gen_cylinder_object_model_3d, gen_plane_object_model_3d, gen_box_object_model_3d

Modul

3D Metrology

gen_sphere_object_model_3d_center (: : X, Y, Z, Radius : ObjectModel3D)

Erzeugen eines 3D-Objektmodells aus X-,Y-,Z-Koordinaten, welche eine Kugel beschreibt.

gen_sphere_object_model_3d_center erzeugt ein kugelförmiges 3D Primitiv, d.h. ein 3D-Objektmodell das eine Kugel beschreibt. Eine Kugel wird über ihren Mittelpunkt in X, Y und Z sowie ihren Radius in Radius beschrieben. Falls mehrere Kugeln erzeugt werden und die Anzahl der Werte des Parameters Radius 1 ist, werden alle Kugeln mit dem gleichen Radius erzeugt. Die Handle des resultierenden 3D-Objektmodells wird im Parameter ObjectModel3D zurückgegeben.

Parameter

- ▷ **X** (input_control) ..... point3d.x(-array) ⇝ *real* / integer
  Die X-Koordinate des Mittelpunkts der Kugel.
- ▷ **Y** (input_control) ..... point3d.y(-array) ⇝ *real* / integer
  Die Y-Koordinate des Mittelpunkts der Kugel.
- ▷ **Z** (input_control) ..... point3d.z(-array) ⇝ *real* / integer
  Die Z-Koordinate des Mittelpunkts der Kugel.
- ▷ **Radius** (input_control) ..... number(-array) ⇝ *real*
  Der Radius der Kugel.
- ▷ **ObjectModel3D** (output_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Objektmodells, das die Kugel beschreibt.

Ergebnis

gen_sphere_object_model_3d_center liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

smallest_sphere_object_model_3d

Nachfolger

get_object_model_3d_params, sample_object_model_3d, clear_object_model_3d

Alternativen

gen_sphere_object_model_3d

Siehe auch

gen_cylinder_object_model_3d, gen_plane_object_model_3d, gen_box_object_model_3d

Modul

3D Metrology

read_object_model_3d (: : FileName, Scale, GenParamName, GenParamValue : ObjectModel3D, Status)

Lesen eines 3D-Objektmodells aus einer Datei.

read_object_model_3d liest ein 3D-Objektmodell aus der Datei FileName und gibt das Handle des 3D-Objektmodells in ObjectModel3D zurück.

Der Operator unterstützt die folgenden Dateiformate:

'om3': HALCON-Format für 3D-Objektmodelle. Dateien mit diesem Format können mit write_object_model_3d geschrieben werden. Die Standard-Dateiendung für dieses Format ist 'om3'.

'dxf': AUTOCAD-Dateiformat. HALCON kann nur die ASCII-version des Formats lesen. Weitere Details über dieses Format finden sich weiter unten. Die Standard-Dateiendung für dieses Format ist 'dxf'.

'off': 'Object File Format'. Dies ist ein einfaches ASCII-basiertes Format, welches 3D-Punkte und Polygone speichern kann. Das binäre OFF-Format wird nicht unterstützt. Die Standard-Dateiendung für dieses Format ist 'off'.

'ply': 'Polygon File Format' (auch Stanford Triangle Format). Dies ist ein einfaches Dateiformat, welches 3D-Punkte, Punktnormalen, Polygone, Farbinformationen und punktbasierte benutzerdefinierte Attribute speichern kann. HALCON kann sowohl die ASCII- als auch die Binärversion des Formats lesen. Sollte die zu lesende Datei zusätzliche, nicht unterstützte Informationen beinhalten, so werden diese ignoriert und es werden nur die bekannten Attribute eingelesen. Die Standard-Dateiendung für dieses Format ist 'ply'.

'obj': 'OBJ file format' (auch 'Wavefront OBJ-Format'). Dies ist ein ASCII-basiertes Format, welches 3D-Punkte, Polygone, Normalen, Texturkoordinaten, Materialien und anderes speichern kann. HALCON unterstützt das Einlesen von Punkten ('v'-Zeilen) und Polygonen ('f'-Zeilen), andere Daten werden ignoriert. Die Standard-Dateiendung für dieses Format ist 'obj'.

'stl',

'stl_ascii',

'stl_binary': 'STL-Dateiformat' (auch 'Stereolithography', 'SurfaceTesselationLanguage', 'StandardTriangulationLanguage' oder 'StandardTesselationLanguage'). Dieses Dateiformat kann Dreiecke und Dreiecksnormalen speichern. Da das HALCON 3D-Objektmodell allerdings keine Dreiecksnormalen unterstützt, werden nur die Dreiecke eingelesen während die Dreiecksnormalen ignoriert werden. Die Normalen werden bei Bedarf aus den Dreiecken selbst berechnet. HALCON kann sowohl die ASCII- als auch die Binärversion des Formats lesen. Wird *'stl'* angegeben so erkennt HALCON automatisch ob die Datei in der ASCII- oder der Binärversion des Formats vorliegt. Mit *'stl_ascii'* bzw. *'stl_binary'* kann explizit das ASCII- bzw. Binärformat gewählt werden. Die Standard-Dateiendung für dieses Format ist 'stl'.

Wird eine DXF-Datei eingelesen, so enthält der Ausgabeparameter Status Informationen über die Anzahl der gelesenen 3D-Flächen und, falls erforderlich, Hinweise auf Teile der DXF-Datei, die nicht gelesen werden konnten.

Der Parameter Scale definiert die Dimension der in der Datei auftretenden Koordinaten. Wird der Parameter z.B. auf *'mm'* gesetzt, so werden alle Einheiten in *'mm'* interpretiert und durch Multiplikation mit *0.001* in die HALCON-übliche Einheit *'m'* umgewandelt. Ein Wert von *'100 mm'* wird so zu *'0.1 m'*. Alternativ zu einer Einheit kann für den Parameter auch direkt eine Zahl angegeben werden, die mit allen Koordinatenwerten multipliziert wird. Die Zuordnung von Einheiten zu Faktoren ergibt sich aus der folgenden Tabelle:

Einheit	Skalierungsfaktor
m	1
dm	0.1
cm	0.01
mm	0.001
um, microns	10^{-6}
nm	10^{-9}
km	1000
in	0.0254
ft	0.3048
yd	0.9144

Der Parameter `Scale` wird für Dateien vom Typ *'om3'* ignoriert, d.h. om3-Dateien werden immer unskaliert eingelesen. Zum nachträglichen Skalieren eines 3D-Objektmodells kann `affine_trans_object_model_3d` verwendet werden.

Eine Reihe weiterer optionaler Parameter kann gesetzt werden. Die Namen und Werte dieser Parameter werden in `GenParamName` und `GenParamValue` übergeben. Einige der Parameter gelten nicht für alle Dateitypen. Die folgenden Werte sind für `GenParamName` möglich:

'file_type': Erzwingen eines Dateityps. Wird dieser Parameter nicht gesetzt, so versucht `read_object_model_3d` den Dateityp automatisch aus der Dateiendung und dem Anfang der Datei zu erkennen. Wird dieser Parameter gesetzt, so wird die Datei als Datei diesen Types interpretiert.

Werteliste: 'om3', *'dxf'*, *'off'*, *'ply'*, *'obj'*, *'stl'*.

'convert_to_triangles': Wandelt alle Flächen in Dreiecke um. Ist dieser Parameter auf *'true'* gesetzt, so werden alle eingelesenen Polygone in Dreiecke konvertiert.

Werteliste: 'true', *'false'*.

Standardwert: 'false'.

Gültig für Dateitypen: 'dxf', *'ply'*, *'off'*, *'obj'*.

'invert_normals': Invertieren der Normalenrichtungen und Flächen. Ist dieser Parameter auf *'true'* gesetzt, so wird die Ausrichtung aller Normalen und Flächen invertiert.

Werteliste: 'true', *'false'*.

Standardwert: 'false'.

Gültig für Dateitypen: 'dxf', *'ply'*, *'off'*, *'obj'*, *'stl'*.

'max_approx_error', *'min_num_points'*: DXF-spezifische Parameter (siehe unten).

Gültig für Dateitypen: 'dxf'

'xyz_map_width': Erstellt für das gelesene 3D-Objektmodell eine Abbildung welche jedem der gelesenen 3D-Punkte eine Bildkoordinate zuweist, analog zu `xyz_to_object_model_3d`. Es wird angenommen, dass die 3D-Punkte in der Datei zeilenweise vorliegen. Der übergebene Wert wird als Breite des Bildes angenommen. Die nötige Höhe des Bildes wird automatisch ermittelt. Ist dieser Wert gesetzt, so kann das gelesene 3D-Objektmodell im Operator `object_model_3d_to_xyz` mit der Methode *'from_xyz_map'* projiziert werden. Nur einer der beiden Parameter *'xyz_map_width'* und *'xyz_map_height'* kann gesetzt werden.

Standardwert: -1.

Gültig für Dateitypen: 'ply', *'off'*, *'obj'*.

Restriktion: 'xyz_map_width' > 0

'xyz_map_height': Analog zu *'xyz_map_width'*, jedoch wird angenommen dass die Punkte spaltenweise vorliegen. Die Breite des Bildes wird automatisch ermittelt. Nur einer der beiden Parameter *'xyz_map_width'* und *'xyz_map_height'* kann gesetzt werden.

Standardwert: -1.

Gültig für Dateitypen: 'ply', *'off'*, *'obj'*.

Restriktion: 'xyz_map_height' > 0

Falls ein 2D-Mapping vorhanden ist, wird seine Verwendung auf Grund besserer Robustheit und höherer Verarbeitungsgeschwindigkeit empfohlen. Vor allem für die Operatoren `sample_object_model_3d` und

surface_normals_object_model_3d, sowie in der Vorbearbeitung eines 3D-Objektmodells für das oberflächenbasierte 3D-Matching (Glättung, Entfernung der Ausreißer, und Reduzieren der Domäne) ist daher die Verwendung des 2D-Mappings vorteilhaft.

Der Operator `read_object_model_3d` unterstützt die folgenden DXF-Elemente:

- POLYLINE
  - Vielflächennetz (Polyline flag 64)
  - 3D Linienzüge (Polyline flag 8,9)
  - 2D Linienzüge (Polyline flag 0)
- LWPOLYLINE
  - 2D Linienzüge (Polyline flag 0)
- 3DFACE
- LINE
- CIRCLE
- ARC
- SOLID
- BLOCK
- INSERT

Die zweidimensionalen linienhaften DXF-Elemente LINE, CICRLE und ARC werden nicht als Flächen interpretiert. Nur im Fall, dass sie extrudiert sind, werden die dadurch entstehenden Flächen in das 3D-Objektmodell eingefügt. Alle DXF-Elemente, die Linienzüge repräsentieren, werden als 3D Linien zum 3D-Objektmodell hinzugefügt.

Die gekrümmte Oberfläche extrudierter DXF-Elemente vom Typ CIRCLE und ARC wird durch ebene Flächen approximiert. Die Genauigkeit dieser Approximation kann über die beiden generischen Parameter *'min_num_points'* und *'max_approx_error'* gesteuert werden. Der Parameter *'min_num_points'* legt fest, wie viele Stützpunkte mindestens für die Approximation des DXF-Elements CIRCLE bzw. ARC verwendet werden. Der Parameter *'min_num_points'* bezieht sich immer auf den entsprechenden Vollkreis, auch für ARCs, d.h., wenn *'min_num_points'* auf den Wert *50* gesetzt ist und ein DXF-Element vom Typ ARC eingelesen wird, das einen Halbkreis darstellt, so wird dieser durch mindestens 25 Stützpunkte approximiert. Der Parameter *'max_approx_error'* legt fest, wie groß die maximale Abweichung der XLD-Kontur von dem idealen Kreis sein darf. Diese Abweichung wird in der Einheit bestimmt, die in der DXF-Datei verwendet wird. Für die Bestimmung der Genauigkeit der Approximation werden beide Kriterien ausgewertet und dasjenige verwendet, das zu einer genaueren Approximation führt.

Die intern verwendeten Defaultwerte für die generischen Parameter sind:

- *'min_num_points' = 20*
- *'max_approx_error' = 0.25*

Um eine genauere Approximation zu erreichen, muss entweder ein größerer Wert für *'min_num_points'* oder ein kleinerer Wert für *'max_approx_error'* verwendet werden.

Eine Möglichkeit, um eine passende DXF-Datei zu erzeugen, besteht darin, ein 3D-Modell des Objektes mit dem CAD-Programm AutoCAD zu erstellen. Dabei muss darauf geachtet werden, dass die Oberfläche des Objektes modelliert wird, nicht nur die Kanten. Linien, die z.B. Objektkanten darstellen, können von HALCON nicht verwendet werden, da sie nicht die Oberfläche des Objektes beschreiben. Wenn die Modellierung des Objektes abgeschlossen ist, kann das 3D-Modell im DWG-Format abgespeichert werden. Um die DWG-Datei in eine DXF-Datei zu konvertieren, die für die Verwendung in HALCON geeignet ist, sind die folgenden Schritte durchzuführen:

- Exportieren des 3D CAD-Modells in eine 3DS-Datei mit dem AutoCAD-Befehl 3dsout. Dabei erfolgt eine Dreiecksvermaschung der Oberfläche des Objektes, d.h., die Oberfläche wird nur noch durch Dreiecke dargestellt. (Nutzer von AutoCAD 2007 oder neueren Versionen können sich ein Dienstprogramm, das diesen Befehl zur Verfügung stellt, auf den Internetseiten von Autodesk herunterladen.)
- Öffnen einer neuen, leeren Zeichnung.
- Importieren der 3DS-Datei in die neue Zeichnung mit dem AutoCAD-Befehl 3dsin.

- Speichern des Modells in eine DXF R12-Datei.

Nutzer anderer CAD-Programme sollten sicherstellen, dass die Oberfläche des 3D-Modells vor dem Export nach DXF dreiecksvermascht wird. Wenn das CAD-Programm diese Funktionalität nicht zur Verfügung stellt, besteht oft die Möglichkeit, das 3D-Modell im proprietären Format des CAD-Programms abzuspeichern und anschließend mit einem Konverter für CAD-Formate, der die Dreiecksvermaschung durchführen kann, zu konvertieren.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname der zu lesenden Datei.
Defaultwert: 'mvtec_bunny_normals'
Wertevorschläge: `FileName` ∈ {'mvtec_bunny', 'glass_mug', 'bmc_mini', 'pipe_joint', 'clamp_sloped', 'tile_spacer'}
Dateiendung: `.off`, `.ply`, `.dxf`, `.om3`, `.obj`, `.stl`

▷ **Scale** (input_control) . . . . . number ⇝ *string* / real / integer
Skalierung der Daten in der Datei.
Defaultwert: 'm'
Wertevorschläge: `Scale` ∈ {'m', 'cm', 'mm', 'microns', 'um', 'nm', 'km', 'in', 'ft', 'yd', 1.0, 0.01, 0.001, 1.0e-6, 0.0254, 0.3048, 0.9144}

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'convert_to_triangles', 'invert_normals', 'file_type', 'min_num_points', 'max_approx_error', 'xyz_map_width', 'xyz_map_height'}

▷ **GenParamValue** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false', 1, 0, 'om3', 'off', 'ply', 'dxf', 'obj', 'stl', 'stl_binary', 'stl_ascii'}

▷ **ObjectModel3D** (output_control) . . . . . object_model_3d ⇝ *handle*
Handle des eingelesenen 3D-Objektmodells.

▷ **Status** (output_control) . . . . . string(-array) ⇝ *string*
Statusinformation.

Ergebnis

`read_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind und die Datei gelesen werden konnte. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`write_object_model_3d`

Nachfolger

`affine_trans_object_model_3d`, `object_model_3d_to_xyz`, `prepare_object_model_3d`

Alternativen

`xyz_to_object_model_3d`

Siehe auch

`write_object_model_3d`, `clear_object_model_3d`

Modul

3D Metrology

remove_object_model_3d_attrib (: : ObjectModel3D, Attributes : ObjectModel3DOut)

Entfernen von Attributen eines 3D-Objektmodells.

remove_object_model_3d_attrib kopiert das 3D-Objektmodell ObjectModel3D und entfernt in dieser Kopie die Standard-Attribute und/oder erweiterten Attribute in Attributes. Das neue 3D-Objektmodell wird in ObjectModel3DOut zurück geliefert. Auf diese Weise wird das 3D-Objektmodell ObjectModel3D nicht verändert sondern ein neues Modell erzeugt, im Gegensatz zum ansonsten identisch funktionierenden Operator remove_object_model_3d_attrib_mod, der das Eingabe-Modell modifiziert.

Falls die Attribute in Attributes nicht gesetzt sind, wird keine Fehlerbehandlung durchgeführt.

Die folgenden Werte für den Parameter Attributes sind möglich:

'point_normal': Dieser Wert legt fest, dass das Attribut mit den Punktnormalen entfernt wird.

'triangles': Dieser Wert legt fest, dass das Attribut mit den Oberflächendreiecken entfernt wird.

'polygons': Dieser Wert legt fest, dass das Attribut mit den Oberflächenpolygonen entfernt wird.

'lines': Dieser Wert legt fest, dass das Attribut mit den Linien entfernt wird.

'xyz_mapping': Dieser Wert legt fest, dass das Attribut mit der Zuordnung der Bildkoordinaten entfernt wird.

'extended_attribute': Dieser Wert legt fest, dass alle vom Nutzer definierten erweiterten Attribute entfernt werden.

'primitives_all': Dieser Wert legt fest, dass das Attribut mit den Parametern des 3D-Primitivs (eingeschlossen leerer 3D-Primitive) entfernt wird (z.B. generiert vom Operator fit_primitives_object_model_3d).

'primitive_plane': Dieser Wert legt fest, dass das Attribut mit einem 3D-Primitiv "Ebene"entfernt wird (z.B. generiert vom Operator fit_primitives_object_model_3d).

'primitive_sphere': Dieser Wert legt fest, dass das Attribut mit einem 3D-Primitiv "Kugel"entfernt wird (z.B. generiert vom Operator fit_primitives_object_model_3d)

'primitive_cylinder': Dieser Wert legt fest, dass das Attribut mit einem 3D-Primitiv "Zylinder"entfernt wird (z.B. generiert vom Operator fit_primitives_object_model_3d).

'primitive_box': Dieser Wert legt fest, dass das Attribut mit einem 3D-Primitiv "Quader"entfernt wird.

'shape_based_matching_3d_data': Dieser Wert legt fest, dass das Attribut mit dem vorbereiteten 3D-Formmodell für das formbasierte 3D-Matching entfernt wird.

'distance_computation_data': Dieser Wert legt fest, dass das Attribut mit der Datenstruktur für die 3D Distanzberechnung entfernt wird. Diese Datenstruktur kann mit prepare_object_model_3d erstellt und mit distance_object_model_3d verwendet werden.

'score': Dieser Wert legt fest, dass die Daten für das Qualitätsmaß entfernt werden. Das Qualitätsmaß wird in der Regel durch den Operator reconstruct_surface_stereo berechnet.

'red': Dieser Wert legt fest, dass die Daten für den roten Kanal entfernt werden.

'green': Dieser Wert legt fest, dass die Daten für den grünen Kanal entfernt werden.

'blue': Dieser Wert legt fest, dass die Daten für den blauen Kanal entfernt werden.

'original_point_indices': Dieser Wert legt fest, dass das Attribut mit den originalen Punktindizes entfernt werden. Die originalen Punktindizes können vom Operator triangulate_object_model_3d generiert werden.

'all': Dieser Wert legt fest, dass alle verfügbaren Attribute außer die Punktkoordinaten entfernt werden. Die Attribute sind die Punktnormalen, die Oberflächendreiecke, die Oberflächenpolygone, das Mapping, das Formmodell für das Matching, die Parameter eines 3D-Primitivs und die erweiterten Attribute.

Erweiterte Attribute sind Attribute, die aus Standard-Attributen von speziellen Operatoren (z.B. distance_object_model_3d) berechnet werden oder benutzerdefinierte Attribute (gesetzt mit set_object_model_3d_attrib oder set_object_model_3d_attrib_mod). Die erweiterten Attribute können entfernt werden, indem ihre Namen in Attributes gesetzt werden.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des Eingabe-3D-Objektmodells.

▷ **Attributes** (input_control) . . . . . string(-array) ⇝ *string*
Namen der zu entfernenden Attribute.
Defaultwert: 'extended_attribute'
Werteliste: `Attributes` ∈ {'point_normal', 'triangles', 'lines', 'polygons', 'xyz_mapping', 'shape_based_matching_3d_data', 'distance_computation_data', 'primitives_all', 'primitive_plane', 'primitive_sphere', 'primitive_cylinder', 'primitive_box', 'extended_attribute', 'score', 'red', 'green', 'blue', 'original_point_indices', 'all'}

▷ **ObjectModel3DOut** (output_control) . . . . . object_model_3d ⇝ *handle*
Handle des Ausgabe-3D-Objektmodells.

Ergebnis

`remove_object_model_3d_attrib` liefert den Wert TRUE wenn alle Parameter korrekt sind. Falls notwendig, wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`set_object_model_3d_attrib`

Nachfolger

`get_object_model_3d_params`

Alternativen

`remove_object_model_3d_attrib_mod`

Siehe auch

`copy_object_model_3d`, `set_object_model_3d_attrib`

Modul

3D Metrology

remove_object_model_3d_attrib_mod (: : ObjectModel3D,
Attributes :)

Entfernen von Attributen eines 3D-Objektmodells.

`remove_object_model_3d_attrib_mod` entfernt Standard-Attribute und/oder erweiterte Attribute in `Attributes` des 3D-Objektmodells `ObjectModel3D`. Auf diese Weise wird das 3D-Objektmodell verändert, im Gegensatz zum ansonsten identisch funktionierenden Operator `remove_object_model_3d_attrib`, der ein neues Modell erzeugt.

Falls die Attribute in `Attributes` nicht gesetzt sind, wird keine Fehlerbehandlung durchgeführt.

Für eine detaillierte Beschreibung der `Attributes` siehe `remove_object_model_3d_attrib`.

Achtung

`remove_object_model_3d_attrib_mod` entfernt ohne weitere Überprüfung die Attribute `Attributes` vom 3D-Objektmodell. Daher ist auf Konsistenz der Eingabedaten zu achten, da die meisten Operatoren konsistente 3D-Objektmodelle erwarten. Weiterhin sollte die Zuordnung der 3D Punkte zu Bildkoordinaten nicht entfernt werden, da sie die Berechnungen vieler Operatoren beschleunigen kann.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des Eingabe-3D-Objektmodells.

▷ **Attributes** (input_control) . . . . . string(-array) ⇝ *string*
Namen der zu entfernenden Attribute.
Defaultwert: 'extended_attribute'
Werteliste: `Attributes` ∈ {'point_normal', 'triangles', 'lines', 'polygons', 'xyz_mapping', 'shape_based_matching_3d_data', 'distance_computation_data', 'primitives_all', 'primitive_plane', 'primitive_sphere', 'primitive_cylinder', 'primitive_box', 'extended_attribute', 'score', 'red', 'green', 'blue', 'original_point_indices', 'all'}

Ergebnis

`remove_object_model_3d_attrib_mod` liefert den Wert TRUE wenn alle Parameter korrekt sind. Falls notwendig, wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`set_object_model_3d_attrib_mod`

Nachfolger

`get_object_model_3d_params`

Alternativen

`remove_object_model_3d_attrib`

Siehe auch

`copy_object_model_3d`, `set_object_model_3d_attrib_mod`

Modul

3D Metrology

serialize_object_model_3d (: : ObjectModel3D : SerializedItemHandle)

Serialisieren eines 3D-Objektmodells.

`serialize_object_model_3d` serialisiert die Daten eines 3D-Objektmodells (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_object_model_3d` in eine Datei im Dateiformat 'om3' geschrieben werden. Das 3D-Objektmodell wird in dem Handle `ObjectModel3D` übergeben. Das serialisierte 3D-Objektmodell wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_object_model_3d` wieder deserialisiert werden.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_object_model_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d

Nachfolger

read_object_model_3d, fwrite_serialized_item, send_serialized_item, deserialize_object_model_3d

Siehe auch

read_object_model_3d

Modul

3D Metrology

set_object_model_3d_attrib (: : ObjectModel3D, AttribName, AttachExtAttribTo, AttribValues : ObjectModel3DOut)

Setzen von Attributen eines 3D-Objektmodells.

set_object_model_3d_attrib_mod setzt Standard-Attribute und erweiterte Attribute in AttribName eines 3D-Objektmodells ObjectModel3D auf die Werte in AttribValues und liefert ein 3D-Objektmodell mit den neuen Attributwerten in ObjectModel3DOut zurück. set_object_model_3d_attrib ist identisch mit set_object_model_3d_attrib_mod mit der Ausnahme, dass set_object_model_3d_attrib ein neues 3D-Objektmodell erzeugt, anstatt das bestehende zu verändern. Es ist über den Parameter AttachExtAttribTo möglich, die Werte von erweiterten Attributen an bereits existierende Standard-Attribute des 3D-Objektmodells zu binden. Für Standard-Attribute wird AttachExtAttribTo ignoriert.

Falls die Attribute in AttribName nicht existieren, werden sie wenn möglich neu erstellt. Sollen bereits im 3D-Objektmodell vorhandene Attribute gesetzt werden, muss die Länge der neuen Daten der Länge der alten Daten entsprechen. Ist dies der Fall, werden die alten Daten überschrieben. Falls erweiterte Attribute mit AttachExtAttribTo an bereits existierende Standard-Attribute gehängt werden sollen, muss die Länge der neuen Daten den bereits vorhandenen Standard-Attribut-Werten entsprechen.

Standardattribute

Die folgenden Standardattribute können gesetzt werden:

'point_coord_x': Die x-Koordinaten einer 3D-Punktmenge werden im Parameter AttribValues übergeben. Ist das Attribut nicht vorhanden, müssen die x-, y- und z-Koordinaten gleichzeitig mittels *'point_coord_x'*, *'point_coord_y'* und *'point_coord_z'* gesetzt werden. Die Anzahl der x-, y- und z-Koordinaten muss identisch sein.

'point_coord_y': Die y-Koordinaten einer 3D-Punktmenge werden im Parameter AttribValues übergeben. Ist das Attribut nicht vorhanden, müssen die x-, y- und z-Koordinaten gleichzeitig mittels *'point_coord_x'*, *'point_coord_y'* und *'point_coord_z'* gesetzt werden. Die Anzahl der x-, y- und z-Koordinaten muss identisch sein.

'point_coord_z': Die z-Koordinaten einer 3D-Punktmenge werden im Parameter AttribValues übergeben. Ist das Attribut nicht vorhanden, müssen die x-, y- und z-Koordinaten gleichzeitig mittels *'point_coord_x'*, *'point_coord_y'* und *'point_coord_z'* gesetzt werden. Die Anzahl der x-, y- und z-Koordinaten muss identisch sein.

'point_normal_x': Die x-Komponenten der 3D-Punktnormalen einer 3D-Punktmenge werden im Parameter AttribValues übergeben. Ist das Attribut nicht vorhanden, müssen die x-, y- und z-Komponenten der 3D-Punktnormalen gleichzeitig mittels *'point_normal_x'*, *'point_normal_y'* und *'point_normal_z'* gesetzt werden. Die Anzahl der x-, y- und z-Komponenten der Anzahl an vorhandenen 3D-Punkten entsprechen. Die 3D-Punktnormalen werden nicht auf die Länge 1 normiert.

'point_normal_y': Die y-Komponenten der 3D-Punktnormalen einer 3D-Punktmenge werden im Parameter AttribValues übergeben. Ist das Attribut nicht vorhanden, müssen die x-, y- und z-Komponenten der 3D-Punktnormalen gleichzeitig mittels *'point_normal_x'*, *'point_normal_y'* und *'point_normal_z'* gesetzt werden. Die Anzahl der x-, y- und z-Komponenten muss der Anzahl an vorhandenen 3D-Punkten entsprechen. Die 3D-Punktnormalen werden nicht auf die Länge 1 normiert.

'point_normal_z': Die z-Komponenten der 3D-Punktnormalen einer 3D-Punktmenge werden im Parameter AttribValues übergeben. Ist das Attribut nicht vorhanden, müssen die x-, y- und z-Komponenten der

3D-Punktnormalen gleichzeitig mittels *'point_normal_x'*, *'point_normal_y'* und *'point_normal_z'* gesetzt werden. Die Anzahl der x-, y- und z-Komponenten muss der Anzahl an vorhandenen 3D-Punkten entsprechen. Die 3D-Punktnormalen werden nicht auf die Länge 1 normiert.

'score': Das Qualitätsmaß der Rekonstruktion einer 3D-Punktmenge wird im Parameter `AttribValues` übergeben. Da dieses Attribut pro 3D-Punkt gesetzt wird, muss die Anzahl der Komponente der Anzahl an vorhandenen 3D-Punkten entsprechen.

'red': Die Intensitätswerte für den Rotkanal einer 3D-Punktmenge werden im Parameter `AttribValues` übergeben. Die Anzahl Werte muss der Anzahl an vorhandenen 3D-Punkten entsprechen.

'green': Die Intensitätswerte für den Grünkanal einer 3D-Punktmenge werden im Parameter `AttribValues` übergeben. Die Anzahl Werte muss der Anzahl an vorhandenen 3D-Punkten entsprechen.

'blue': Die Intensitätswerte für den Blaukanal einer 3D-Punktmenge werden im Parameter `AttribValues` übergeben. Die Anzahl Werte muss der Anzahl an vorhandenen 3D-Punkten entsprechen.

'triangles': Die Indizes der 3D-Punkte, die Dreiecke repräsentieren, in der folgenden Reihenfolge: Die ersten drei Werte des Parameters `AttribValues` (Eingabewerte 0,1,2) stellen das erste Dreieck dar, die nächsten drei Werte (Eingabewerte 3,4,5) stellen das zweite Dreieck dar, usw. Alle Indizes entsprechen schon vorhandenen 3D-Punkten. Zu beachten ist, dass sich die Richtung der Dreiecke aus der Reihenfolge der Indizes der Punkte ergibt.

'polygons': Die Indizes der 3D-Punkte, die Polygone repräsentieren, in der folgenden Reihenfolge: Der erste Wert des Parameters `AttribValues` enthält die Anzahl n der 3D-Punkte des ersten Polygons. Die folgenden Werte (Eingabewerte 1,2,..,n) repräsentieren das erste Polygon. Der nächste Wert (Eingabewert n+1) enthält die Anzahl m der 3D-Punkte des zweiten Polygons. Die folgenden Werte (Eingabewerte n+2,n+3,..,n+1+m) repräsentieren das zweite Polygon, usw. Alle Indizes entsprechen schon vorhandenen 3D-Punkten.

'lines': Die Indizes der 3D-Punkte, die Linienzüge repräsentieren, in der folgenden Reihenfolge: Der erste Wert des Parameters `AttribValues` enthält die Anzahl n der 3D-Punkte des ersten Linienzuges. Die folgenden Werte (Eingabewerte 1,2,..,n) repräsentieren den ersten Linienzug. Der nächste Wert (Eingabewert n+1) enthält die Anzahl m der 3D-Punkte des zweiten Linienzuges. Die folgenden Werte (Eingabewerte n+2,n+3,..,n+1+m) repräsentieren den zweiten Linienzug usw. Alle Indizes entsprechen schon vorhandenen 3D-Punkten.

'xyz_mapping': Die Zuordnung der Bildkoordinaten zu den 3D-Punkten, in der folgenden Reihenfolge: Der erste Wert (Eingabewert 0) des Parameters `AttribValues` enthält die Breite des entsprechenden Bildes. Der zweite Wert (Eingabewert 1) enthält die Höhe des entsprechenden Bildes. Die folgenden n Werte (Eingabewerte 2,3,..,n+1, mit n der Anzahl von 3D-Punkten) repräsentieren die Zeilenkoordinaten der n Punkte in Bildkoordinaten. Die nächsten n Werte (Eingabewerte n+2,n+3,..,n*2+1) repräsentieren die Spaltenkoordinaten der n Punkte in Bildkoordinaten. Daraus ergibt sich die Gesamtanzahl der Eingabewerte von n*2+2.

'original_point_indices': Die originalen Punktindizes der 3D-Punkte werden mit `AttribValues` gesetzt. Die Anzahl an originalen Punktindizes muss mit der Anzahl an 3D-Punkten übereinstimmen.

Erweiterte Attribute

Erweiterte Attribute sind Attribute, die aus Standard-Attributen von speziellen Operatoren (z.B. `distance_object_model_3d`) berechnet werden oder benutzerdefinierte Attribute. Die Namen von erweiterten Attributen sind frei wählbar, müssen aber mit dem Präfix *'&'* beginnen, also z.B., *'&my_attrib'*. Erweiterte Attribute können eine beliebige Anzahl von Fließkommawerten annehmen.

Erweiterte Attribute können über den Parameter `AttachExtAttribTo` an bereits existierende Standard-Attribute des 3D-Objektmodells gebunden werden. Die folgenden Werte für `AttachExtAttribTo` sind möglich:

'object' **oder** *[]*: Ist dieser Wert gesetzt, wird das erweiterte Attribut an das 3D-Objektmodell als Ganzes gebunden. Die Anzahl der Werte in `AttribValues` ist nicht eingeschränkt.

'points': Ist dieser Wert gesetzt, wird das erweiterte Attribut an die 3D-Punkte des 3D-Objektmodells gebunden. Die Anzahl der Werte in `AttribValues` muss gleich der Anzahl der bereits bestehenden 3D-Punkte sein.

'triangles': Ist dieser Wert gesetzt, wird das erweiterte Attribut an die Dreiecke des 3D-Objektmodells gebunden. Die Anzahl der Werte in `AttribValues` muss gleich der Anzahl der bereits bestehenden Dreiecke sein.

'polygons': Ist dieser Wert gesetzt, wird das erweiterte Attribut an die Polygone des 3D-Objektmodells gebunden. Die Anzahl der Werte in `AttribValues` muss gleich der Anzahl der bereits bestehenden Polygone sein.

'lines': Ist dieser Wert gesetzt, wird das erweiterte Attribut an die Linienzüge des 3D-Objektmodells gebunden. Die Anzahl der Werte in AttribValues muss gleich der Anzahl der bereits bestehenden Linienzüge sein.

Achtung

Falls mehrere Attribute in AttribName übergeben werden, wird AttribValues in gleich lange Teiltupel unterteilt. Jedes Teiltupel wird dann einem Attribut zugeordnet. Z.B.: Wenn AttribName und AttribValues folgende Werte haben:

AttribName := *['&attrib1','&attrib2','&attrib3']*,

AttribValues := *[0.0,1.0,2.0,3.0,4.0,5.0]*,

werden die Attribute mit folgenden Werten gesetzt:

'&attrib1' = [0.0,1.0], *'&attrib2' = [2.0,3.0]*, *'&attrib3' = [4.0,5.0]*.

Folglich ist es nicht möglich, mehrere Attribute unterschiedlicher Länge in einem Aufruf zu setzen.

set_object_model_3d_attrib speichert die Eingabedaten AttribValues unverändert im 3D-Objektmodell. Daher ist auf Konsistenz der Eingabedaten zu achten, da die meisten Operatoren konsistente 3D-Objektmodelle erwarten.

Parameter

- ▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
  Handle des Eingabe-3D-Objektmodells.
- ▷ **AttribName** (input_control) . . . . . string(-array) ⇝ *string*
  Namen der Attribute.
  Werteliste: AttribName ∈ {'point_coord_x', 'point_coord_y', 'point_coord_z', 'point_normal_x', 'point_normal_y', 'point_normal_z', 'triangles', 'polygons', 'lines', 'xyz_mapping', 'red', 'green', 'blue', 'score', 'original_point_indices'}
- ▷ **AttachExtAttribTo** (input_control) . . . . . string ⇝ *string*
  Beschreibt, woran ein erweitertes Attribut gehängt werden soll.
  Defaultwert: []
  Werteliste: AttachExtAttribTo ∈ {[], 'object', 'points', 'polygons', 'triangles', 'lines'}
- ▷ **AttribValues** (input_control) . . . . . real(-array) ⇝ *real* / integer
  Attribut-Werte.
- ▷ **ObjectModel3DOut** (output_control) . . . . . object_model_3d ⇝ *handle*
  Handle des Ausgabe-3D-Objektmodells.

Ergebnis

set_object_model_3d_attrib liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_empty_object_model_3d

Nachfolger

get_object_model_3d_params

Alternativen

set_object_model_3d_attrib_mod

Siehe auch

copy_object_model_3d, remove_object_model_3d_attrib

Modul

3D Metrology

set_object_model_3d_attrib_mod (: : ObjectModel3D, AttribName, AttachExtAttribTo, AttribValues :)

Setzen von Attributen eines 3D-Objektmodells.

set_object_model_3d_attrib_mod setzt Standard-Attribute und erweiterte Attribute in AttribName, eines 3D-Objektmodells ObjectModel3D auf die Werte in AttribValues. set_object_model_3d_attrib_mod ist identisch mit set_object_model_3d_attrib mit der Ausnahme, dass set_object_model_3d_attrib_mod kein neues 3D-Objektmodell erzeugt, sondern das bestehende verändert. Es ist über den Parameter AttachExtAttribTo möglich, die Werte von erweiterten Attributen an bereits existierende Standard-Attribute des 3D-Objektmodells zu binden. Für Standard-Attribute wird AttachExtAttribTo ignoriert.

Falls die Attribute in AttribName nicht existieren, werden sie wenn möglich neu erstellt. Sollen bereits im 3D-Objektmodell vorhandene Attribute gesetzt werden, muss die Länge der neuen Daten der Länge der alten Daten entsprechen. Ist dies der Fall, werden die alten Daten überschrieben. Falls erweiterte Attribute mit AttachExtAttribTo an bereits existierende Standard-Attribute gehängt werden sollen, muss die Länge der neuen Daten den bereits vorhandenen Standard-Attribut-Werten entsprechen.

Für eine detaillierte Beschreibung siehe set_object_model_3d_attrib.

Achtung

Falls mehrere Attribute in AttribName übergeben werden, wird AttribValues in gleich lange Teiltupel unterteilt. Jedes Teiltupel wird dann einem Attribut zugeordnet. Z.B.: Wenn AttribName und AttribValues folgende Werte haben:

AttribName := *['&attrib1','&attrib2','&attrib3']*,

AttribValues := *[0.0,1.0,2.0,3.0,4.0,5.0]*,

werden die Attribute mit folgenden Werten gesetzt:

'&attrib1' = [0.0,1.0], *'&attrib2' = [2.0,3.0]*, *'&attrib3' = [4.0,5.0]*.

Folglich ist es nicht möglich, mehrere Attribute unterschiedlicher Länge in einem Aufruf zu setzen.

set_object_model_3d_attrib_mod verändert den Inhalt eines bereits bestehenden 3D-Objektmodells. Die Eingabedaten AttribValues werden unverändert im 3D-Objektmodell gespeichert. Daher ist auf Konsistenz der Eingabedaten zu achten, da die meisten Operatoren konsistente 3D-Objektmodelle erwarten.

Parameter

- ▷ **ObjectModel3D** (input_control) ..... object_model_3d ⇝ *handle*
  Handle des 3D-Objektmodells.
- ▷ **AttribName** (input_control) ..... string(-array) ⇝ *string*
  Namen der Attribute.
  Werteliste: AttribName ∈ {'point_coord_x', 'point_coord_y', 'point_coord_z', 'point_normal_x', 'point_normal_y', 'point_normal_z', 'triangles', 'polygons', 'lines', 'xyz_mapping', 'red', 'green', 'blue', 'score', 'original_point_indices'}
- ▷ **AttachExtAttribTo** (input_control) ..... string ⇝ *string*
  Beschreibt, woran ein erweitertes Attribut gehängt werden soll.
  Defaultwert: []
  Werteliste: AttachExtAttribTo ∈ {[], 'object', 'points', 'polygons', 'triangles', 'lines'}
- ▷ **AttribValues** (input_control) ..... real(-array) ⇝ *real* / integer
  Attribut-Werte.

Ergebnis

set_object_model_3d_attrib_mod liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_empty_object_model_3d

Nachfolger

get_object_model_3d_params

Alternativen

set_object_model_3d_attrib

Siehe auch

copy_object_model_3d, remove_object_model_3d_attrib_mod

Modul

3D Metrology

union_object_model_3d (: : ObjectModels3D,
Method : UnionObjectModel3D)

Kombinieren mehrerer 3D-Objektmodelle zu einem neuen 3D-Objektmodell.

union_object_model_3d kombiniert alle 3D-Objektmodelle in ObjectModels3D zu einem einzigen 3D-Objektmodell, das in UnionObjectModel3D zurückgegeben wird.

Sollten Bereiche der angegebenen 3D-Objektmodelle überlappen, kann das bei einer resultierenden 2D Abbildung der 3D Punkte, sowie bei Polygonen und Dreiecken zu Ergebnissen führen, die nicht weiter in vollem Umfang verwendbar sind, da diese auch überlappend sein könnten.

Die momentan einzige unterstützte Methode in Method ist bisher *'points_surface'*, die alle Punkte, Oberflächen und Linien im Ausgabemodell UnionObjectModel3D kombiniert. Erweiterte Attribute werden kopiert, soweit keine Lücken entstehen, d.h. wenn sie in allen Eingabeobjektmodellen vorhanden sind, in denen das Standard-Attribut, an das sie gebunden sind, vorhanden ist.

Achtung

union_object_model_3d ignoriert 3D-Objektmodelle vom Typ 3D-Primitiv und 3D-Formmodell.

Parameter

- ▷ **ObjectModels3D** (input_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle der Eingabe-3D-Objektmodelle.
- ▷ **Method** (input_control) ..... string ⇝ *string*
  Methode, die für die Vereinigung benutzt wird.
  Defaultwert: 'points_surface'
  Werteliste: Method ∈ {'points_surface'}
- ▷ **UnionObjectModel3D** (output_control) ..... object_model_3d ⇝ *handle*
  Handle des resultierenden 3D-Objektmodells.

Beispiel

```
gen_object_model_3d_from_points ([0,0,0,0],[1,1,0,0], [0,1,1,0],\
                                 ObjectModel3D1)
gen_object_model_3d_from_points ([1,1,1,1],[1,1,0,0], [0,1,1,0],\
                                 ObjectModel3D2)
get_object_model_3d_params (ObjectModel3D1, 'diameter', DiameterOld)
union_object_model_3d ([ObjectModel3D1,ObjectModel3D2], 'points_surface',\
                       UnionObjectModel3D)
get_object_model_3d_params (UnionObjectModel3D, 'diameter', DiameterNew)
```

Ergebnis

union_object_model_3d liefert den Wert TRUE wenn alle Parameter korrekt sind. Sollte es kein gemeinsames Attribut in der Eingabe geben, wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

get_object_model_3d_params

Nachfolger

connection_object_model_3d, convex_hull_object_model_3d

Siehe auch

gen_box_object_model_3d, gen_sphere_object_model_3d, gen_cylinder_object_model_3d

Modul

3D Metrology

write_object_model_3d (: : ObjectModel3D, FileType, FileName, GenParamName, GenParamValue :)

Schreiben eines 3D-Objektmodells in eine Datei.

write_object_model_3d schreibt das 3D-Objektmodell ObjectModel3D in die Datei FileName. Die Datei kann mittels read_object_model_3d wieder eingelesen werden oder in einem entsprechenden CAD-Programm geladen werden. Es ist zu beachten, dass Primitive nur in dem HALCON-Format *'om3'* gespeichert werden. Falls die Notwendigkeit besteht, die Primitive in einem anderen Format zu speichern, muss zuvor sample_object_model_3d aufgerufen werden. Dies führt allerdings zu einer Umwandlung der Primitive in durch 3D-Punkte abgetastete Objekte, welche einer Annäherung an die Primitive entsprechen.

Alle in die Datei geschriebenen Koordinaten werden in Metern geschrieben. Wird die Datei daher später mit read_object_model_3d wieder eingelesen, so muss der Parameter Scale auf *'m'* gesetzt werden, um eine Skalierung der Daten zu vermeiden.

Der Parameter FileType legt das Format der zu schreibenden Datei fest. Die folgenden Dateitypen werden von write_object_model_3d unterstützt:

'om3': HALCON-Format für 3D-Objektmodelle. Dateien mit diesem Format können mit read_object_model_3d wieder gelesen werden. Die Standard-Dateiendung für dieses Format ist 'om3'.

'dxf': AUTOCAD-Dateiformat. Weitere Details zu dem Format finden sich in der Dokumentation zu read_object_model_3d. Die Standard-Dateiendung für dieses Format ist 'dxf'.

'off': 'Object File Format'. Dies ist ein einfaches ASCII-basiertes Format welches 3D-Punkte und Polygone speichern kann. Die Standard-Dateiendung für dieses Format ist 'off'.

'ply',

'ply_binary': 'Polygon File Format' (auch Stanford Triangle Format). Dies ist ein einfaches Dateiformat welches 3D-Punkte, Punktnormalen, Polygone, Farbinformationen und punktbasierte benutzerdefinierte Attribute speichern kann. HALCON schreibt sowohl die ASCII- als auch die Binärversion des Dateiformats. Die Standard-Dateiendung für dieses Format ist 'ply'.

'obj': 'OBJ File Format' (auch 'Wavefront OBJ-Format'). Dies ist ein ASCII-basiertes Format, welches 3D-Punkte, Polygone, Normalen und Dreiecke, welche in Polygone umgewandelt werden, speichern kann. Die Standard-Dateiendung für dieses Format ist 'obj'.

'stl',

'stl_binary',

'stl_ascii': 'STL-Dateiformat' (auch 'Stereolithography', 'SurfaceTesselationLanguage', 'StandardTriangulationLanguage' oder 'StandardTesselationLanguage'). Dieses Dateiformat kann Dreiecke und Dreiecksnormalen speichern. Da das HALCON 3D Objektmodell allerdings keine Dreiecksnormalen unterstützt und Punktnormalen (wie sie z.B. mit surface_normals_object_model_3d berechnet werden) vom STL-Format nicht unterstützt werden, werden keine Normalen in die Datei geschrieben. Enthält das 3D Objektmodell Polygone, werden diese vor dem Speichern in Dreiecke umgewandelt. STL-Dateien können im ASCII oder im Binärformat geschrieben werden. Wird *'stl'* oder *'stl_binary'* angegeben, so wird das Binärformat geschrieben, für *'stl_ascii'* das ASCII-Format. Die Standard-Dateiendung für dieses Format ist 'stl'.

Eine Reihe weiterer optionaler Parameter kann gesetzt werden. Die Namen und Werte dieser Parameter werden in `GenParamName` und `GenParamValue` übergeben. Einige der Parameter gelten nicht für alle Dateitypen. Die folgenden Werte sind für `GenParamName` möglich:

'invert_normals': Invertieren der Normalenrichtung und Flächenausrichtung vor dem Speichern des 3D-Objektmodells. Ist dieser Parameter auf *'true'* gesetzt, so wird für die Formate *'off'*, *'ply'*, *'obj'* und *'stl'* der Umlaufsinn aller Flächen (Dreiecke und Polygone) invertiert. Für Formate, die Punktnormalen speichern (*'ply'*,*'obj'*), werden zusätzlich die Normalenrichtungen vor dem Speichern invertiert. Für die Dateitypen *'om3'* und *'dxf'* hat der Parameter keinen Effekt.

Werteliste: 'true', 'false'.

Standardwert: 'false'.

Gültig für Dateitypen: 'off', 'ply', 'obj', 'stl'.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des zu schreibenden 3D-Objektmodells.

▷ **`FileType`** (input_control) . . . . . string ⇝ *string*
Typ der zu schreibenden Datei.
Defaultwert: 'om3'
Werteliste: `FileType` ∈ {'off', 'ply', 'ply_binary', 'dxf', 'om3', 'obj', 'stl', 'stl_binary', 'stl_ascii'}

▷ **`FileName`** (input_control) . . . . . filename.write ⇝ *string*
Dateiname der zu schreibenden Datei.
Dateiendung: `.off`, `.ply`, `.dxf`, `.om3`, `.obj`, `.stl`

▷ **`GenParamName`** (input_control) . . . . . string(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'invert_normals'}

▷ **`GenParamValue`** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false'}

Ergebnis

`write_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind und die Datei geschrieben werden konnte. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`read_object_model_3d`

Siehe auch

`read_object_model_3d`

Modul

3D Metrology

4.2 Merkmale

`area_object_model_3d` `( : : ObjectModel3D : Area )`

Berechnen der Oberfläche von vermaschten 3D-Objektmodellen.

area_object_model_3d berechnet die Oberflächen von vermaschten 3D-Objektmodellen. Die Handle der 3D-Objektmodelle werden in ObjectModel3D übergeben. Die berechneten Oberflächen werden in Area zurückgegeben.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle der 3D-Objektmodelle.

▷ **Area** (output_control) . . . . . number(-array) ⇝ *real*
Die berechneten Flächen.
Parameteranzahl: Area == ObjectModel3D

Beispiel

```
gen_box_object_model_3d ([0,0,0,0,0,0,0],3,2,1, ObjectModel3D)
convex_hull_object_model_3d (ObjectModel3D, ObjectModel3DConvexHull)
area_object_model_3d (ObjectModel3DConvexHull, Area)
```

Ergebnis

area_object_model_3d liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

connection_object_model_3d, select_points_object_model_3d, prepare_object_model_3d, convex_hull_object_model_3d

Nachfolger

select_object_model_3d

Siehe auch

volume_object_model_3d_relative_to_plane, max_diameter_object_model_3d, moments_object_model_3d

Modul

3D Metrology

distance_object_model_3d (: : ObjectModel3DFrom, ObjectModel3DTo, Pose, MaxDistance, GenParamName, GenParamValue :)

Bestimmt die Abstände der Punkte eines 3D-Objektmodells zu einem anderen 3D-Objektmodell.

Der Operator distance_object_model_3d bestimmt die Abstände der Punkte im 3D-Objektmodell ObjectModel3DFrom zu den Punkten, Dreiecken oder dem Primitiv im 3D-Objektmodell ObjectModel3DTo. Die Abstände werden im 3D-Objektmodell ObjectModel3DFrom als erweitertes Attribut mit dem Namen *'&distance'* gespeichert. Die Abstände können danach mittels get_object_model_3d_params abgerufen werden oder mittels select_points_object_model_3d oder anderen Operatoren, die erweiterte Attribute verwenden, weiterverarbeitet werden.

Die Zieldaten (Punkte, Dreiecke oder Primitiv) werden anhand der in ObjectModel3DTo vorhandenen Daten ausgewählt. Dabei werden die ersten vorhandenen Daten in der folgenden Reihenfolge verwendet: Primitiv, Dreiecke, Punkte. Alternativ zu dieser automatischen Auswahl können die Zieldaten auch manuell mit dem generischen Parameter *'distance_to'* gesetzt werden (s.u.). Allgemeine Polygone werden intern trianguliert bevor der Abstand zu den Dreiecken berechnet wird. In diesem Sinne ist ein Operator-Aufruf mit schon triangulierten Objekten schneller als einer für Objekte mit allgemeinen Polygonen.

MaxDistance kann benutzt werden, um einen Schwellenwert für die berechneten Distanzen zu setzen. Wenn MaxDistance auf *0* gesetzt wird, werden alle Distanzen berechnet. Wenn MaxDistance auf einen anderen

Wert gesetzt wird, werden alle Punkte, bei denen die Distanz MaxDistance überschreiten würde, nicht verarbeitet. Für diese Punkte wird der Wert von MaxDistance in dem Attribut *'&distance'* gespeichert. Das Setzen von MaxDistance kann also die Geschwindigkeit des Operators deutlich erhöhen, wenn ein Wert ungleich *0* eingestellt wird.

Ist Pose ein nicht-leeres Tupel, so gibt es eine Pose an, welche vor der Distanzbestimmung auf die Punkte von ObjectModel3DFrom angewandt wird. Die Pose kann über den generischen Parameter *'invert_pose'* invertiert werden (s.u.).

Je nach Zieldatentyp (Punkte, Dreiecke oder Primitiv) gibt es verschiedene Methoden zur Abstandsbestimmung. Einige dieser Methoden bauen eine Datenstruktur über den Elementen von ObjectModel3DTo auf, welche die Berechnung beschleunigt. Der Aufbau dieser Datenstruktur für das 3D-Objektmodell ObjectModel3DTo kann auch durch den Operator prepare_object_model_3d durchgeführt werden. Dies hat den Vorteil, dass der Aufbau auch bei mehreren Aufrufen von distance_object_model_3d mit demselben Ziel-Objektmodell nur einmal durchgeführt werden muss, beispielsweise bei einem Suchmodell. Objekte mit allgemeinen Polygonen können zusätzlich mit dem Operator prepare_object_model_3d trianguliert werden und die Dreiecke werden im 3D-Objekt gespeichert, um die Abstandsberechnung weiter zu beschleunigen. Die Triangulierung wird nur ausgeführt, wenn der generische Parameter *'distance_to'* auf *'triangles'* gesetzt ist. Entgegen der Triangulierung im Operator triangulate_object_model_3d wird hier das Attribut mit den Polygonen nicht gelöscht.

Wenn der Abstand zu Punkten oder zu Dreiecken berechnet wird kann der Operator optional den Index des jeweils nächsten Punktes oder Dreiecks für jeden Punkt aus ObjectModel3DFrom zurückgeben. Dazu muss der generische Parameter *'store_closest_index'* auf *'true'* gesetzt werden (siehe unten). Der Index wird als erweitertes Attribut mit dem Namen *'&closest_index'* im 3D-Objektmodell ObjectModel3DFrom gespeichert. Der Index kann nicht berechnet werden, wenn die *'voxel'*-Methode verwendet wird. Wenn der Abstand eines Punktes zu seinem nächsten Punkt oder Dreieck die in MaxDistance definierte maximale Distanz überschreitet, so wird der Index seines nächsten Elements auf -1 gesetzt.

Optional können auch vorzeichenbehaftete Abstände zu Punkten, Dreiecken oder einem Primitiv berechnet werden. Dazu muss der generische Parameter *'signed_distances'* auf *'true'* gesetzt werden. Vorzeichenbehaftete Abstände können nicht berechnet werden, wenn bei Abständen von Punkten zu Punkten die *'voxel'*-Methode verwendet wird.

Im Folgenden werden die verschiedenen Zieldatentypen und Methoden erklärt, sowie ihre Vor- und Nachteile beschrieben. Der Operator distance_object_model_3d bestimmt, abhängig von den Zieldaten, automatisch eine Methode. Alternativ kann die Methode manuell über den generischen Parameter *'mode'* gesetzt werden.

Abstand zu Punkten: Für die Bestimmung der Punkt-zu-Punkt-Abstände werden die folgenden Methoden unterstützt:

- **Lineare Suche:** Für jeden Punkt aus ObjectModel3DFrom werden die Abstände zu allen Punkten in ObjectModel3DTo berechnet, und der kleinste Abstand verwendet. Diese Methode benötigt keine vorberechnete Datenstruktur, und ist für wenige Punkte in ObjectModel3DTo die schnellste.
- **KD-Baum:** Die Punkte in ObjectModel3DTo werden in einem KD-Baum organisiert, welcher die Suche nach dem nächsten Punkt beschleunigt. Der Aufbau des Baums ist sehr effizient. Die Suchzeit ist dabei in etwa logarithmisch zu der Anzahl der Punkten in ObjectModel3DTo. Die Suchzeit ist allerdings nicht konstant und kann stark von der Lage der Suchpunkte in ObjectModel3DFrom abhängen.
- **Voxel:** Die Punkte in ObjectModel3DTo werden in einer Voxelstruktur gespeichert. Diese Voxelstruktur erlaubt eine Suche in annähernd konstanter Zeit, d.h. unabhängig von der Lage des Suchpunktes und der Größe der Zielpunktwolke.
  Der Aufbau der Voxelstruktur dauert allerdings deutlich länger als der Aufbau eines KD-Baums. Es ist möglich, diese Datenstruktur vorzuberechnen, indem man prepare_object_model_3d mit Purpose *'distance_computation'* auf ObjectModel3DTo anwendet.

Abstand zu Dreiecken: Für die Bestimmung des kleinsten Punkt-zu-Dreieck-Abstands werden die folgenden Methoden unterstützt:

- **Lineare Suche:** Für jeden Punkt aus ObjectModel3DFrom werden die Abstände zu allen Dreiecken in ObjectModel3DTo berechnet und der kleinste Abstand verwendet. Diese Methode benötigt keine vorberechnete Datenstruktur, und ist für wenige Dreiecke in ObjectModel3DTo die schnellste.
- **KD-Baum:** Die in ObjectModel3DTo gespeicherten Dreiecke werden in einem KD-Baum gespeichert, wodurch die Suche nach dem nächsten Dreieck beschleunigt wird. Der Aufbau des Baums ist vergleichsweise effizient. Die Suchzeit ist in etwa logarithmisch zu der Anzahl der Dreiecke in ObjectModel3DTo. Die Suchzeit ist allerdings nicht konstant und kann von der Lage der Suchpunkte in ObjectModel3DFrom abhängen.

Voxel: Die Dreiecke in `ObjectModel3DTo` werden in einer Voxelstruktur gespeichert. Diese Voxelstruktur erlaubt eine Suche in annähernd konstanter Zeit, d.h. unabhängig von der Lage des Suchpunktes und der Anzahl der Dreiecke in der Zielstruktur.
Der Aufbau der Voxelstruktur dauert allerdings deutlich länger als der Aufbau eines KD-Baums. Es ist möglich, diese Datenstruktur vorzuberechnen, indem man `prepare_object_model_3d` mit `Purpose` *'distance_computation'* auf `ObjectModel3DTo` anwendet. Für den Aufbau der Voxelstruktur werden die Dreiecke abgetastet. Die entsprechende Abtastdistanz kann mit den generischen Parameter *'sampling_dist_rel'* und *'sampling_dist_abs'* gesetzt werden.
Standardmäßig wird *'sampling_dist_rel'* auf 0.03 gesetzt. Eine genaue Beschreibung der beiden Parameter findet sich unten. Bei der Verwendung der Voxelstruktur für die Bestimmung der Abstände zu Dreiecken können die berechneten Distanzen in einigen Fällen um ca. 10% der Abtastdistanz von den tatsächlichen Abständen nach oben abweichen.

Abstand zu Primitiven: Da `ObjectModel3DTo` nur ein Primitiv enthalten kann, werden die Punkt-zu-Primitiv-Abstände immer linear durchgeführt. Der Aufbau oder die Verwendung einer Datenstruktur ist weder nötig noch möglich.

Das Berechnen von Abständen zu primitiven Ebenen, die mittels `segment_object_model_3d` oder `fit_primitives_object_model_3d` erzeugt wurden, kann vergleichsweise langsam sein, da diese Ebenen eine komplexe Ausdehnung beinhalten. Wenn nur der Abstand zu der Ebene relevant ist und die Ausdehnung ignoriert werden soll, wird empfohlen, die Pose der Ebene mittels `get_object_model_3d_params` mit dem Parameter *'primitive_parameter_pose'* abzurufen und mittels `gen_plane_object_model_3d` eine neue Ebene ohne Ausdehnung zu erzeugen.

Die folgende Tabelle gibt eine Übersicht über die verschiedenen Zieldaten, Methoden, sowie deren Vor- und Nachteile.

Zieldaten	Methode	Aufbau-Zeit	Ungefähre Suchzeit	Eigenschaften
Punkte	linear	0	$O(N)$	· Keine Vorberechnung · Schnellste Methode für kleine N · Standardmethode für $N < 100$
Punkte	kd-tree	$O(N \log(N))$	$O(\log(N))$	· Schnelle Vorberechnung · Suchzeit nicht konstant · Standardmethode für $N \geq 100$
Punkte	voxel	$O(N \log(N))$	$O(\log(\log(N)))$	· Langsame Vorberechnung · Sehr schnelle Suche · Standardmethode für Vorberechnung mit `prepare_object_model_3d`
Dreiecke	linear	0	$O(N)$	· Keine Vorberechnung · Schnellste Methode für kleine N · Standardmethode
Dreiecke	kd-tree	$O(N \log(N))$	$O(\log(N))$	· Schnelle Vorberechnung · Suchzeit nicht konstant
Dreiecke	voxel	$O(N \log(N))$	$O(\log(\log(N)))$	· Langsame Vorberechnung · Benötigt Abtastdistanz · Sehr schnelle Suche · Kleine Fehler möglich · Standardmethode für Vorberechnung mit `prepare_object_model_3d`
Primitiv	linear	0	$O(1)$	

Zusätzlich zu den oben beschriebenen Parametern können die folgenden Parameter verwendet werden, um den Operator zu beeinflussen. Diese Parameter und ihre Werte können in den Parametern `GenParamName` und `GenParamValue` gesetzt werden. Alle der folgenden Parameter sind optional.

'distance_to' Mit diesem Parameter können die Zieldaten explizit ausgewählt werden.

- *'auto'* (Standardwert) Die Zieldaten werden automatisch ausgewählt. Dazu wird überprüft, welches der folgenden Attribute in `ObjectModel3DTo` vorhanden ist, und das erste vorhandene verwendet: Primitiv, Dreiecke, Punkte.
- *'primitive'* Es werden die Abstände zu dem in `ObjectModel3DTo` vorhandenen Primitiv berechnet.
- *'triangles'* Es werden die Abstände zu den in `ObjectModel3DTo` vorhandenen Dreiecken berechnet.
- *'points'* Es werden die Abstände zu den in `ObjectModel3DTo` vorhandenen Punkten berechnet.

'method' Mit diesem Parameter kann die zu verwendende Methode explizit gesetzt werden. Nicht alle Methoden sind für alle Zieldaten möglich. Die möglichen Kombinationen sind in obiger Tabelle aufgelistet.

- *'auto'* (Standardwert) Es wird die Standardmethode für die ausgewählten Zieldaten verwendet.
- *'linear'* Die Abstände werden über eine lineare Suche berechnet.

'kd-tree' Die Abstände werden über einen KD-Baum berechnet.

'voxel' Die Abstände werden über eine Voxelstruktur berechnet.

'invert_pose' Mit diesem Parameter kann die in `Pose` übergebene Pose invertiert werden.

'false' (Standardwert) Die Pose wird nicht invertiert.

'true' Die Pose wird invertiert.

'output_attribute' Mit diesem Parameter kann der Name des erweiterten Attributs gesetzt werden, in welchem die berechneten Abstände gespeichert werden. Standardmäßig werden die Abstände in dem erweiterten Attribut mit dem Namen *'&distance'* im Objektmodell `ObjectModel3DFrom` gespeichert. Wird das gleiche Objektmodell für mehrere Aufrufe von `distance_object_model_3d` verwendet, so wird das Attribut überschrieben. Um dies zu vermeiden, kann der Name des Attributs verändert werden. Gültige Namen von erweiterten Attribute fangen mit einem *'&'* an.

'sampling_dist_rel', *'sampling_dist_abs'* Diese Parameter werden verwendet, wenn die Abstände zu Dreiecken mit der Voxel-basierten Methode berechnet werden. Für den Aufbau der Voxelstruktur werden die Dreiecke regelmäßig abgetastet. Die Abtastdistanz kann entweder absolut mit dem Parameter *'sampling_dist_abs'* gesetzt werden, oder relativ zum Durchmesser des achsenparallelen umschließenden Quaders mit *'sampling_dist_rel'*. Nur einer der beiden Parameter kann gesetzt werden. Der Durchmesser des achsenparallelen umschließenden Quaders kann mit `get_object_model_3d_params` abgerufen werden. Die Erstellung der Voxelstruktur für Dreiecke kann sehr lange dauern und wird i.d.R. mittels `prepare_object_model_3d` vorberechnet (siehe oben).

'store_closest_index' Dieser Parameter aktiviert die Berechnung des Index des nächsten Punktes oder Dreiecks für jeden Punkt in `ObjectModel3DFrom` (siehe oben). Die Indizes werden im erweiterten Attribut '&closest_index' zurückgegeben.

'false' (Standardwert) Der Index wird nicht zurückgegeben.

'true' Der Index wird zurückgegeben.

'signed_distances' Dieser Parameter wird verwendet, um vorzeichenbehaftete Abstände zwischen den Punkten im 3D-Objektmodell `ObjectModel3DFrom` zu den Punkten, Dreiecken oder dem Primitiv im 3D-Objektmodell `ObjectModel3DTo` zu berechnen.

'false' (Standardwert) Es werden Abstände ohne Vorzeichen zurückgegeben.

'true' Es werden vorzeichenbehaftete Abstände zurückgegeben.

Abhängig von den vorhandenen Zieldaten (Punkte, Dreiecke oder Primitive) sind die folgenden Besonderheiten zu beachten:

Abstand zu Punkten: Eine Berechnung von vorzeichenbehafteten Abständen zu Punkten wird nur für die Methoden *'kd-tree''* und *'linear'* unterstützt. Unabhängig davon welche der beiden Methoden für die Bestimmung des kleinsten Punkt-zu-Punkt-Abstands verwendet wird, ist eine Berechnung der vorzeichenbehafteten Distanz nur möglich, wenn Punktnormalen für die Punkte im 3D-Objektmodell `ObjectModel3DTo` vorhanden sind oder mit dem Operator `set_object_model_3d_attrib_mod` gesetzt wurden.

Abstand zu Dreiecken: Vorzeichenbehaftete Distanzen können für alle oben aufgeführten Methoden berechnet werden. Bei vorzeichenbehafteten Abständen von Punkten zu Dreiecken wird die Distanz negativ gesetzt, wenn das Skalarprodukt mit dem Normalenvektor des Dreiecks kleiner Null ist.

Abstand zu Primitiven: Berechnet man den Abstand zu zylinderförmigen, kugelförmigen oder quaderförmigen Primitiven, so erhalten Punkte des 3D-Objektmodells `ObjectModel3DFrom` einen negativen Abstand, wenn sie sich innerhalb des Primitivs befinden und einen positiven, wenn sie außerhalb davon liegen. Werden Abstände von Punkten zu Ebenen berechnet, so erhalten Punkte einen negativen Abstand, wenn sie unterhalb der Ebene liegen und einen positiven wenn sie oberhalb davon liegen.

Parameter

▷ **`ObjectModel3DFrom`** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells mit den Quelldaten.

▷ **`ObjectModel3DTo`** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des 3D-Objektmodells mit den Zieldaten.

▷ **`Pose`** (input_control) . . . . . pose ⇝ *real* / integer
Pose der Quelldaten in den Zieldaten.
Defaultwert: []

▷ **MaxDistance** (input_control) . . . . . number ⇝ *real* / integer
Maximal interessanter Abstand.
Defaultwert: 0

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'distance_to', 'method', 'invert_pose', 'output_attribute', 'sampling_dist_rel', 'sampling_dist_abs', 'signed_distances', 'store_closest_index'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Werte der generischen Parameter.
Defaultwert: []
Werteliste: GenParamValue ∈ {'auto', 'triangles', 'points', 'polygons', 'primitive', 'kd-tree', 'voxel', 'linear', 'true', 'false'}

Ergebnis

distance_object_model_3d liefert den Wert TRUE wenn alle Parameter korrekt sind. Wenn notwendig wird eine Ausnahmebehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

prepare_object_model_3d, read_object_model_3d, find_surface_model, xyz_to_object_model_3d

Nachfolger

get_object_model_3d_params, render_object_model_3d, disp_object_model_3d, clear_object_model_3d

Siehe auch

prepare_object_model_3d

Modul

3D Metrology

get_object_model_3d_params (: : ObjectModel3D, GenParamName : GenParamValue)

Auslesen der Attribute von 3D-Objektmodellen.

Ein 3D-Objektmodell besteht aus einer Menge von Attributen und ihren Metadaten. Mit Hilfe des Operators get_object_model_3d_params lassen sich Attribute und die Metadaten von 3D-Objektmodellen abfragen. Der Name des Attributes oder der Metadaten, welche abgefragt werden sollen, wird in GenParamName übergeben, der korrespondierende Wert wird in GenParamValue zurückgeliefert. Wenn das abzufragende Attribut oder die Metadaten nicht vorhanden sind, wird eine Fehlerbehandlung durchgeführt. get_object_model_3d_params unterstützt den Zugriff auf mehrere 3D-Objektmodelle und mehrere Attribute auf einmal. Zu beachten ist, dass Attribute oder Metadaten verschiedene Längen haben können. Einige der Standardattribute haben eine definierte Länge, die in der Attributbeschreibung weiter unten jeweils angegeben ist. Die Länge anderer Attribute hängt vom 3D-Objektmodell ab und kann abgefragt werden, indem der Parameter GenParamName z.B. auf *'num_points'*, *'num_triangles'*, *'num_polygons'* oder *'num_lines'* gesetzt wird. So lässt sich beispielsweise die Länge des Standardattributs *'point_coord_x'* abfragen, indem GenParamName auf *'num_points'* gesetzt wird.

Standardattribute

Im Einzelnen kann auf die folgenden Standardattribute und deren Metadaten zugegriffen werden:

'point_coord_x': Die x-Koordinaten der 3D Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator xyz_to_object_model_3d oder read_object_model_3d erstellt.

'point_coord_y': Die y-Koordinaten der 3D Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `xyz_to_object_model_3d` oder `read_object_model_3d` erstellt.

'point_coord_z': Die z-Koordinaten der 3D Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `xyz_to_object_model_3d` oder `read_object_model_3d` erstellt.

'point_normal_x': Die x-Komponenten der 3D Punktnormalen der 3D Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `smooth_object_model_3d` erstellt.

'point_normal_y': Die y-Komponenten der 3D Punktnormalen der 3D Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `smooth_object_model_3d` erstellt.

'point_normal_z': Die z-Komponenten der 3D Punktnormalen der 3D Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `smooth_object_model_3d` erstellt.

'score': Das Qualitätsmaß der Rekonstruktion einer 3D-Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `reconstruct_surface_stereo` erstellt.

'red': Der Rotkanal einer 3D-Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `reconstruct_surface_stereo` erstellt.

'green': Der Grünkanal einer 3D-Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `reconstruct_surface_stereo` erstellt.

'blue': Der Blaukanal einer 3D-Punktmenge. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `reconstruct_surface_stereo` erstellt.

'mapping_row': Die Zeilenkomponenten der Zuordnung der 3D Punkte zu Bildkoordinaten. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden, die Höhe des Originalbildes mit *'mapping_size'*. Dieses Attribut wird typischerweise mit dem Operator `xyz_to_object_model_3d` erstellt.

'mapping_col': Die Spaltenkomponenten der Zuordnung der 3D Punkte zu Bildkoordinaten. Die Anzahl der hier zurückgegebenen Punkte kann mit *'num_points'* ermittelt werden, die Breite des Originalbildes mit *'mapping_size'*. Dieses Attribut wird typischerweise mit dem Operator `xyz_to_object_model_3d` erstellt.

'mapping_size': Die Größe des Originalbildes. Zurückgegeben wird ein Tupel mit den beiden Einträgen Breite und Höhe.

'triangles': Die Indizes der 3D Punkte, welche Dreiecke repräsentieren, in der folgenden Reihenfolge: Die ersten drei Rückgabewerte (Rückgabewert 0,1,2) stellen das erste Dreieck dar, die nächsten drei Rückgabewerte (Rückgabewerte 3,4,5) stellen das zweite Dreieck dar, usw. Alle Indizes entsprechen den Indizes der Koordinaten der 3D Punkte. Zugriff auf die Koordinaten der 3D Punkte ist mit dem generischen Parameter `GenParamName` mit den Werten *'point_coord_x'*, *'point_coord_y'* und *'point_coord_z'* möglich. Die Anzahl der hier zurückgegebenen Dreiecke kann mit *'num_triangles'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `triangulate_object_model_3d` oder `read_object_model_3d` erstellt.

'polygons': Die Indizes der 3D Punkte, welche Polygone repräsentieren, in der folgenden Reihenfolge: Der erste Rückgabewert enthält die Anzahl n der 3D Punkte des ersten Polygons. Die folgenden Rückgabewerte (Rückgabewerte 1,2,..,n) repräsentieren das erste Polygon. Der nächste Rückgabewert (Rückgabewert n+1) enthält die Anzahl m der 3D Punkte des zweiten Polygons. Die folgenden Rückgabewerte (Rückgabewerte n+2,n+3,..,n+1+m) repräsentieren das zweite Polygon, usw. Alle Indizes entsprechen den Indizes der Koordinaten der 3D Punkte. Zugriff auf die Koordinaten der 3D Punkte ist mit dem generischen Parameter `GenParamName` mit den Werten *'point_coord_x'*, *'point_coord_y'* und *'point_coord_z'* möglich. Die Anzahl der hier zurückgegebenen Polygone kann mit *'num_polygons'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `read_object_model_3d` erstellt.

'lines': Die Indizes der 3D Punkte, welche Linienzüge repräsentieren, in der folgenden Reihenfolge: Der erste Rückgabewert enthält die Anzahl n0 der 3D Punkte des ersten Linienzuges. Die folgenden Rückgabewerte (Rückgabewerte 1,2,..,n) repräsentieren den ersten Linienzug. Der nächste Rückgabewert (Rückgabewert

n+1) enthält die Anzahl m der 3D Punkte des zweiten Linienzuges. Die folgenden Rückgabewerte (Rückgabewerte n+2,n+3,..,n+1+m) repräsentieren den zweiten Linienzug usw. Alle Indizes entsprechen den Indizes der Koordinaten der 3D Punkte. Zugriff auf die Koordinaten der 3D Punkte ist mit dem generischen Parameter `GenParamName` mit den Werten *'point_coord_x'*, *'point_coord_y'* und *'point_coord_z'* möglich. Die Anzahl der hier zurückgegebenen Linien kann mit *'num_lines'* ermittelt werden. Dieses Attribut wird typischerweise mit dem Operator `intersect_plane_object_model_3d` erstellt.

'diameter_axis_aligned_bounding_box': Der Durchmesser der 3D Punktmenge, der durch die Länge der Diagonale des kleinsten umschließenden achsenparallelen Quaders (siehe Parameter *'bounding_box1'*) definiert wird. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'center': 3D-Koordinaten des Mittelpunkts des kleinsten umschließenden achsenparallelen Quaders der 3D Punkte (siehe Parameter *'bounding_box1'*). Es werden hier drei Werte pro 3D-Objektmodell zurückgegeben. Handelt es sich um ein 3D-Primitiv ohne 3D-Koordinaten gilt folgendes:

Wenn der Typ des 3D-Primitivs ein Zylinder ist (siehe `gen_cylinder_object_model_3d`) und Begrenzungen vorhanden sind, wird der Mittelpunkt zwischen den Begrenzungen zurückgegeben. Sind keine Begrenzungen vorhanden werden die Translationsparameter der Pose zurückgegeben.

Wenn der Typ des 3D-Primitivs eine Ebene ist (siehe `gen_plane_object_model_3d`) und Begrenzungen vorhanden sind, wird der Schwerpunkt der Ebene aus den Begrenzungen berechnet. Sind keine Begrenzungen der Ebene vorhanden werden die Translationsparameter der Pose zurückgegeben.

Wenn der Typ des 3D-Primitivs eine Kugel oder ein Quader ist (siehe `gen_sphere_object_model_3d` oder `gen_box_object_model_3d`) wird der Mittelpunkt des Objektmodells zurückgegeben.

'original_point_indices': Indizes der 3D-Punkte in einem anderen 3D-Objektmodell (Länge kann mit *'num_points'* abgefragt werden). Typischerweise wird das Attribut von `triangulate_object_model_3d` generiert.

'primitive_type': Der Type des 3D-Primitivs (typischerweise generiert vom Operator `fit_primitives_object_model_3d`). Der Rückgabewert einer Kugel ist *'sphere'*. Der Rückgabewert eines Zylinders ist 'cylinder'. Der Rückgabewert einer Ebene ist *'plane'*. Der Rückgabewert eines Quaders ist *'box'*. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'primitive_parameter': Die Parameter des 3D-Primitivs (z.B. generiert vom Operator `fit_primitives_object_model_3d`). Je nach Typ werden vier bis zehn Werte pro 3D-Objektmodell zurückgegeben.

Wenn der Typ des 3D-Primitivs ein Zylinder ist (siehe `gen_cylinder_object_model_3d`), sind die Rückgabewerte die (x-, y-, z-) Koordinaten des Mittelpunktes [`x_center`, `y_center`, `z_center`], die normierten (x-, y-, z-) Richtungen der Hauptachse des Zylinders [`x_axis`, `y_axis`, `z_axis`] und der Zylinderradius [`radius`]. Die Reihenfolge ist [`x_center`, `y_center`, `z_center`, `x_axis`, `y_axis`, `z_axis`, `radius`].

Wenn der Typ des 3D-Primitivs eine Kugel ist (siehe `gen_sphere_object_model_3d`), sind die Rückgabewerte die (x-, y-, z-) Koordinaten des Mittelpunktes [`x_center`, `y_center`, `z_center`] und der Kugelradius [`radius`]. Die Reihenfolge ist [`x_center`, `y_center`, `z_center`, `radius`].

Wenn der Typ des 3D-Primitivs eine Ebene ist (siehe `gen_plane_object_model_3d`), sind die Rückgabewerte die 4 Parameter der Hesseschen Normalform, d.h. der (x-, y-, z-) Einheitsnormalenvektor [`x`, `y`, `z`] und der orthogonale Abstand (d) der Ebene vom Ursprung des Koordinatensystems. Die Reihenfolge ist [`x`, `y`, `z`, `d`]. Das Vorzeichen des Abstands (d) legt die Seite der Ebene fest, auf welcher sich der Ursprung befindet.

Wenn der Typ des 3D-Primitivs ein Quader ist (siehe `gen_box_object_model_3d`), sind die Rückgabewerte die 3D Pose (Verschiebung, Rotation, Rotationstyp) und die halben Kantenlängen (length1, length2, length3) des Quaders. length1 ist die Länge des Quaders entlang der x-Achse der angegebenen Pose. length2 ist die Länge des Quaders entlang der y-Achse der angegebenen Pose. length3 ist die Länge des Quaders entlang der z-Achse der angegebenen Pose. Die Reihenfolge ist [`trans_x`, `trans_y`, `trans_z`, `rot_x`, `rot_y`, `rot_z`, `rot_type`, `length1`, `length2`, `length3`]. Mehr Details zu 3D Posen und die entsprechenden Transformationsmatrizen sind im Operator `create_pose` zu finden.

'primitive_parameter_pose': Die Parameter des 3D-Primitivs in Form einer 3D Pose (typischerweise generiert vom Operator `fit_primitives_object_model_3d`). Für alle Typen von 3D-Primitiven sind die Rückgabewerte die 3D Pose (Verschiebung, Rotation, Rotationstyp). Mehr Details zu 3D Posen und die entsprechenden Transformationsmatrizen sind im Operator `create_pose` zu finden. Je nach Typ werden sieben bis zehn Werte pro 3D-Objektmodell zurückgegeben.

Wenn der Typ des 3D-Primitivs ein Zylinder ist (siehe gen_cylinder_object_model_3d), wird zusätzlich der Zylinderradius [radius] zurückgegeben. Die Reihenfolge ist [trans_x, trans_y, trans_z, rot_x, rot_y, rot_z, rot_type, radius].

Wenn der Typ des 3D-Primitivs eine Kugel ist (siehe gen_sphere_object_model_3d), wird zusätzlich der Kugelradius [radius] zurückgegeben. Die Reihenfolge ist [trans_x, trans_y, trans_z, rot_x, rot_y, rot_z, rot_type, radius].

Wenn der Typ des 3D-Primitivs eine Ebene ist (siehe gen_plane_object_model_3d), ist die Reihenfolge [trans_x, trans_y, trans_z, rot_x, rot_y, rot_z, rot_type].

Wenn der Typ des 3D-Primitivs ein Quader ist (siehe gen_box_object_model_3d), werden zusätzlich die halben Kantenlängen (length1, length2, length3) des Quaders zurückgegeben. length1 ist die Länge des Quaders entlang der x-Achse der angegebenen Pose. length2 ist die Länge des Quaders entlang der y-Achse der angegebenen Pose. length3 ist die Länge des Quaders entlang der z-Achse der angegebenen Pose. Die Reihenfolge ist [trans_x, trans_y, trans_z, rot_x, rot_y, rot_z, rot_type, length1, length2, length3].

'primitive_pose': Die Parameter des 3D-Primitivs in Form einer 3D Pose (typischerweise generiert vom Operator fit_primitives_object_model_3d). Für alle Typen von 3D-Primitiven sind die Rückgabewerte die 3D Pose (Verschiebung, Rotation, Rotationstyp). Mehr Details zu 3D Posen und die entsprechenden Transformationsmatrizen sind im Operator create_pose zu finden. Es werden sieben Werte pro 3D-Objektmodell zurückgegeben. Die Reihenfolge ist [trans_x, trans_y, trans_z, rot_x, rot_y, rot_z, rot_type].

'primitive_parameter_extension': Die Ausdehnung der 3D-Primitive vom Typ Zylinder und Ebene (typischerweise generiert vom Operator fit_primitives_object_model_3d). Die Anzahl der Attribute hängt vom Typ ab und kann mit *'num_primitive_parameter_extension'* abgefragt werden.

Wenn der Typ des 3D-Primitivs ein Zylinder ist (siehe gen_cylinder_object_model_3d), sind die Rückgabewerte (MinExtent, MaxExtent) die Ausdehnung des Zylinders in Richtung der Rotationsachse. Diese Werte werden in der Reihenfolge [MinExtent, MaxExtent] zurückgegeben. MinExtent gibt die Länge des Zylinders entgegen der Richtung der Rotationsachse an. MaxExtent gibt die Länge des Zylinders entlang der Richtung der Rotationsachse an.

Wenn der Typ des 3D-Primitivs eine Ebene ist (generiert mit fit_primitives_object_model_3d), dann ist der Rückgabewert ein Tupel von koplanaren Punkten aus der gefitteten Ebene. Die Reihenfolge ist [x-Koordinate von Punkt 1, x-Koordinate von Punkt 2, x-Koordinate von Punkt 3, ..., y-Koordinate von Punkt 1, y-Koordinate von Punkt 2, x-Koordinate von Punkt 3, ...]. Die Koordinaten beschreiben die Stützstellen einer konvexen Hülle. Diese wird aus den Projektionen derjenigen Punkte auf die gefittete Ebene gebildet, die dem Ebenenfit Rechnung tragen. Wurde die Ebene mit gen_plane_object_model_3d generiert, werden alle Punkte zurückgeliefert, die zum Generieren der Ebene benutzt wurden (XExtent, YExtent).

'primitive_rms': Der quadratische Restfehler des 3D-Primitivs (typischerweise generiert vom Operator fit_primitives_object_model_3d). Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'reference_point': 3D-Koordinaten des Referenzpunktes des vorbereiteten 3D-Formmodells für das Matching. Der Referenzpunkt ist der Mittelpunkt des kleinsten umschließenden achsenparallelen Quaders (siehe Parameter *'bounding_box1'*). Es werden hier drei Werte pro 3D-Objektmodell zurückgegeben.

'bounding_box1': Kleinster umschließender achsenparalleler Quader in der Form (min_x, min_y, min_z, max_x, max_y, max_z). Es werden hier sechs Werte pro 3D-Objektmodell zurückgegeben.

'num_points': Die Anzahl der Punkte. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'num_triangles': Die Anzahl der Oberflächendreiecke. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'num_polygons': Die Anzahl der Oberflächen. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'num_lines': Die Anzahl der Linienzüge. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'num_primitive_parameter_extension': Die Anzahl der erweiterten Daten von 3D-Primitiven. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_points': Das Vorhandensein von 3D Punkten. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_point_normals': Das Vorhandensein von 3D Punktnormalen. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_triangles': Das Vorhandensein von Oberflächendreiecken. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_polygons': Das Vorhandensein von Oberflächen oder Polygonen. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_lines': Das Vorhandensein von 3D Linienzügen. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_xyz_mapping': Das Vorhandensein der Zuordnung der 3D Punkte zu Bildkoordinaten. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_shape_based_matching_3d_data': Das Vorhandensein eines Formmodells für das formbasierte 3D-Matching. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_distance_computation_data': Das Vorhandensein einer vorberechneten Datenstruktur für die 3D Abstandsbestimmung. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben. Die Datenstruktur kann mit dem Operator `prepare_object_model_3d` erstellt werden. Sie wird von dem Operator `distance_object_model_3d` verwendet.

'has_surface_based_matching_data': Das Vorhandensein von Daten für das oberflächenbasierte 3D-Matching. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_segmentation_data': Das Vorhandensein von Daten für die 3D Segmentierung. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_primitive_data': Das Vorhandensein eines 3D-Primitivs. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'has_primitive_rms': Das Vorhandensein eines quadratischen Restfehlers eines 3D-Primitivs. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'neighbor_distance':

'neighbor_distance N': Für jeden Punkt der Abstand des N-ten Nachbarpunktes. N muss ein positiver Integer-Wert sein und ist standardmäßig 25. Für jeden Punkt werden die übrigen Punkte anhand ihres Abstands sortiert, und der Abstand des N-ten Punktes zurückgeliefert.

'num_neighbors X': Für jeden Punkt die Anzahl an Nachbarn mit einem Abstand von höchstens X.

'num_neighbors_fast X': Für jeden Punkt die ungefähre Anzahl an Nachbarn mit einem Abstand von höchstens X. Die Abstände werden mittels Voxeln approximiert, was üblicherweise schneller ist als die exakte Berechnung für *'num_neighbors'*.

Erweiterte Attribute

Erweiterte Attribute sind Attribute mit benutzerdefinierten Namen und Daten. Sie werden mit dem Operator `set_object_model_3d_attrib` erstellt. Zugegriffen werden kann auf die folgenden erweiterten Attribute und deren Metadaten:

'&attribute_name': Der benutzerdefinierte Name des erweiterten Attributs. Zu beachten dass der Name mit *'&'*, z.B., *'&my_attrib'* beginnen muss. Die Daten der abgefragten erweiterten Attribute werden in `GenParamValue` zurückgegeben. Die Reihenfolge ist die gleiche wie die Reihenfolge der Attributnamen, die in `GenParamName` übergeben wird.

'extended_attribute_names': Die Namen aller erweiterten Attribute. Es wird hier pro erweitertem Attribut ein Wert zurückgegeben.

'extended_attribute_types': Die Typen aller erweiterten Attribute. Es wird hier pro erweitertem Attribut ein Wert zurückgegeben. Dabei sind die Ausgabewerte wie die ausgegebenen Namen der erweiterten Attribute geordnet.

'has_extended_attribute': Das Vorhandensein mindestens eines erweiterten Attributs. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

'num_extended_attribute': Die Anzahl der erweiterten Attribute. Es wird hier ein Wert pro 3D-Objektmodell zurückgegeben.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle der 3D-Objektmodelle.

▷ **`GenParamName`** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Attribute, die für die 3D-Objektmodelle abgefragt werden soll.
Defaultwert: 'num_points'
Werteliste: `GenParamName` ∈ {'point_coord_x', 'point_coord_y', 'point_coord_z', 'point_normal_x', 'point_normal_y', 'point_normal_z', 'mapping_row', 'mapping_col', 'mapping_size', 'triangles', 'polygons',

'lines', 'diameter_axis_aligned_bounding_box', 'center', 'primitive_type', 'primitive_rms', 'primitive_parameter', 'primitive_parameter_pose', 'primitive_pose', 'primitive_parameter_extension', 'reference_point', 'bounding_box1', 'num_points', 'num_triangles', 'num_polygons', 'num_lines', 'num_primitive_parameter_extension', 'has_points', 'has_point_normals', 'has_triangles', 'has_polygons', 'has_lines', 'has_xyz_mapping', 'has_shape_based_matching_3d_data', 'has_surface_based_matching_data', 'has_segmentation_data', 'has_primitive_data', 'has_primitive_rms', 'extended_attribute_names', 'extended_attribute_types', 'has_extended_attribute', 'num_extended_attribute', 'has_distance_computation_data', 'red', 'green', 'blue', 'score', 'neighbor_distance', 'num_neighbors', 'num_neighbors_fast', 'original_point_indices'}

▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Werte des generischen Parameters.

Ergebnis

`get_object_model_3d_params` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`, `prepare_object_model_3d`, `sample_object_model_3d`, `triangulate_object_model_3d`, `intersect_plane_object_model_3d`, `set_object_model_3d_attrib`, `fit_primitives_object_model_3d`, `gen_plane_object_model_3d`, `gen_sphere_object_model_3d`, `gen_cylinder_object_model_3d`, `gen_box_object_model_3d`, `gen_sphere_object_model_3d_center`

Nachfolger

`write_object_model_3d`, `clear_object_model_3d`

Modul

3D Metrology

max_diameter_object_model_3d (: : ObjectModel3D : Diameter)

Berechnen des maximalen Durchmessers eines 3D-Objektmodells.

`max_diameter_object_model_3d` berechnet den maximalen Durchmesser eines 3D-Objektmodells und gibt ihn in `Diameter` zurück. Der maximale Durchmesser ist gleich dem maximalen Abstand zweier Punkte auf der konvexen Hülle des 3D-Objektmodells.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells.

▷ **Diameter** (output_control) . . . . . number(-array) ⇝ *real*
Der berechnete Durchmesser.
Parameteranzahl: Diameter == ObjectModel3D

Beispiel

```
gen_object_model_3d_from_points (rand(200), rand(200),\
                                 rand(200), ObjectModel3D)
max_diameter_object_model_3d (ObjectModel3D, Diameter)
```

Ergebnis

`max_diameter_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `connection_object_model_3d`

Nachfolger

`select_object_model_3d`

Siehe auch

`volume_object_model_3d_relative_to_plane`, `area_object_model_3d`, `moments_object_model_3d`

Modul

3D Metrology

moments_object_model_3d (: : ObjectModel3D, MomentsToCalculate : Moments)

Berechnen des Mittelwerts oder des zentralen Moments zweiter Ordnung für ein 3D-Objektmodell.

`moments_object_model_3d` berechnet die Zentralmomente erster und zweiter Ordnung. Im Parameter `MomentsToCalculate` kann *'mean'* ausgewählt werden, um den Mittelwert der Punktkoordinaten zu ermitteln. Es kann *'central_moment_2_points'* angegeben werden, um das zentrale Moment zweiter Ordnung zu berechnen. In diesem Fall enthalten die Rückgabewerte in `Moments` die Varianzen entlang der x-, y-, z-, x-y-, x-z- und y-z-Achsen. Die drei Hauptachsen der zum 3D-Objektmodell gehörenden Punktwolke können mit *'principal_axes'* berechnet werden. Das Ergebnis wird als Pose zurückgegeben, die ihren Ursprung im Mittelwert der Punktkoordinaten hat. Das der Pose entsprechende Koordinatensystem hat die x-Achse entlang der ersten Hauptachse, die y-Achse entlang der zweiten Hauptachse und die z-Achse entlang der dritten Hauptachse ausgerichtet.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells.

▷ **MomentsToCalculate** (input_control) . . . . . number(-array) ⇝ *string*
Das Moment, das berechnet werden soll.
Defaultwert: 'mean_points'
Werteliste: `MomentsToCalculate` ∈ {'mean_points', 'central_moment_2_points', 'principal_axes'}

▷ **Moments** (output_control) . . . . . number(-array) ⇝ *real*
Das berechnete Moment.
Parameteranzahl: Moments == ObjectModel3D

Beispiel

```
gen_object_model_3d_from_points (rand(200), rand(200),\
                                 rand(200), ObjectModel3D)
moments_object_model_3d (ObjectModel3D, ['mean_points',\
                         'central_moment_2_points','principal_axes'], \
                         Moments)
```

Ergebnis

`moments_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `connection_object_model_3d`

Nachfolger

`project_object_model_3d`, `object_model_3d_to_xyz`

Siehe auch

`volume_object_model_3d_relative_to_plane`

Modul

3D Metrology

select_object_model_3d (: : ObjectModel3D, Feature, Operation, MinValue, MaxValue : ObjectModel3DSelected)

Auswählen von 3D-Objektmodellen anhand von globalen Merkmalen.

`select_object_model_3d` wählt anhand von globalen Merkmalen 3D-Objektmodelle aus. Es gibt folgende Merkmale, die in `Feature` angegeben werden können:

'mean_points_x': Der Mittelwert der x-Komponente der Punkte im 3D-Objektmodell.

'mean_points_y': Der Mittelwert der y-Komponente der Punkte im 3D-Objektmodell.

'mean_points_z': Der Mittelwert der z-Komponente der Punkte im 3D-Objektmodell.

'diameter': Der maximale Durchmesser des 3D-Objektmodells.

'volume': Das Volumen des vermaschten 3D-Objektmodells über der x-y Ebene im Koordinatenursprung (entspricht der Standardparametrisierung von `volume_object_model_3d_relative_to_plane` mit der Pose [0,0,0,0,0,0,0]. Die Ebene kann nicht verändert werden.

'volume_axis_aligned_bounding_box': Das Volumen des umschließenden achsenparallelen Quaders des 3D-Objektmodells.

'area': Die Fläche der Vermaschung des 3D-Objektmodells.

'diameter_axis_aligned_bounding_box': Der Durchmesser der 3D Punktmenge, der durch die Länge der Diagonale des kleinsten umschließenden achsenparallelen Quaders bestimmt ist.

'diameter_bounding_box': Der Durchmesser der 3D Punktmenge, der durch die Länge der Diagonale des kleinsten umschließenden Quaders beliebiger Orientierung bestimmt ist. Die Berechnung dieses Merkmals hat eine hohe Berechnungskomplexität.

'diameter_object': Der Durchmesser der 3D Punktmenge, der durch den maximalen Abstand zweier Punkte bestimmt ist.

'central_moment_2_x': Die x-Komponente des Zentralmoments zweiter Ordnung der Punkte im 3D-Objektmodell.

'central_moment_2_y': Die y-Komponente des Zentralmoments zweiter Ordnung der Punkte im 3D-Objektmodell.

'central_moment_2_z': Die z-Komponente des Zentralmoments zweiter Ordnung der Punkte im 3D-Objektmodell.

'central_moment_2_xy': Die xy-Komponente des Zentralmoments zweiter Ordnung der Punkte im 3D-Objektmodell.

'central_moment_2_xz': Die xz-Komponente des Zentralmoments zweiter Ordnung der Punkte im 3D-Objektmodell.

'central_moment_2_yz': Die yz-Komponente des Zentralmoments zweiter Ordnung der Punkte im 3D-Objektmodell.

'num_points': Die Anzahl der Punkte.

'num_triangles': Die Anzahl der Oberflächendreiecke.

'num_faces': Die Anzahl der Oberflächen.

'num_lines': Die Anzahl der Linienzüge.

'has_points': Das Vorhandensein von 3D Punkten.

'has_point_normals': Das Vorhandensein von 3D Punktnormalen.

'has_triangles': Das Vorhandensein von Oberflächendreiecken.

'has_faces': Das Vorhandensein von Oberflächen oder Polygonen.

'has_lines': Das Vorhandensein von 3D Linienzügen.

'has_xyz_mapping': Das Vorhandensein der Zuordnung der 3D Punkte zu Bildkoordinaten.

'has_shape_based_matching_3d_data': Das Vorhandensein eines Formmodells für das formbasierte 3D-Matching.

'has_surface_based_matching_data': Das Vorhandensein von Daten für das oberflächenbasierte 3D-Matching.

'has_segmentation_data': Das Vorhandensein von Daten für die 3D Segmentierung.

'has_primitive_data': Das Vorhandensein eines 3D-Primitivs.

Für alle Merkmale, die in `Feature` angegeben werden, muss ein minimaler und ein maximaler Wert in `MinValue` und `MaxValue` angegeben werden. Mit diesen Werten werden jene 3D-Objektmodelle ausgewählt und kopiert, die alle gegebenen Bedingungen erfüllen. Für `MinValue` und `MaxValue` kann jeweils *'min'* und *'max'* gesetzt werden um die jeweilige Grenze zu ignorieren.

In `Operation` kann eine logische Operation angegeben werden, um Merkmale zu verknüpfen. Die unterstützten Operationen sind *'or'* und *'and'*.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle der 3D-Objektmodelle.

▷ **`Feature`** (input_control) . . . . . string(-array) ⇝ *string*
Liste der Merkmale für die Selektion.
Defaultwert: 'has_triangles'
Werteliste: `Feature` ∈ {'mean_points_x', 'mean_points_y', 'mean_points_z', 'volume', 'volume_axis_aligned_bounding_box', 'central_moment_2_x', 'central_moment_2_y', 'central_moment_2_z', 'central_moment_2_xy', 'central_moment_2_xz', 'central_moment_2_yz', 'diameter_axis_aligned_bounding_box', 'diameter_bounding_box', 'diameter_object', 'area', 'has_point_normals', 'has_triangles', 'has_faces', 'has_lines', 'has_xyz_mapping', 'has_point_normals', 'has_shape_based_matching_3d_data', 'has_surface_based_matching_data', 'has_segmentation_data', 'has_primitive_data', 'num_points', 'num_triangles', 'num_faces', 'num_lines'}

▷ **`Operation`** (input_control) . . . . . string ⇝ *string*
Logischer Operator, um die Merkmale in `Feature` zu kombinieren.
Defaultwert: 'and'
Werteliste: `Operation` ∈ {'and', 'or'}

▷ **`MinValue`** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
Minimaler Wert für das entsprechende Merkmal.
Defaultwert: 1
Wertevorschläge: `MinValue` ∈ {0, 1, 100, 0.1, 'true', 'false', 'min'}

▷ **`MaxValue`** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
Maximaler Wert für das entsprechende Merkmal.
Defaultwert: 1
Wertevorschläge: `MaxValue` ∈ {0, 1, 10, 100, 0.1, 'true', 'false', 'max'}

▷ **`ObjectModel3DSelected`** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle der 3D-Objektmodelle, die alle gegebenen Bedingungen erfüllen.

Beispiel

```
gen_object_model_3d_from_points (rand(20)-1.0, rand(20)-1.0,\
                                 rand(20)-1.0, ObjectModel3D1)
gen_object_model_3d_from_points (rand(20), rand(20),\
                                 rand(20), ObjectModel3D2)
select_object_model_3d ([ObjectModel3D1, ObjectModel3D2],\
                        'mean_points_x', 'and', 0, 1, ObjectModel3DSelected)
```

Ergebnis

`select_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `select_points_object_model_3d`, `connection_object_model_3d`

Nachfolger

`project_object_model_3d`, `object_model_3d_to_xyz`

Siehe auch

`volume_object_model_3d_relative_to_plane`, `area_object_model_3d`, `max_diameter_object_model_3d`, `moments_object_model_3d`

Modul

3D Metrology

smallest_bounding_box_object_model_3d (: : ObjectModel3D, Type : Pose, Length1, Length2, Length3)

Berechnen des kleinsten umschließenden Quaders eines 3D-Objektmodells.

`smallest_bounding_box_object_model_3d` berechnet aus einem 3D-Objektmodell den kleinsten umschließenden Quader. Das Ergebnis wird mit einer Pose und drei Längen beschrieben. Der Quader wird über ein Koordinatensystem im Raum bestimmt (`Pose`), das seine x-Achse entlang der längsten Seite des Quaders hat, die y-Achse entlang der zweitlängsten Seite und die z-Achse entlang der drittlängsten Seite. Die Längen werden in den jeweiligen Parametern `Length1`, `Length2` und `Length3` zurückgegeben. Es gibt zwei verschiedene Berechnungsarten für den kleinsten umschließenden Quader:

'oriented': Sucht nach einer Annäherung des kleinsten orientierten umschließenden Quader. Die Berechnung ist vergleichsweise aufwendig. Das entsprechende Verfahren ist randomisiert und kann daher bei jedem Aufruf eine anderen orientierte umschließenden Quader zurückliefern.

'axis_aligned': Bildet aus den maximalen Ausdehnungen des 3D-Objektmodells entlang der Koordinatenachsen einen umschließenden Quader.

Um nur die Ecken des Quaders im Falle *'axis_aligned'* zu erhalten, kann alternativ der Operator `get_object_model_3d_params` mit dem Parameter *'bounding_box1'* verwendet werden.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle der 3D-Objektmodelle, das 3D-Punkte enthält.

▷ **Type** (input_control) . . . . . string ⇝ *string*
Methode zur Berechnung des Quaders.
Defaultwert: 'oriented'
Werteliste: `Type` ∈ {'oriented', 'axis_aligned'}

▷ **Pose** (output_control) . . . . . pose(-array) ⇝ *real* / integer
Die errechnete Lage des Zentrums des Quaders im Raum.

▷ **Length1** (output_control) . . . . . number(-array) ⇝ *real*
Die Länge der längsten Seite des Quaders entlang der x-Achse der errechneten Pose.
Parameteranzahl: Length1 == ObjectModel3D

▷ **Length2** (output_control) . . . . . number(-array) ⇝ *real*
Die Länge der zweitlängsten Seite des Quaders entlang der y-Achse der errechneten Pose.
Parameteranzahl: Length2 == ObjectModel3D

▷ **Length3** (output_control) . . . . . number(-array) ⇝ *real*
Die Länge der drittlängsten Seite des Quaders entlang der z-Achse der errechneten Pose.
Parameteranzahl: Length3 == ObjectModel3D

Beispiel

```
gen_object_model_3d_from_points (rand(20), rand(20), rand(20), \
                                 ObjectModel3D)
smallest_bounding_box_object_model_3d (ObjectModel3D, 'oriented', \
                                       Pose, Length1, Length2, Length3)
gen_box_object_model_3d (Pose, Length1, Length2, Length3, ObjectModel3D1)
dev_get_window (WindowHandle)
visualize_object_model_3d (WindowHandle, [ObjectModel3D,ObjectModel3D1], \
                           [], [], ['alpha_1'], [0.5], [], [], [], PoseOut)
```

Ergebnis

`smallest_bounding_box_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`connection_object_model_3d`, `simplify_object_model_3d`

Nachfolger

`gen_box_object_model_3d`

Siehe auch

`smallest_sphere_object_model_3d`

Modul

3D Metrology

smallest_sphere_object_model_3d (: : ObjectModel3D : CenterPoint, Radius)

Berechnen der kleinsten umschließenden Kugel eines 3D-Objektmodells.

`smallest_sphere_object_model_3d` berechnet über die 3D-Punkte in einem 3D-Objektmodell die kleinste umschließende Kugel. Die Parameter der Kugel werden in den Parametern `CenterPoint` und `Radius` zurückgegeben.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells, das 3D-Punkte enthält.

▷ **CenterPoint** (output_control) . . . . . number-array ⇝ *real*
Das Zentrum der Kugel.
Parameteranzahl: CenterPoint == 3 * ObjectModel3D

▷ **Radius** (output_control) . . . . . number(-array) ⇝ *real*
Der Radius der Kugel.
Parameteranzahl: Radius == ObjectModel3D

Beispiel

```
gen_object_model_3d_from_points (rand(20), rand(20), rand(20),\
                                 ObjectModel3D)
```

```
smallest_sphere_object_model_3d(ObjectModel3D, CenterPoint, Radius)
gen_sphere_object_model_3d_center (CenterPoint[0], CenterPoint[1], \
                                   CenterPoint[2], Radius, ObjectModel3D1)
dev_get_window (WindowHandle)
visualize_object_model_3d (WindowHandle, [ObjectModel3D,ObjectModel3D1], \
                           [], [], ['alpha_1'], [0.5], [], [], [], PoseOut)
```

Ergebnis

`smallest_sphere_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`connection_object_model_3d`

Nachfolger

`gen_sphere_object_model_3d`

Siehe auch

`smallest_bounding_box_object_model_3d`

Modul

3D Metrology

volume_object_model_3d_relative_to_plane (: : ObjectModel3D, Plane, Mode, UseFaceOrientation : Volume)

Berechnen des Volumens eines 3D-Objektmodells.

`volume_object_model_3d_relative_to_plane` berechnet das Volumen unter den Flächen eines 3D-Objektmodells relativ zu einer Ebene im Raum. Die Ebene wird durch die x-y-Ebene der Pose in `Plane` definiert.

Für das 3D-Objektmodell `ObjectModel3D` muss eine Triangulierung oder eine Polygonliste vorhanden sein. Falls das 3D-Objektmodell eine wasserdichte und geordnete Vermaschung besitzt, berechnet der Operator mit den Standardeinstellungen das tatsächliche Volumen. Um Fälle zu behandeln, wo das Netz nicht wasserdicht oder die Vermaschung nicht konsistent ist, kann die Berechnung des Volumens mit den Parametern `Mode` und `UseFaceOrientation` beeinflusst werden.

Wie das Volumen berechnet wird:

Zuerst berechnet der Operator das Volumen der Prismen, die sich aus der Projektion der einzelnen Flächen auf die Ebene ergeben.

Die einzelnen Volumen der Prismen können positiv oder negativ sein, abhängig von der Orientierung (hin zur oder weg von der Ebene) oder der Lage (oberhalb oder unterhalb) der Fläche. Das kann mit dem Parameter `UseFaceOrientation` eingestellt werden.

Danach werden die Volumen der Prismen abhängig vom Parameter `Mode` aufsummiert.

Das in `Volume` zurückgegebene Volumen ist der Absolutbetrag der berechneten Summe.

Einstellen der Parameter:

`Mode` kann folgende Werte annehmen:

'signed' (default) Die Volumen oberhalb und unterhalb der Ebene werden addiert.

'unsigned' Das Volumen unterhalb der Ebene wird vom Volumen oberhalb der Ebene abgezogen.

'positive' Nur Flächen oberhalb der Ebene werden berücksichtigt.

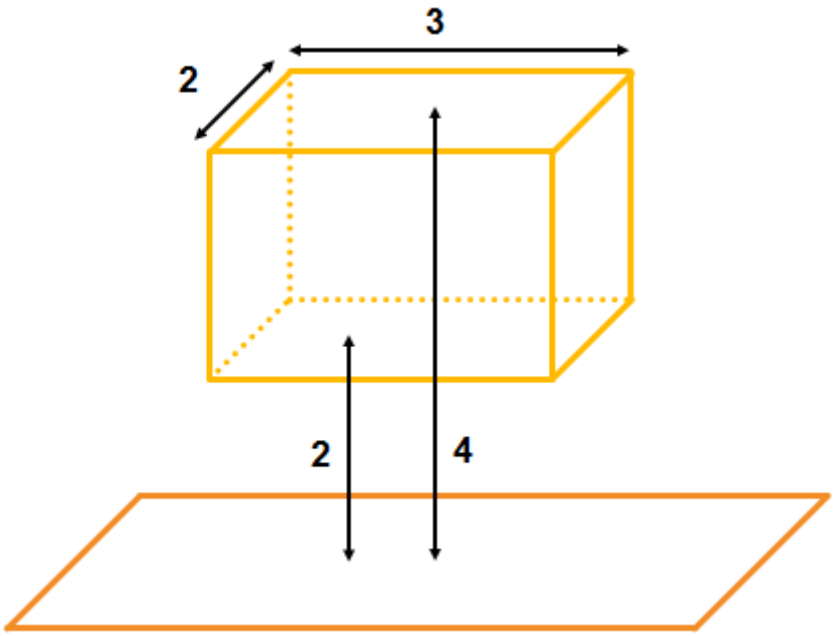
'negative' Nur Flächen unterhalb der Ebene werden berücksichtigt.

`UseFaceOrientation` kann folgende Werte annehmen:

'true' (default) Verwendet die Orientierung der Fläche relativ zur Ebene. Eine Fläche zeigt weg von der Ebene, falls die Eckpunkte von der Ebene aus betrachtet im Uhrzeigersinn angeordnet sind. Das Volumen eines Prismas ist **positiv**, wenn die Orientierung der Fläche **weg von der Ebene** zeigt. Das Volumen eines Prismas ist **negativ**, wenn die Orientierung der Fläche **hin zu der Ebene** zeigt.

'false' Das Volumen eines Prismas is **positiv**, wenn die Fläche **oberhalb** der Ebene liegt. Das Volumen eines Prismas is **negativ**, wenn die Fläche **unterhalb** der Ebene liegt.

Zum Beispiel wird mit der Standardeinstellung (`Mode`: *'signed'*, `UseFaceOrientation`: *'true'*) das Volumen eine geschlossenen Objektes berechnet. In diesem Fall wird die Ebene `Plane` zwar benötigt, hat aber keinen Einfluss auf das berechnete Endergebnis.



Beispiel: (A) `Mode`: *'signed'*, `UseFaceOrientation`: *'true'*: $V = (2 * 3 * 4) + (-2 * 3 * 2) = 24 - 12 = 12$ (B) `Mode`: *'signed'*, `UseFaceOrientation`: *'false'*: $V = (2 * 3 * 4) + (2 * 3 * 2) = 24 + 12 = 36$ (C) `Mode`: *'negative'*: $V = 0$

Achtung

Ist die Entfernung zwischen der Ebene und dem Objekt sehr groß (ca. 10000 mal größer als der Durchmesser), kann die Berechnung des Volumens numerisch instabil werden.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle eines 3D-Objektmodells.

▷ **`Plane`** (input_control) . . . . . pose(-array) ⇝ *real* / integer
Pose der Ebene.
Defaultwert: [0,0,0,0,0,0,0]

▷ **`Mode`** (input_control) . . . . . string(-array) ⇝ *string*
Methode, mit der Volumina ober- und unterhalb der Referenzebene kombiniert werden.
Defaultwert: 'signed'
Werteliste: `Mode` ∈ {'positive', 'negative', 'unsigned', 'signed'}

▷ **`UseFaceOrientation`** (input_control) . . . . . string(-array) ⇝ *string*
Legt fest, ob die Orientierung der Dreiecke Einfluss auf das resultierende Vorzeichen des darunter liegenden Volumens haben soll.
Defaultwert: 'true'
Werteliste: `UseFaceOrientation` ∈ {'true', 'false'}

▷ **Volume** (output_control) ..... number(-array) ⇝ *real*
Absolutbetrag des berechneten Volumens.
Parameteranzahl: Volume == ObjectModel3D

Beispiel

```
gen_box_object_model_3d ([0,0,0,0,0,0,0],3,2,1, ObjectModel3D)
convex_hull_object_model_3d (ObjectModel3D, ObjectModel3DConvexHull)
volume_object_model_3d_relative_to_plane (ObjectModel3DConvexHull,\
                                          [0,0,0,0,0,0,0], 'signed',\
                                          'true', Volume)
```

Ergebnis

`volume_object_model_3d_relative_to_plane` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`, `select_points_object_model_3d`

Nachfolger

`project_object_model_3d`, `object_model_3d_to_xyz`

Siehe auch

`area_object_model_3d`

Modul

3D Metrology

4.3 Segmentation

`fit_primitives_object_model_3d` (: : `ObjectModel3D`, `GenParamName`, `GenParamValue` : `ObjectModel3DOut`)

Anpassen von 3D Primitiven an 3D Punkte.

Der Operator `fit_primitives_object_model_3d` passt ein 3D Primitiv an eine Menge von 3D Punkten an. Die 3D Punkte sind durch ein 3D-Objektmodellhandle `ObjectModel3D` gegeben. Die 3D Formen, die als 3D Primitive zur Verfügung stehen, umfassen eine Ebene, eine Kugel und einen Zylinder. Eine Segmentierung der Eingabepunkte erfolgt nicht, d.h., es muss sichergestellt sein, dass die 3D Punktmenge bereits einem 3D Primitiv entspricht. Eine Segmentierung kann z.B. mit dem Operator `segment_object_model_3d` durchgeführt werden.

Das Ergebnis von `fit_primitives_object_model_3d` ist ein Handle `ObjectModel3DOut` für das 3D-Objektmodell, welches u.a. die Parameter des angepassten Primitivs enthält. Die Information, die im 3D-Objektmodell gespeichert ist, kann mit dem Operator `get_object_model_3d_params` abgerufen werden. Für Primitive vom Typ Ebene und Zylinder wird auch deren Ausdehnung bestimmt. Auch diese Werte können mit `get_object_model_3d_params` ermittelt werden.

Die Parameter des Zylinder sind die (x-, y-, z-) Koordinaten des Mittelpunktes, die normierten (x-, y-, z-) Richtungen der Hauptachse des Zylinders und der Zylinderradius. Der Mittelpunkt liegt nicht notwendigerweise im Schwerpunkt des Zylinders (siehe hierzu auch die Erläuterung der Parameter MinExtent und MaxExtent des Operators `gen_cylinder_object_model_3d`). Das Vorzeichen der Hauptachse ist so gewählt, dass die Hauptachse in den Halbraum zeigt, der den Ursprung beinhaltet. Die Parameter der Kugel sind die (x-, y-, z-) Koordinaten des Mittelpunktes und der Kugelradius. Die Parameter der Ebene sind die 4 Parameter der Hesseschen Normalform, d.h. der (x-, y-, z-) Einheitsnormalenvektor und der orthogonale Abstand der Ebene vom Ursprung

des Koordinatensystems. Das Vorzeichen in der Hesseschen Normalform ist so gewählt, dass der Normalenvektor in den Halbraum zeigt, der den Ursprung beinhaltet, und der Abstand nicht positiv ist.

Kann kein 3D Primitiv an die 3D Punktmenge angepasst werden, enthält das zurückgegebene 3D-Objektmodell kein Primitiv. Abhängig von den gewählten Parameterwerten für *'output_point_coord'* und *'output_xyz_mapping'* (siehe unten) ist das zurückgelieferte Objektmodell entweder leer oder enthält die 3D Punkte oder enthält die 3D Punkte zusammen mit der Zuordnung der 3D Punkte zu Bildkoordinaten des Eingabe-Objektmodells `ObjectModel3D`.

Um das Fitting zu steuern, können in `GenParamName` und `GenParamValue` generische Parameter angepasst werden. In den meisten Fällen reichen die Standardwerte aus, so dass keine Anpassung nötig ist. Die folgenden generischen Parameter und die dazugehörigen Werte sind verfügbar:

'primitive_type': Der Parameter legt fest, welches 3D Primitiv an die 3D Punktmenge angepasst werden soll. Ein bestimmtes 3D Primitiv wird ausgewählt, indem *'primitive_type'* auf *'cylinder'* (Zylinder), *'sphere'* (Kugel) oder *'plane'* (Ebene) gesetzt wird. Soll aus mehreren 3D Primitiven das am besten passende 3D Primitiv, d.h., das 3D Primitiv mit dem kleinsten quadratischen Restfehler, gewählt werden, können die gewählten Primitivtypen gemeinsam in einem Tupel gesetzt werden. Soll aus allen 3D Primitiven das am besten passende 3D Primitiv gewählt werden, kann *'primitive_type'* auf *'all'* gesetzt werden.

Werteliste: 'cylinder', *'sphere'*, *'plane'*, *'all'*

Defaultwert: 'cylinder'

'fitting_algorithm': Der Parameter legt den benutzten Algorithmus für das Fitting fest. Für Ebenen ist das Ergebnis des Fittings für die verschiedenen Algorithmen identisch. Ist *'fitting_algorithm'* auf *'least_squares'* gesetzt, werden die quadratischen Distanzen zwischen den Punkten und dem resultierenden Primitiv minimiert. Ist *'fitting_algorithm'* auf *'least_squares_huber'* gesetzt, ist der Ansatz ähnlich zu dem mit *'least_squares'* nur dass zusätzlich eine Gewichtung der Punkte vorgenommen wird, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden (siehe unten). Ist *'fitting_algorithm'* auf *'least_squares_tukey'* gesetzt, ist der Ansatz ähnlich zu dem mit *'least_squares'* nur dass zusätzlich eine Gewichtung der Punkte vorgenommen wird und Ausreißer nach dem Ansatz von Tukey ignoriert werden (siehe unten).

Für *'least_squares_huber'* und *'least_squares_tukey'* wird eine robuste Fehlerstatistik verwendet, um die Standardabweichung der Abstände der Objektpunkte (ohne Ausreißer) von dem approximierenden 3D Primitiv zu ermitteln. Ausreißer werden nach dem Ansatz von Tukey komplett eliminiert, während bei dem Verfahren nach Huber diese noch gering, d.h. linear, gewichtet werden. In der Praxis ist der Ansatz nach Tukey zu empfehlen.

Werteliste: 'least_squares', *'least_squares_huber'*, *'least_squares_tukey'*

Defaultwert: 'least_squares'

'min_radius': Der Parameter legt den minimalen Radius des Zylinders oder der Kugel in Metern fest. Falls ein Zylinder oder eine Kugel mit einem kleineren Radius angepasst wird, wird ein leeres 3D-Objektmodell zurückgegeben. Falls eine Ebene angepasst wird, wird dieser Parameter ignoriert.

Wertevorschläge: 0.01, *0.02*, *0.1*

Defaultwert: 0.01

'max_radius': Der Parameter legt den maximalen Radius des Zylinders oder der Kugel in Metern fest. Falls ein Zylinder oder eine Kugel mit einem größeren Radius angepasst wird, wird ein leeres 3D-Objektmodell zurückgegeben. Falls eine Ebene angepasst wird, wird dieser Parameter ignoriert.

Wertevorschläge: 0.02, *0.04*, *0.2*

Defaultwert: 0.2

'output_point_coord': Der Parameter legt fest, ob die für das Fitting genutzten 3D Punkte in das 3D-Ausgabeobjektmodell kopiert werden sollen. Wird *'copy_point_coord'* auf *'true'* gesetzt, werden die 3D Punkte kopiert. Wird *'copy_point_coord'* auf *'false'* gesetzt, werden keine 3D Punkte kopiert.

Werteliste: 'true', *'false'*

Defaultwert: 'true'

'output_xyz_mapping': Der Parameter legt fest, ob eine eventuell vorhandene Zuordnung der 3D Punkte zu Bildkoordinaten in das 3D-Ausgabeobjektmodell kopiert werden soll. Diese Information ist sinnvoll, wenn danach z.B. der Operator `object_model_3d_to_xyz` verwendet werden soll (z.B. für die Visualisierung). Ist *'output_xyz_mapping'* auf *'true'* gesetzt, wird die Zuordnung der 3D Punkte zu den Bildkoordinaten kopiert. Es ist zu beachten, dass der Parameter nur dann gültig ist, wenn eine Zuordnung der 3D Punkte zu den Bildkoordinaten im 3D-Eingabeobjektmodell enthalten ist. Wenn das 3D-Eingabeobjektmodell

von einem anderen 3D-Objektmodell, welches die Zuordnung der 3D Punkte zu den Bildkoordinaten enthält, durch Kopieren mit copy_object_model_3d abgeleitet wird, ist bereits beim Kopieren mit copy_object_model_3d darauf zu achten, dass die Zuordnung ebenfalls kopiert wird. Weiterhin ist der Parameter nur gültig, wenn auch die 3D Punkte in das 3D-Ausgabeobjektmodell kopiert werden, d.h., wenn *'output_point_coord'* auf *'true'* gesetzt ist. Ist *'output_xyz_mapping'* auf *'false'* gesetzt, wird die Zuordnung der 3D Punkte zu den Bildkoordinaten nicht kopiert.

Werteliste: 'true', 'false'

Defaultwert: 'false'

Die notwendige minimale Anzahl an 3D Punkten zum Anpassen einer Ebene ist drei. Die notwendige minimale Anzahl an 3D Punkten zum Anpassen einer Kugel ist vier. Die notwendige minimale Anzahl an 3D Punkten zum Anpassen eines Zylinders ist fünf.

Parameter

- ▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Eingabeobjektmodells.
- ▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
  Name des generischen Parameters.
  Parameteranzahl: GenParamName == GenParamValue
  Werteliste: GenParamName ∈ {'primitive_type', 'fitting_algorithm', 'min_radius', 'max_radius', 'output_point_coord', 'output_xyz_mapping'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.name-array ⇝ *string* / real / integer
  Wert des generischen Parameters.
  Parameteranzahl: GenParamValue == GenParamName
  Wertevorschläge: GenParamValue ∈ {'cylinder', 'sphere', 'plane', 'all', 'least_squares', 'least_squares_huber', 'least_squares_tukey', 0.01, 0.05, 0.1, 0.2, 'true', 'false'}
- ▷ **ObjectModel3DOut** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Ausgabeobjektmodells.

Ergebnis

fit_primitives_object_model_3d liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

xyz_to_object_model_3d, read_object_model_3d

Nachfolger

get_object_model_3d_params, object_model_3d_to_xyz, write_object_model_3d, clear_object_model_3d

Alternativen

segment_object_model_3d

Modul

3D Metrology

reduce_object_model_3d_by_view (Region : : ObjectModel3D, CamParam, Pose : ObjectModel3DReduced)

Reduzieren eines 3D-Objektmodells auf die Punkte, deren Projektion innerhalb einer Region liegt.

reduce_object_model_3d_by_view projiziert die Punkte des in ObjectModel3D übergebenen 3D-Objektmodells in eine Bildebene, die durch CamParam und Pose gegeben ist, und reduziert das 3D-Objektmodell

auf die Punkte innerhalb der Region, die durch `Region` gegeben ist. Die Punkte im 3D-Objektmodell werden zuerst mit der Pose transformiert und anschließend mit den Kameraparametern projiziert.

Alle Punkte, die außerhalb der Region liegen werden entfernt. Alle Dreiecke und Polygone, deren Ecken teilweise außerhalb der Region sind, werden ebenfalls entfernt.

Alternativ zu Kameraparametern und einer Pose kann auch ein in `ObjectModel3D` enthaltenes XYZ-Mapping verwendet werden. Dafür muss `CamParam` auf *'xyz_mapping'* oder ein leeres Tupel gesetzt werden und als `Pose` ein leeres Tupel übergeben werden. In diesem Fall werden die ursprünglichen Bildpositionen der 3D-Punkte verwendet und darauf geprüft, ob sie innerhalb der Region liegen.

Achtung

Kameras mit hyperzentrischen Objektiven werden nicht unterstützt.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Region zur Selektion von Teilen des 3D-Objektmodells.
- ▷ **`ObjectModel3D`** (input_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Objektmodells.
- ▷ **`CamParam`** (input_control) ..... campar ⇝ *real* / integer / string
  Kameraparameter.
  Wertevorschläge: `CamParam` ∈ {'xyz_mapping', []}
- ▷ **`Pose`** (input_control) ..... pose(-array) ⇝ *real* / integer
  3D Posen der Objekte im Kamerakoordinatensystem.
  Parameteranzahl: Pose == 7
- ▷ **`ObjectModel3DReduced`** (output_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle des reduzierten 3D-Objektmodells.

Beispiel

```
gen_object_model_3d_from_points (200*(rand(100)-0.5), \
                                 200*(rand(100)-0.5), \
                                 200*(rand(100)-0.5), ObjectModel3D)
gen_circle (Circle, 240, 320, 60)
CamParam := ['area_scan_telecentric_division',1,0,1,1,320,240,640,480]
Pose := [0,0,1,0,0,0,0]
reduce_object_model_3d_by_view (Circle, ObjectModel3D, CamParam, \
                                Pose, ObjectModel3DReduced)
dev_get_window (WindowHandle)
visualize_object_model_3d (WindowHandle, [ObjectModel3D, \
                           ObjectModel3DReduced], CamParam, Pose, \
                           ['color_0', 'point_size_1'], ['blue',6], \
                           [], [], [], PoseOut)
```

Ergebnis

`reduce_object_model_3d_by_view` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`project_object_model_3d`, `object_model_3d_to_xyz`

Siehe auch

`select_points_object_model_3d`

Modul

3D Metrology

segment_object_model_3d (: : ObjectModel3D, GenParamName, GenParamValue : ObjectModel3DOut)

Segmentieren von 3D Punkten mit ähnlichen Eigenschaften.

Der Operator segment_object_model_3d segmentiert eine Menge von 3D Punkten in Teilmengen von benachbarten 3D Punkten mit ähnlichen Eigenschaften, z.B. einer gleichen Normalenrichtung oder Krümmung. Standardmäßig wird anschließend versucht, an die segmentierten Punktteilmengen 3D Primitive, d.h. einfache 3D Formen wie Ebenen, Kugeln oder Zylinder, anzupassen. Die ursprüngliche Punktmenge ist durch ein 3D-Objektmodell mit dem Handle ObjectModel3D gegeben. Der Operator gibt ein oder mehrere Handles von 3D-Objektmodellen (ObjectModel3DOut) zurück, welche jeweils die 3D Punkte der segmentierten Teilmengen sowie weitere Informationen, z.B. die Parameter der angepassten 3D Primitive, enthalten. Die enthaltene Information kann mit dem Operator get_object_model_3d_params abgerufen werden.

Vor der Segmentierung sollte das 3D-Eingabeobjekt mit dem Operator prepare_object_model_3d für die Segmentierung vorbereitet werden. Ist das 3D-Eingabeobjektmodell nicht vorbereitet, wird der Operator intern in segment_object_model_3d aufgerufen, um das 3D-Eingabeobjektmodell um Attribute zu erweitern, die vorher nur implizit im Model vorhanden waren.

Um die Segmentierung und das Fitting zu steuern, können in GenParamName und GenParamValue generische Parameter angepasst werden. In den meisten Fällen reichen die Standardwerte aus, so dass keine Anpassung nötig ist. Die folgenden generischen Parameter und die dazugehörigen Werte sind verfügbar:

'max_orientation_diff': Der Parameter legt fest, bis zu welchem maximalen Wert sich die Winkel zwischen den Punktnormalen zweier benachbarter 3D Punkte unterscheiden dürfen, damit diese in die gleiche Punktteilmenge eingeordnet werden. Für einen Zylinder oder eine Kugel hängt dieser Wert von der Größe des Objekts und der Entfernung zwischen den benachbarten 3D Punkten ab. D.h., wenn der Zylinder oder die Kugel einen sehr kleinen Radius hat oder die 3D Punktemenge eine geringe Punktdichte aufweist, muss der Wert höher gesetzt werden. Für eine Ebene ist der Wert unabhängig von der Größe des Objekts und kann auf einen kleinen Wert gesetzt werden.

Wertevorschläge: 0.10, *0.15*, *0.20*

Defaultwert: 0.15

'max_curvature_diff': Der Parameter legt fest, um welchen maximalen Wert sich die Krümmungen der Oberflächen an den Positionen zweier benachbarter 3D Punkte unterscheiden darf, damit die Punkte in die gleiche Punktteilmenge eingeordnet werden. Dieser Wert hängt vom Rauschen der 3D Punkte ab. D.h., wenn das Rauschen sehr hoch ist, muss der Wert höher gesetzt werden. Im Allgemeinen nimmt die Anzahl der 3D-Ausgabeobjektmodelle für größere Werte ab, da mehr 3D Punkte zu einer Teilmenge von 3D Punkten zusammengeführt werden.

Wertevorschläge: 0.03, *0.04*, *0.05*

Defaultwert: 0.05

'min_area': Der Parameter legt die minimale Anzahl an Punkten fest, aus der zusammenhängende Punktteilmengen nach der Segmentierung bestehen müssen. Für eine Punktteilmenge, die weniger Punkte enthält, werden die Punkte gelöscht und es wird kein 3D-Ausgabeobjekthandle erstellt.

Wertevorschläge: 1, *10*, *100*

Defaultwert: 100

'fitting': Der Parameter legt fest, ob eine Anpassung durch 3D Primitive erfolgen soll. Ist *'fitting'* auf *'true'* gesetzt, was standardmäßig der Fall ist, wird eine Anpassung durch 3D Primitive vorgenommen. Die zurückgegebenen Handles der 3D-Ausgabeobjekte enthalten dann zusätzlich zu den 3D Punkten der segmentierten Punktteilmengen die Parameter der 3D Primitive. Die Ausgabeparameter eines Zylinders, einer Kugel oder einer Ebene sind für den Operator fit_primitives_object_model_3d beschrieben. Falls *'fitting'* auf *'false'* gesetzt ist, wird nur die Segmentierung durchgeführt und die zurückgegebenen Handles enthalten nur die 3D Punkte der segmentierten Punktteilmengen. Eine nachfolgende Anpassung durch 3D Primitive kann mit dem Operator fit_primitives_object_model_3d durchgeführt werden.

Werteliste: 'false', *'true'*

Defaultwert: 'true'

'output_xyz_mapping': Der Parameter legt fest, ob eine eventuell vorhandene Zuordnung der segmentierten 3D Punkte zu Bildkoordinaten in das 3D-Ausgabeobjektmodell kopiert werden soll. Diese Information ist sinnvoll, wenn danach z.B. der Operator object_model_3d_to_xyz verwendet werden soll (z.B. für die Visualisierung). Ist *'output_xyz_mapping'* auf *'true'* gesetzt, wird die Zuordnung der 3D Punkte zu den

Bildkoordinaten kopiert. Es ist zu beachten, dass der Parameter nur dann gültig ist, wenn eine Zuordnung der 3D Punkte zu den Bildkoordinaten im 3D-Eingabeobjektmodell enthalten ist. Wenn das 3D-Eingabeobjektmodell von einem anderen 3D-Objektmodell, welches die Zuordnung der 3D Punkte zu den Bildkoordinaten enthält, durch Kopieren mit `copy_object_model_3d` abgeleitet wird, ist bereits beim Kopieren mit `copy_object_model_3d` darauf zu achten, dass die Zuordnung ebenfalls kopiert wird. Weiterhin ist der Parameter nur gültig, wenn auch die 3D Punkte in das 3D-Ausgabeobjektmodell kopiert werden, d.h., wenn *'output_point_coord'* auf *'true'* gesetzt ist. Ist *'output_xyz_mapping'* auf *'false'* gesetzt, wird die Zuordnung der 3D Punkte zu den Bildkoordinaten nicht kopiert.

Werteliste: 'true', 'false'

Defaultwert: 'false'

'primitive_type','fitting_algorithm', 'min_radius','max_radius', 'output_point_coord': Diese Parameter werden genutzt, wenn *'fitting'* auf *'true'* gesetzt ist. Die Bedeutung und Verwendung der Parameter ist für den Operator `fit_primitives_object_model_3d` beschrieben.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Eingabeobjektmodells.

▷ **`GenParamName`** (input_control) . . . . . attribute.name-array ⇝ *string*
Name des generischen Parameters.
Parameteranzahl: GenParamName == GenParamValue
Werteliste: `GenParamName` ∈ {'max_orientation_diff', 'max_curvature_diff', 'min_area', 'primitive_type', 'fitting_algorithm', 'min_radius', 'max_radius', 'output_point_coord', 'output_xyz_mapping'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.name-array ⇝ *string* / real / integer
Wert des generischen Parameters.
Parameteranzahl: GenParamValue == GenParamName
Wertevorschläge: `GenParamValue` ∈ {0.15, 0.05, 100, 'true', 'false', 'cylinder', 'sphere', 'plane', 'all', 'least_squares', 'least_squares_huber', 'least_squares_tukey'}

▷ **`ObjectModel3DOut`** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Ausgabeobjektmodells.

Ergebnis

`segment_object_model_3d` liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`xyz_to_object_model_3d`, `read_object_model_3d`, `prepare_object_model_3d`

Nachfolger

`get_object_model_3d_params`, `object_model_3d_to_xyz`, `write_object_model_3d`, `clear_object_model_3d`

Siehe auch

`fit_primitives_object_model_3d`

Modul

3D Metrology

`select_points_object_model_3d` `( : : ObjectModel3D, Attrib, MinValue, MaxValue : ObjectModel3DThresholded )`

Anwenden eines Schwellenwerts auf Attribute in 3D-Objektmodellen.

`select_points_object_model_3d` wählt Punkte des 3D-Objektmodells `ObjectModel3D` anhand der in `Attrib`, `MinValue` und `MaxValue` übergebenen Attribute und Schwellenwerte aus. Die ausgewählten Punkte werden im 3D-Objektmodell `ObjectModel3DThresholded` zurückgegeben. Alle Attribute, die mit den Punkten zusammenhängen, wie z.B. Polygone oder Dreiecke, werden entsprechend angepasst, so dass keine Information mehr auf die entfernten Punkte verweist.

`Attrib` kann entweder ein Tupel von Zahlen enthalten, das genau so viele Einträge enthält wie `ObjectModel3D` Punkte hat, oder eine Liste von Attributnamen, auf die die Schwellenwerte angewendet werden.

Wenn `Attrib` ein Tupel von Zahlen enthält, muss in `MinValue` und `MaxValue` jeweils genau eine Zahl übergeben werden. Alle Punkte, für die der Eintrag in `Attrib` zwischen diesen beiden Schwellenwerten liegen, werden in das Ausgabeobjektmodell `ObjectModel3DThresholded` übernommen.

Alternativ kann `Attrib` eine Liste von Attributnamen enthalten, die sich auf Eigenschaften des 3D-Objektmodells `ObjectModel3D` beziehen. Alle Punkte, für die der im Attribut `Attrib` gespeicherte Wert innerhalb des durch `MinValue` und `MaxValue` spezifizierten Intervalls liegt, werden behalten. `MinValue` und `MaxValue` müssen jeweils genau so viele Einträge haben wie `Attrib`. Wenn `Attrib` mehrere Elemente enthält, werden nur die Punkte in die Ausgabe übernommen, die alle Kriterien erfüllen.

Je nach den Eigenschaften des Objektmodells sind in `Attrib` als Attributnamen möglich:

'point_coord_x': Die x-Koordinaten der 3D Punkte.

'point_coord_y': Die y-Koordinaten der 3D Punkte.

'point_coord_z': Die z-Koordinaten der 3D Punkte.

'point_normal_x': Die x-Komponenten der 3D Punktnormalen der 3D Punkte.

'point_normal_y': Die y-Komponenten der 3D Punktnormalen der 3D Punkte.

'point_normal_z': Die z-Komponenten der 3D Punktnormalen der 3D Punkte.

'mapping_row': Die Zeilenkomponenten der 2D Abbildung der 3D Punkte.

'mapping_col': Die Spaltenkomponenten der 2D Abbildung der 3D Punkte.

'neighbor_distance':

'neighbor_distance N': Der Abstand des N-ten Nachbarpunktes. N muss ein positiver Integer-Wert sein und ist standardmäßig 25. Für jeden Punkt werden die übrigen Punkte anhand ihres Abstands sortiert, und der Abstand des N-ten Punktes für den Schwellenwert verwendet.

'num_neighbors X': Die Anzahl an Nachbarn mit einem Abstand von höchstens X. Dieser Filter kann verwendet werden, um dünn besetzte Gebiete des 3D-Objektmodells zu entfernen, wie etwa Ausreißerpunkte oder Punkte, die durch Glättungen zwischen 3D-Oberflächen entstehen.

'num_neighbors_fast X': Die ungefähre Anzahl an Nachbarn mit einem Abstand von höchstens X. Die Abstände werden mittels Voxeln approximiert, was üblicherweise schneller ist als die exakte Berechnung für *'num_neighbors'*.

Erweiterte Attribute: Es kann der Name eines erweiterten Attributs des Types *'vertices'* angegeben werden. Dessen Werte dienen dann als Entscheidungsgrundlage, um Punkte beizubehalten oder nicht.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle von 3D-Objektmodellen.

▷ **`Attrib`** (input_control) . . . . . string(-array) ⇝ *string*
Attribute, auf die ein Schwellenwert angewandt werden soll.
Defaultwert: 'point_coord_z'
Werteliste: `Attrib` ∈ {'point_coord_x', 'point_coord_y', 'point_coord_z', 'point_normal_x', 'point_normal_y', 'point_normal_z', 'mapping_row', 'mapping_col', 'neighbor_distance', 'num_neighbors', 'num_neighbors_fast'}

▷ **`MinValue`** (input_control) . . . . . number(-array) ⇝ *real* / integer
Untere Schranke für die Attribute.
Defaultwert: 0.5

▷ **`MaxValue`** (input_control) . . . . . number(-array) ⇝ *real* / integer
Obere Schranke für die Attribute.
Defaultwert: 1.0

▷ **`ObjectModel3DThresholded`** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle der reduzierten 3D-Objektmodelle.

Beispiel

```
gen_object_model_3d_from_points (rand(100), rand(100),\
                                 rand(100), ObjectModel3D)
select_points_object_model_3d (ObjectModel3D, 'point_coord_z',\
                               0.5, 1, ObjectModel3DThresholded)
get_object_model_3d_params (ObjectModel3DThresholded, 'num_points',\
                            NumPoints)
```

Ergebnis

`select_points_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt. Falls die notwendigen Punkte im 3D-Objektmodell nicht vorhanden sind, d.h. ein leeres Objektmodell übergeben wurde, wird der Fehler 9515 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`connection_object_model_3d`, `project_object_model_3d`, `object_model_3d_to_xyz`

Siehe auch

`connection_object_model_3d`, `reduce_object_model_3d_by_view`

Modul

3D Metrology

4.4 Transformationen

affine_trans_object_model_3d (: : ObjectModel3D, HomMat3D : ObjectModel3DAffineTrans)

Anwenden einer beliebigen affinen 3D-Transformation auf 3D-Objektmodelle.

`affine_trans_object_model_3d` wendet beliebige affine 3D-Transformationen (Skalierung, Drehung, Verschiebung) auf 3D-Objektmodelle an und liefert den Handle der transformierten 3D-Objektmodelle zurück. Die Transformation wird durch eine oder mehrere homogene Transformationsmatrizen beschrieben, die in `HomMat3D` übergeben werden. Die Anzahl der Transformationsmatrizen ist entweder eins oder entspricht der Anzahl an 3D-Objektmodellen in `ObjectModel3D`.

Die Transformationsmatrizen können mit Hilfe der Operatoren `hom_mat3d_identity`, `hom_mat3d_scale`, `hom_mat3d_rotate`, `hom_mat3d_translate` etc. aufgebaut werden oder das Resultat von `pose_to_hom_mat3d` sein (siehe `affine_trans_point_3d`).

Der Operator `affine_trans_object_model_3d` wird in der Regel nicht für das 3D-Matching benötigt. Falls dort eine Rotation des 3D-Objektmodells in eine Referenzorientierung erfolgen soll, wird empfohlen, stattdessen in `create_shape_model_3d` entsprechende Werte für die Parameter `RefRotX`, `RefRotY`, `RefRotZ` und `OrderOfRotation` zu übergeben.

Der Operator unterstützt ein oder mehrere 3D-Objektmodelle mit einer Transformationsmatrix (N:1), ein 3D-Objektmodell mit ein oder mehreren Transformationsmatritzen (1:N) sowie eine Transformationsmatrix pro übergebenem 3D-Objektmodell, die dann nur auf das jeweilige 3D-Objektmodell angewandt wird (N:N). Dabei kann N auch für 0 stehen, d.h. es kann keine Matrix oder kein Objektmodell übergeben werden, in welchem Fall in `ObjectModel3DAffineTrans` ein leeres Tupel zurückgegeben wird. Dies kann z.B. verwendet werden, um die Ergebnisse von anderen Operatoren zu transformieren, ohne vorher zu prüfen ob tatsächlich mindestens ein Ergebnis vorhanden ist.

Achtung

`affine_trans_object_model_3d` transformiert Attribute vom Typ Punktkoordinaten, Punktnormalen und vorbereitetem 3D-Formmodell für das formbasierte 3D-Matching. Alle anderen Attribute werden kopiert ohne sie zu verändern. Um 3D-Primitive zu transformieren, muss der Operator `rigid_trans_object_model_3d` verwendet werden.

Parameter

- ▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle der 3D-Objektmodelle.
- ▷ **`HomMat3D`** (input_control) . . . . . hom_mat3d(-array) ⇝ *real*
  Transformationsmatrizen.
- ▷ **`ObjectModel3DAffineTrans`** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle der transformierten 3D-Objektmodelle.

Ergebnis

`affine_trans_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`project_object_model_3d`, `object_model_3d_to_xyz`

Siehe auch

`affine_trans_point_3d`, `rigid_trans_object_model_3d`, `projective_trans_object_model_3d`

Modul

3D Metrology

`connection_object_model_3d` `( : : ObjectModel3D, Feature, Value : ObjectModel3DConnected )`

Berechnen der Zusammenhangskomponenten eines 3D-Objektmodells.

`connection_object_model_3d` berechnet innerhalb des 3D-Objektmodells `ObjectModel3D` die Zusammenhangskomponenten. Der Zusammenhang zweier Komponenten wird mithilfe eines Attributs und einem darauf definierten Distanzmaß ermittelt, das durch den Parameter `Feature` festgelegt wird:

'distance_3d': Alle Punkte, deren euklidischer Abstand zueinander kleiner als `Value` ist, werden als zusammenhängend angesehen.

'angle': Punkte werden als zusammenhängend angesehen, falls der Winkel zwischen den Normalen der Punkte kleiner als `Value` ist. `Value` wird in Bogenmaß angegeben und muss zwischen 0 und pi liegen.

Voraussetzung: Das 3D-Objektmodell muss Normalen enthalten. Diese können mit `surface_normals_object_model_3d` berechnet werden.

'distance_mapping': Punkte werden als zusammenhängend angesehen, falls der Abstand zwischen den im Pixel-Mapping gespeicherten Pixel kleiner ist als `Value`. Um das gleiche Verhalten wie im Operator `connection` zu erhalten und eine 8-er Nachbarschaft zu erhalten, sollte für `Value` ein Wert größer als 1.5 angegeben werden.

Voraussetzung: Das 3D-Objektmodell muss Pixel-Mapping enthalten. Das ist der Fall, wenn das 3D-Objektmodell mit `xyz_to_object_model_3d` erstellt wurde.

'mesh': Bestimmt die Zusammenhangskomponenten in der Vermaschung des 3D-Objektmodells. Ein Dreieck oder Polygon verbindet hierbei mehrere 3D-Punkte zu einer Zusammenhangskomponente. Gibt es zwei Polygone, die den selben Punkt enthalten, werden alle Punkte beider Polygone zu einer Zusammenhangskomponente zusammengefasst.

Voraussetzung: Das 3D-Objektmodell muss trianguliert worden sein, was mit `triangulate_object_model_3d` erreicht werden kann. Falls das 3D-Objektmodell schon ein Pixel-Mapping enthält, kann alternativ auch `prepare_object_model_3d` genutzt werden, um das 3D-Objekt schnell zu triangulieren. Hierbei ist der Parameter `Purpose` auf den Wert *'segmentation'* zu setzen.

'lines': Bestimmt die Zusammenhangskomponenten bezüglich der enthaltenen Linienzüge des 3D-Objektmodells.

Voraussetzung: Das 3D-Objektmodell muss Linienzüge enthalten. Diese können mit `intersect_plane_object_model_3d` berechnet werden.

Die benötigten Attribute können auch manuell mit `set_object_model_3d_attrib` oder `set_object_model_3d_attrib_mod` gesetzt werden. Es ist möglich, dass 3D-Objektmodell bereits die benötigten Attribute enthält, insbesondere, wenn es mit `read_object_model_3d` eingelesen wurde, oder mit `deserialize_object_model_3d` deserialisiert wurde. Um zu überprüfen, ob das benötigte Attribut vorhanden ist, kann `get_object_model_3d_params` genutzt werden.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle eines 3D-Objektmodells.

▷ **`Feature`** (input_control) . . . . . string(-array) ⇝ *string*
Attribut, aufgrund dessen die Zusammenhangskomponenten berechnet werden.
Defaultwert: 'distance_3d'
Werteliste: `Feature` ∈ {'distance_3d', 'angle', 'distance_mapping', 'mesh', 'lines'}

▷ **`Value`** (input_control) . . . . . number(-array) ⇝ *real* / integer
Obere Schranke für den Abstand zweier Zusammenhangskomponenten.
Defaultwert: 1.0
Wertevorschläge: `Value` ∈ {1.0, 1.1, 1.5, 10.0, 100.0}

▷ **`ObjectModel3DConnected`** (output_control) . . . . . object_model_3d-array ⇝ *handle*
Handle der 3D-Objektmodelle der Zusammenhangskomponenten.

Beispiel

```
gen_object_model_3d_from_points (rand(100), rand(100),\
                                 rand(100), ObjectModel3D)
connection_object_model_3d (ObjectModel3D, 'distance_3d', 0.2,\
                            ObjectModel3DConnected)
dev_get_window (WindowHandle)
visualize_object_model_3d (WindowHandle, [ObjectModel3DConnected], [], [],\
                           ['colored'], [12], [], [], [], PoseOut)
```

Ergebnis

`connection_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`, `select_points_object_model_3d`

Nachfolger

`project_object_model_3d`, `object_model_3d_to_xyz`

Siehe auch

select_points_object_model_3d

Modul

3D Metrology

convex_hull_object_model_3d (: : ObjectModel3D : ObjectModel3DConvexHull)

Berechnen der konvexen Hülle eines 3D-Objektmodells.

convex_hull_object_model_3d berechnet die konvexe Hülle eines 3D-Objektmodells. Die konvexe Hülle wird in Form eines Handles eines 3D-Objektmodells in ObjectModel3DConvexHull zurückgegeben.

Sollte eine der Dimensionen der Eingabepunkte keine Ausdehnung ausweisen, werden Linien statt Dreiecken für die Darstellung der konvexen Hülle verwendet.

Parameter

- ▷ **ObjectModel3D** (input_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Objektmodells.
- ▷ **ObjectModel3DConvexHull** (output_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Objektmodells, das die konvexe Hülle des Eingabeobjektes beschreibt.
  Parameteranzahl: ObjectModel3DConvexHull == ObjectModel3D

Beispiel

```
gen_object_model_3d_from_points (rand(20)-0.5, rand(20)-0.5,\
                                 rand(20)-0.5, ObjectModel3D)
convex_hull_object_model_3d (ObjectModel3D, ObjectModel3DConvexHull)
dev_get_window (WindowHandle)
visualize_object_model_3d (WindowHandle, [ObjectModel3DConvexHull], \
                           [], [], [], [], [], [], [], PoseOut)
```

Ergebnis

convex_hull_object_model_3d liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_object_model_3d, connection_object_model_3d, select_points_object_model_3d

Nachfolger

project_object_model_3d

Modul

3D Metrology

edges_object_model_3d (: : ObjectModel3D, MinAmplitude, GenParamName, GenParamValue : ObjectModel3DEdges)

Finden von Kanten in einem 3D-Objektmodell.

edges_object_model_3d findet 3D-Kanten im 3D-Objektmodell ObjectModel3D und gibt diese im 3D-Objektmodell ObjectModel3DEdges zurück.

ObjectModel3D muss ein XYZ-Mapping enthalten. Das Mapping wird beispielsweise bei der Erzeugung eines 3D-Objektmodells mittels xyz_to_object_model_3d erstellt oder von einigen Sensoren mit zurückgeliefert. Der Parameter MinAmplitude gibt die minimale Kantenamplitude an, d.h. die Mindestgröße des Sprungs an einer 3D-Kante. Der Schwellenwert wird in der gleichen Einheit gesetzt, die in ObjectModel3D verwendet wird.

Die extrahierten Kantenpunkte sind eine Untermenge der Punkte des Eingabeobjektmodells. Zusätzlich zu den Koordinaten der Kanten enthält ObjectModel3DEdges in den Normalenvektoren die Blickrichtung vom Sensorursprung zu den Kantenpunkten. Die Attribute *'edge_dir_x'*, *'edge_dir_y'* und *'edge_dir_z'* enthalten für jeden Kantenpunkt einen normalisierten Vektor, welcher senkrecht auf Blickrichtung und Kantenrichtung steht. Die Attribute sind derart gesetzt, dass ObjectModel3DEdges für die 3D-Kanten in find_surface_model verwendet werden kann.

Optional können generische Parameter angegeben werden um die Kantenextraktion zu beeinflussen. Falls erwünscht, können die generischen Parameter und deren Werte in GenParamName und GenParamValue übergeben werden. Die folgenden Werte für GenParamName sind möglich:

'max_gap': Dieser Parameter gibt die maximale Größe einer Lücke in Pixeln in den XYZ-Bildern an, damit die Lücke geschlossen wird. Für Lücken die einen größeren Durchmesser haben als dieser Wert werden 3D-Kanten an den Rändern der Lücke erzeugt, bei Lücken mit einem kleineren Durchmesser entsprechend nicht. Dadurch können beispielsweise Kanten unterdrückt werden, die durch kleinere Lücken in den 3D-Daten entstehen die der Sensor nicht rekonstruieren konnte, sowie Kanten an der weiter hinten liegenden Fläche eines Sprungs in den Tiefendaten. Dieser Wert sollte beispielsweise für besonders hochauflösende Sensoren erhöht werden.

Standardwert: 30.

'estimate_viewpose': Dieser Parameter kann benutzt werden, um die automatische Berechnung der Blickrichtung auszuschalten und einen manuellen Blickpunkt zu setzen.

Standardwert: 'true'.

'viewpoint': Dieser Parameter wirkt sich nur dann aus, wenn *'estimate_viewpose'* auf *'false'* gesetzt wird. Er gibt die 3D-Position des Blickpunktes an, von dem aus die 3D-Punkte in ObjectModel3D gesehen werden. Der Blickpunkt wird verwendet, um die Blickrichtungen und die Kantenrichtungen zu berechnen und liegt standardmäßig im Ursprung *'0 0 0'*. Falls der Blickpunkt nicht im Ursprung liegt, etwa weil das 3D-Objektmodell mittels rigid_trans_object_model_3d transformiert wurde oder falls der 3D-Sensor eine ähnliche Transformation durchgeführt hat, muss der ursprüngliche Blickpunkt gesetzt werden. Dazu wird in GenParamValue ein String mit den drei Koordinaten (x, y und z) des Blickpunktes, getrennt durch Leerzeichen, übergeben. Der Blickpunkt wird im gleichen Koordinatensystem wie ObjectModel3D definiert. Zur Benutzung dieses Parameters müssen die Werte der X-, Y-, und Z- Bilder aus object_model_3d_to_xyz jeweils steigende Werte von links nach rechts, oben nach unten, und näher nach weiter entfernt von der Kamera haben.

Standardwert: '0 0 0'.

Parameter

- ▷ **ObjectModel3D** (input_control) . . . . . object_model_3d ⇝ *handle*
  Handle des 3D-Objektmodells, dessen Kanten berechnet werden sollen.
- ▷ **MinAmplitude** (input_control) . . . . . number ⇝ *real* / integer
  Kantenschwellenwert.
- ▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'max_gap', 'estimate_viewpose', 'viewpoint'}
- ▷ **GenParamValue** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Werte der generischen Parameter.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {'0 0 0', 10, 30, 100, 'true', 'false'}
- ▷ **ObjectModel3DEdges** (output_control) . . . . . object_model_3d ⇝ *handle*
  3D-Objektmodell, welches die berechneten Kanten enthält.

Ergebnis

edges_object_model_3d liefert den Wert TRUE wenn alle Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`find_surface_model`, `find_surface_model_image`, `refine_surface_model_pose`, `refine_surface_model_pose_image`

Modul

3D Metrology

fuse_object_model_3d (: : ObjectModel3D, BoundingBox, Resolution, SurfaceTolerance, MinThickness, Smoothing, NormalDirection, GenParamName, GenParamValue : ObjectModel3DFusion)

Fusionieren von 3D-Objektmodellen.

`fuse_object_model_3d` fusioniert mehrere Punktwolken, welche die Oberfläche eines Objekts repräsentieren, zu einer wasserdichten Oberfläche `ObjectModel3DFusion`. Der Operator vereinfacht die Nachverarbeitung von Punktwolken, welche bereits im selben Koordinatensystem erfasst sind. Insbesondere können folgende Prozesse oft einfach und mit hoher Qualität abgewickelt werden: Vereinigung, Unterdrückung von Ausreißern, Kompromiss aus Glättung und Kantenerhaltung, äquidistantes Subsampling, Löcher füllen, Vermaschung der resultierenden Oberfläche. Der Operator `fuse_object_model_3d` hat allerdings eine relativ lange Rechenzeit.

Sollen Punktwolken fusioniert werden, welche über Stereo-Rekonstruktion aufgenommen wurden, sollte stattdessen `reconstruct_surface_stereo` verwendet werden.

Ablauf

1. Aufnahme der Punktwolken; diese in ein einheitliches Koordinatensystem bringen, z.B. mittels `register_object_model_3d_pair` und `register_object_model_3d_global`.
2. Berechnen der Dreiecke oder Punktnormalen für die Punktwolken mittels `triangulate_object_model_3d` oder `surface_normals_object_model_3d`, sofern diese nicht bereits vorhanden sind. Die Triangulierung ist geeigneter, falls die Oberflächen viele Ausreißer oder zu schließende Löcher haben. Ansonsten, im Falle von glatten Oberflächen, kann man mit Punktnormalen arbeiten.
3. Inspizieren der Normalen des Eingabe-Modells mittels `visualize_object_model_3d` mit der Option `GenParamName` `'disp_normals'` oder `dev_inspect_ctrl`. Die Punkt- oder Dreiecksnormalen müssen konsistent entweder nach innen oder außen gerichtet sein. Der Parameter `NormalDirection` muss dann entsprechend auf *'inwards'* oder *'outwards'* gesetzt werden.
4. Festlegen des umschließenden Quaders über den Parameter `BoundingBox`. Für eine erste Näherung des umschließenden Quaders `BoundingBox` kann `get_object_model_3d_params` verwendet werden, mit `GenParamName` gesetzt auf `'bounding_box1'`.
5. Die Werte für die initialen Parameter kann man wie folgt abschätzen: Einen groben Wert für `Resolution` (z.B. 1/100 der Raumdiagonalen des umschließenden Quaders `BoundingBox`), `SurfaceTolerance` etwas größer (z.B. 5*`Resolution`), `MinThickness` als minimale Dicke des Objekts (sollte die Eingabe-Punktwolke das Objekt nur von einer Seite aus darstellen, wird empfohlen, den Wert sehr hoch zu setzen, damit das Objekt erst durch den umschließenden Quader `BoundingBox` abgeschnitten wird) und `Smoothing` mit Wert *1.0*.
6. Ausführen des Operators `fuse_object_model_3d` und nachjustieren der Parameterwerte zur Verbesserung des Ergebnisses bezüglich Qualität und Rechenzeit, siehe auch unten. Zur Vermeidung langer Rechenzeiten ist eine Auflösung `Resolution` empfohlen, die gerade noch ausreichend ist, um Objektdetails erkennen zu können, während die anderen Parameter eingestellt werden. Zur Optimierung sollten auch die zusätzlichen Parameter von `GenParamName` in Erwägung gezogen werden.

Parameterbeschreibung

In dem HDevelop-Beispiel `fuse_object_model_3d_workflow` wird gezeigt, wie die hier aufgelisteten Parameter für eine bestimmte Anwendung optimiert werden können.

Die Eingabe-Punktwolken `ObjectModel3D` müssen in einem gemeinsamen Koordinatensystem sein und stellen die initiale Oberfläche dar. Des Weiteren müssen diese Punktwolken Dreiecke oder Punktnormalen beinhalten. Sollten beide Attribute vorhanden sein, werden, aufgrund der Geschwindigkeitsvorteile, als Standardeinstellung Punktnormalen verwendet. Wenn die Verwendung von Dreiecken gewünscht ist, müssen die Dreiecke und Punkte mittels `copy_object_model_3d` in ein neues Objektmodell überführt werden. Mit Oberflächen, die viele Ausreißer oder zu füllende Löcher beinhalten, empfehlen wir, die Triangulierung zu verwenden, während glatte Oberflächen mittels Punktnormalen verarbeitet werden können. Die Punkt- oder Dreiecksnormalen müssen entweder konsistent ins Innere oder Äußere des Objekts gerichtet sein.

`NormalDirection` gibt an, ob die Punkt- oder Dreiecksnormalen nach innen (*'inwards'*), oder außen (*'outwards'*) gerichtet sind. Wird nur ein einzelner Wert angegeben, so wird dieser für alle Eingabe-Modelle verwendet. Ansonsten muss die Anzahl Eingabewerte der Anzahl an Eingabe-Modellen entsprechen.

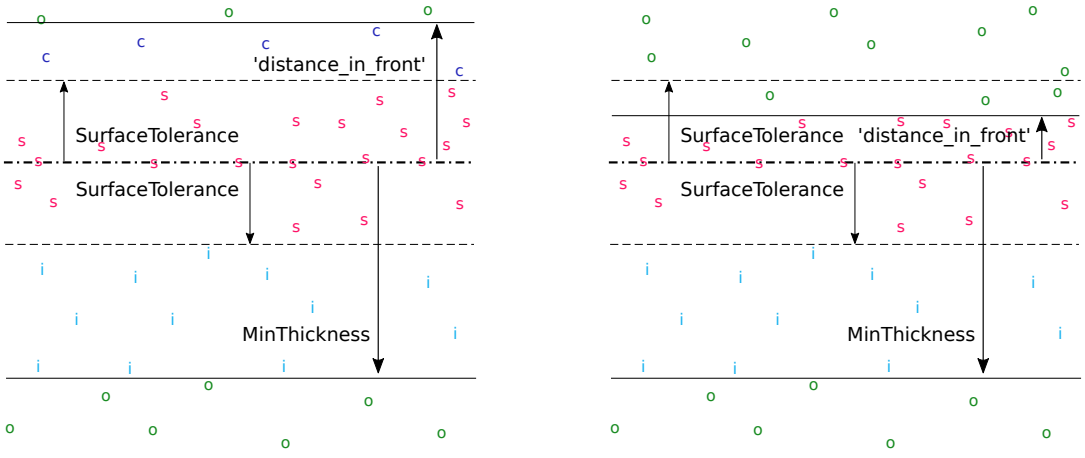
Der umschließende Quader `BoundingBox` definiert den berücksichtigten Bereich, sowohl die Eingabepunkte als auch die Ausgabepunkte betreffend. Punkte außerhalb des umschließenden Quaders werden verworfen, ebenso auch Dreiecke mit zumindest einem Punkt außerhalb des umschließenden Quaders. Der umschließende Quader `BoundingBox` wird durch das Tupel `[x1,y1,z1,x2,y2,z2]` festgelegt, welches die beiden gegenüberliegenden Ecken `P1=[x1,y1,z1]` und `P2=[x2,y2,z2]` des Quaders angibt (mit Kanten parallel zu den Koordinatenachsen). Damit der umschließende Quader festgelegt werden kann, muss `P1` in der unteren vorderen Ecke und `P2` in der oberen hinteren Ecke des Quaders sein, d.h. `x1<x2`, `y1<y2` und `z1<z2`. Der umschließende Quader wird in der gleichen Einheit gesetzt, die in `ObjectModel3D` verwendet wird. Der Operator versucht, eine geschlossene Oberfläche zu kreieren. Falls die Eingabe-Punktwolke das Objekt nur von einer Seite repräsentiert, will man üblicherweise den umschließenden Quader als Begrenzung des unbekannten Teils. Dafür setzt man `MinThickness` z.B. auf einen Wert größer oder gleich der Länge der Diagonale des umschließenden Quaders (welche über `get_object_model_3d_params` mit dem Parameter *'diameter_axis_aligned_bounding_box'* erhalten werden kann). Wird ein Objekt durch eine Seite des umschließenden Quaders abgeschnitten, hat das Objekt an dieser spezifischen Ebene der Oberfläche keine Punkte, d.h. es ist hohl. Um einen eventuellen unbekannten Teil des Objekts an der richtigen Stelle zu begrenzen, muss die Eingabe-Punktwolke möglicherweise rotiert werden, da die Kanten des umschließenden Quaders immer parallel zu den Koordinatenachsen sind. Eine solche Rotation ist z. B. möglich über die Operatoren `affine_trans_object_model_3d` oder `rigid_trans_object_model_3d`.

Die Auflösung `Resolution` definiert die Distanz benachbarter Stützstellen in Richtung der Koordinatenachsen bei der Diskretisierung des umschließenden Quaders, `BoundingBox`. `Resolution` wird in der gleichen Einheit gesetzt, die in `ObjectModel3D` verwendet wird. Ein zu kleiner Wert, d.h. eine zu hohe Auflösung, wird die Rechenzeit unnötig erhöhen während ein zu großer Wert zu hohem Detailverlust bei der Rekonstruktion führen kann. Wir empfehlen mit einer groben Auflösung zu beginnen. Bei Änderung der Auflösung `Resolution` muss `Smoothing` eventuell adaptiert werden. Zudem muss zur Vermeidung von Diskretisierungseffekten `Resolution` stets kleiner sein als `SurfaceTolerance`.

`SurfaceTolerance` legt fest, wie viel Rauschen der Eingabe-Punktwolke von der Innen- und Außenseite her zur Oberfläche vereint wird. Die Ausnahme ist gegeben durch `SurfaceTolerance` größer als *'distance_in_front'*, dann bestimmt *'distance_in_front'* die Dicke der Oberfläche zur Außenseite. `SurfaceTolerance` wird in der gleichen Einheit gesetzt, die in `ObjectModel3D` verwendet wird. `SurfaceTolerance` gibt die Dicke der Oberfläche an, während `MinThickness` die Dicke des gefüllten Objekts nach innen bestimmt. Daher muss `SurfaceTolerance` kleiner sein als `MinThickness`. Punkte, die laut `NormalDirection` (sowie `GenParamName`=*'angle_threshold'*) auf der Innenseite der Objektoberfläche liegen, werden somit als sicher innerhalb eines Objekts betrachtet, wenn ihre Distanz zur initialen Oberfläche größer ist als `SurfaceTolerance`, aber kleiner als `MinThickness`. Um Diskretisierungseffekte zu vermeiden, muss `SurfaceTolerance` stets etwas größer als `Resolution` sein.

`MinThickness` legt die Dicke des Objekts auf der initialen Oberfläche in Normalenrichtung fest. Sie wird in der gleichen Einheit gesetzt, die in `ObjectModel3D` verwendet wird. Punkte, die laut `NormalDirection` (sowie `GenParamName`=*'angle_threshold'*) auf der Innenseite der Objektoberfläche liegen, werden somit nur als innerhalb des Objekts betrachtet, falls ihre Distanz zur initialen Oberfläche den Wert `MinThickness` nicht überschreitet. Dies kann zu Hohlräumen im Objekt führen. `MinThickness` muss stets größer als `SurfaceTolerance` sein. Der Operator versucht, eine geschlossene Oberfläche zu erstellen. Für Punktwolken, welche das Objekt von verschiedenen Seiten repräsentieren, wird `MinThickness` am besten als die Dicke des schmalsten Bereichs festgelegt. Sollte die Eingabe-Punktwolke das Objekt nur von einer Seite repräsentieren, will man üblicherweise den

umschließenden Quader als Begrenzung des unbekannten Teils. Damit die Ausdehnung des Objekts nicht bereits durch die gesetzte Dicke begrenzt wird, muss `MinThickness` groß genug gesetzt werden. Aufgrund der fehlenden Beobachtung der Objektrückseite wird die Rekonstruktion dieses Teils wahrscheinlich falsch sein. Falls verschiedene Objekte nur von einer Seite betrachtet werden, sollte `MinThickness` kleiner sein, damit die rekonstruierten Objekte schmal genug bleiben, um nicht zu einer Oberfläche zusammengefasst zu werden. Zu kleine Werte können Löcher oder Doppelwände in den vereinigten Punktwolken ergeben. Hingegen können zu große Werte zu einer deformierten Punktwolke oder gar zum Aufblasen der Punktwolke Richtung Oberfläche führen (wird die Oberfläche über den umschließenden Quader hinaus aufgeblasen, so wird sie an der entsprechenden Ebene abgeschnitten und es werden wie oben beschrieben keine Punkte für diese Ebene zurückgegeben).



(1) (2)

Schematische Darstellung der Parameter `SurfaceTolerance`, `MinThickness` und der Größe *'distance_in_front'* anhand einer Beispiel-Oberfläche (_.): `o` sind Punkte außerhalb, `s` Punkte der Oberfläche, `i` Punkte sicher im Inneren des Objekts `c` Punkte die für die Bestimmung der Oberfläche auch miteinbezogen werden. (1): *'distance_in_front'* kleiner als `SurfaceTolerance` (2): *'distance_in_front'* größer als `SurfaceTolerance`.

`Smoothing` bestimmt, wie eine kleine Gesamtabweichung der Distanzfunktion gegenüber der Datentreue gewichtet wird. Damit kann also die 'Unruhe' der rekonstruierten Oberfläche durch den Glättungsparameter `Smoothing` gesteuert werden. Ein geeigneter Wert für `Smoothing`, um eine passende und visuell ansprechende Oberfläche zu erhalten, muss durch Ausprobieren gefunden werden. Ein zu kleiner Wert führt dazu, dass viele Ausreißer miteinbezogen werden, selbst wenn die Oberfläche dadurch große Sprünge aufweist. Zu große Werte hingegen führen zu Verlust an Wiedergabetreue gegenüber den Eingabe-Punktwolken (wie der Algorithmus die Distanz zu den Eingabe-Punktwolken sieht, hängt stark von `SurfaceTolerance` und `MinThickness` ab). `Smoothing` muss eventuell adaptiert werden, wenn `Resolution` geändert wird.

Mit `GenParamName` und `GenParamValue` können die folgenden zusätzlichen Parameternamen und -werte eingestellt werden:

'distance_in_front' Punkte, die gemäß `NormalDirection` (sowie `GenParamName`=*'angle_threshold'*) auf der Außenseite der Objektoberfläche liegen, werden nur dann als Teil des Objekts betrachtet, wenn ihre Distanz zur initialen Oberfläche nicht größer ist als *'distance_in_front'*. Dies ist das Pendant für die Außenseite zu `MinThickness` der Innenseite, mit der Ausnahme, dass *'distance_in_front'* auch kleiner als `SurfaceTolerance` sein darf. Falls *'distance_in_front'* kleiner ist als `SurfaceTolerance`, bestimmt dies die Oberflächendicke des Objekts zur Außenseite hin. Dieser Parameter ist nützlich, falls Löcher in der Oberfläche entlang eines Sprunges gefüllt werden sollen (z.B. entlang der Blickrichtung des Sensors). In diesem Fall kann *'distance_in_front'* auf einen kleinen Wert gesetzt werden, um falsche Initialisierungen zu vermeiden. *'distance_in_front'* wird in der gleichen Einheit gesetzt, die in `ObjectModel3D` verwendet wird. Sie sollte stets etwas größer als `Resolution` sein, um Diskretisierungseffekte zu vermeiden. Der Defaultwert für *'distance_in_front'* ist ein Wert größer als die Raumdiagonale des umschließenden Quaders, wodurch alle Punkte außerhalb des Objekts berücksichtigt werden.

Wertevorschläge: 0.001, 0.1, 1, 10.

Standardwert: größer als die Raumdiagonale des umschließenden Quaders.

Restriktion: 'distance_in_front' > 0.

'angle_threshold' legt den Winkel des Kegels um die Oberflächennormalen fest und wird in [rad] gesetzt. Bei der Bestimmung der Distanzinformation für die Datentreue werden nur jene Punkte berücksichtigt, die innerhalb

eines solchen Kegels, mit Spitze auf dem nächsten Oberflächenpunkt, liegen. Wenn Distanzen zu Dreiecken bestimmt werden, kann *'angle_threshold'* auf *0.0* gesetzt werden, wodurch ausschließlich die Volumen über den Dreiecksflächen (also gerade Prismen) berücksichtigt werden. Werden Punktnormale verwendet und damit Distanzen zu Normalen bestimmt, muss *'angle_threshold'* auf einen größeren Wert gesetzt werden.

Falls Ausreißer das Resultat stören, kann ein Reduzieren des Wertes *'angle_threshold'* helfen. Falls Löcher entlang eines Sprunges in der Oberfläche gefüllt werden sollen (z.B. entlang der Blickrichtung des Sensors), kann ein Vergrößern des Wertes für *'angle_threshold'* helfen.

Wertevorschläge: 'rad(0.0)', *'rad(10.0)'*, *'rad(30.0)'*.

Standardwert: 'rad(10.0)'.

Restriktion: 'angle_threshold' >= 0

'point_meshing' bestimmt, ob die Ausgabepunkte mit dem Algorithmus 'marching tetrahedra' trianguliert werden sollen, Dazu muss *'point_meshing'* auf *'isosurface'* gesetzt werden. Wird eine Vermaschung der Isofläche aktiviert, sind auch bei gleicher `Resolution` mehr Stützpunkte in `ObjectModel3DFusion`.

Wertevorschläge: 'none', *'isosurface'*.

Standardwert: 'isosurface'.

Fusions-Algorithmus

Der Algorithmus produziert eine wasserdichte, geschlossene Oberfläche (welche eventuell am umschließenden Quader `BoundingBox` abgeschnitten ist). Ziel ist es, eine möglichst glatte und trotzdem formtreue Oberfläche zu erhalten. Dazu wird der umschließende Quader gesampelt und jedem Sample-Punkt ein Distanzwert zu einer sogenannten Isofläche (bestehend aus Punkten mit Distanzwert 0) zugewiesen. Die endgültigen Distanzwerte (und somit die Isofläche) erhält man durch Minimierung einer Fehlerfunktion, basierend sowohl auf der Datentreue zur Eingabe-Punktwolke als auch der totalen Variation ('Sprunghaftigkeit') der Distanzfunktion. Dadurch erreicht man eine Fusion der Eingabe-Punktwolken (siehe das Paper unter Referenzen).

Die Berechnung der Isofläche kann durch die Parameter des Operators beeinflusst werden. Die Auflösung, d.h. die Distanz zwischen den Stützstellen des umschließenden Quaders (in Richtung der Koordinatenachsen), kann durch den Parameter `Resolution` bestimmt werden.

Datentreue zu den Eingabe-Punktwolken wird erfasst als die vorzeichenbehafteten Distanzen der Abtastpunkte, d.h. Stützstellen des umschließende Quaders, zu ihren nächsten Nachbarn (Punkte oder Dreiecke) der Eingabe-Punktwolke. Ob ein Abtastpunkt als innerhalb oder außerhalb des Objektes liegend betrachtet wird (Vorzeichen der Distanz), wird bestimmt durch die Normale des nächsten Nachbarn auf der initialen Oberfläche und der gesetzten Richtung `NormalDirection`. Um zu bestimmen, ob ein Abtastpunkt in Bezug auf die Eingabe-Punktwolke sicher innerhalb oder außerhalb eines Objektes liegt, wird die Distanz zum nächsten Nachbarn auf der initialen Oberfläche bestimmt. Ein Abtastpunkt der Innenseite wird als sicher im Objekt liegend betrachtet, wenn die Distanz größer als `SurfaceTolerance`, aber kleiner als `MinThickness` ist, während ein Abtastpunkt der Außenseite als außerhalb gilt, wenn die Distanz größer *'distance_in_front'* ist.

Datentreue zu den Eingabe-Punktwolken wird nur berücksichtigt für Abtastpunkte, die maximal `MinThickness` innerhalb oder `GenParamName` *'distance_in_front'* außerhalb der initialen Oberfläche liegen.

Des Weiteren wird die Datentreue für einen gegebenen Abtastpunkt nicht gepflegt, falls er nicht innerhalb eines Kegels mit Winkel `GenParamName` *'angle_threshold'* liegt. Also dann nicht, wenn die Verbindungslinie des Abtastpunktes zu seinem nächsten Nachbarn auf der initialen Oberfläche und die Oberflächennormale des nächsten Nachbarn einen Winkel größer als `GenParamName` *'angle_threshold'* bilden. Während die Distanz zu nächsten Nachbardreiecken oft überzeugendere Resultate liefert, können die Distanzen zu den nächsten Punkten viel schneller berechnet werden.

Die anschließende Optimierung der Distanzwerte ist die selbe wie in `reconstruct_surface_stereo` mit `Method`=*'surface_fusion'*.

Der Parameter `Smoothing` reguliert die 'Sprunghaftigkeit' der Distanzfunktion durch Gewichten der beiden Terme in der Fehlerfunktion: Datentreue zur Eingabe-Punktwolke einerseits, Gesamtabweichung der Distanzfunktion andererseits. Der geeignete Wert für `Smoothing`, um eine passende und visuell ansprechende Oberfläche zu erhalten, muss durch Ausprobieren gefunden werden.

Die 3D Punkte des in `ObjectModel3DFusion` zurückgegebenen Objektmodells stammen aus der Isofläche, auf der die Distanzfunktion Null ist. Die Normalen werden aus dem Gradienten der Distanzfunktion berechnet. Die dadurch erhaltene Punktwolke kann auch vermascht werden mittels Algorithmus 'marching tetrahedra' durch Setzen des Parameters `GenParamName` *'point_meshing'* auf `GenParamValue` *'isosurface'*.

Troubleshooting

Es wird empfohlen, sich an den oben stehenden Ablauf zu halten. Wenn die Rekonstruktionsergebnisse nicht zufriedenstellend sind, können die folgenden Verbesserungsmaßnahmen versucht werden:

Qualität der Eingabe-Punktwolken Die Eingabe-Punktwolken sollten die gesamte Objektoberfläche repräsentieren. Falls Punktnormalen benutzt werden, sollten die Punkte auf der ganzen Oberfläche dicht liegen und nicht nur entlang der Objektkanten. Insbesondere mit CAD-Daten sollte die Triangulierung verwendet werden.

Verwendete Attribute Das Verwenden von Dreiecken statt Punktnormalen führt typischerweise zu besseren Ergebnissen. Sollten beide Attribute vorhanden sein, werden, aufgrund der Geschwindigkeitsvorteile, als Standardeinstellung Punktnormalen verwendet. Bei Bevorzugung von Dreiecken verwenden Sie bitte `copy_object_model_3d`, damit nur Informationen zu Punkten und Dreiecken vorhanden sind.

Ausreißer Falls Ausreißer des Eingabe-Modells die ausgegebene Oberfläche selbst für große Werte von `Smoothing` stören, kann man versuchen, `GenParamName` *'angle_threshold'* zu reduzieren. Falls gewünscht, können Ausreißer des Eingabe-Modells auch entfernt werden, z.B. mittels `connection_object_model_3d`. In kleinerem Ausmaß kann auch das Modifizieren von `GenParamName` *'distance_in_front'* helfen, gewisse Ausreißer zu minimieren.

Löcher füllen Falls Löcher selbst für große Werte von `Smoothing` nicht gefüllt werden (z.B. ein Sprung der Oberfläche in Blickrichtung des Sensors), kann man versuchen, `GenParamName` *'distance_in_front'* zu reduzieren. Vergrößern von `GenParamName` *'angle_threshold'* kann dem Algorithmus helfen, die Löcher zu schließen. Beachten Sie, dass `triangulate_object_model_3d` beim Triangulieren Löcher von Sensordaten, welche eine 2D Zuordnung enthalten, füllen kann.

Leerer Rückgabewert Ist die Rückgabe leer, kann man versuchen, `Smoothing` zu reduzieren. Sollte die Rückgabe selbst für sehr kleine Werte von `Smoothing` leer bleiben, empfehlen wir zu überprüfen, ob ob `MinThickness` zu groß ist und ob die `NormalDirection` richtig gesetzt ist.

Rechenzeit

Um die Rechenzeit zu verkürzen, empfehlen wir folgende Punkte zu beachten:

Ausdehnung des umschließenden Quaders Der umschließende Quader sollte möglichst eng um die zu fusionierenden Punktwolken liegen. Ansonsten wird die Rechenzeit erheblich erhöht, ohne irgendeinen Vorteil zu generieren.

Auflösung Ein höherer Wert für die Auflösung `Resolution` beschleunigt die Fusion merklich.

Verwendete Attribute Das Verwenden von Punktnormalen anstelle von Dreiecken beschleunigt die Ausführung. Sollten beide Attribute vorhanden sein, werden als Standardeinstellung Punktnormalen verwendet.

Dichte der Eingabe-Punktewolken Die Eingabe-Punktwolken können ausgedünnt werden ,entweder mittels `sample_object_model_3d` (falls Punktnormalen verwendet werden) oder mittels `simplify_object_model_3d` mit *'true'* für den Parameter `GenParamName` *'avoid_triangle_flips'* (bei Verwendung von Dreiecken).

Oberflächendistanz Unnötig große Werte für `MinThickness` und `GenParamName` *'distance_in_front'* können die Vorbereitung und die Distanzberechnung verlangsamen.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle eines 3D-Objektmodells.

▷ **`BoundingBox`** (input_control) . . . . . number-array ⇝ *real* / integer
Zwei gegenüberliegende Ecken des umschließenden Quaders.

▷ **`Resolution`** (input_control) . . . . . number ⇝ *real* / integer
Verwendete Auflösung innerhalb des umschließenden Quaders.
Defaultwert: 1.0
Wertevorschläge: `Resolution` ∈ {1.0, 1.1, 1.5, 10.0, 100.0}

▷ **`SurfaceTolerance`** (input_control) . . . . . number ⇝ *real* / integer
Distanz des erwarteten Rauschens zur Oberfläche.
Defaultwert: 1.0
Wertevorschläge: `SurfaceTolerance` ∈ {1.0, 1.1, 1.5, 10.0, 100.0}

▷ **`MinThickness`** (input_control) . . . . . number ⇝ *real* / integer
Minimale Dicke des Objekts in Richtung Normale zur Oberfläche.
Defaultwert: 1.0
Wertevorschläge: `MinThickness` ∈ {1.0, 1.1, 1.5, 10.0, 100.0}

▷ **Smoothing** (input_control) ..... number ⇝ *real* / integer
Gewichtungsfaktor zur Datentreue.
Defaultwert: 1.0
Wertevorschläge: Smoothing ∈ {1.0, 1.1, 1.5, 10.0, 100.0}

▷ **NormalDirection** (input_control) ..... string(-array) ⇝ *string*
Richtung der Normalen im Eingabe-Modell.
Defaultwert: 'inwards'
Werteliste: NormalDirection ∈ {'inwards', 'outwards'}

▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
Name des generischen Parameters.
Defaultwert: []
Werteliste: GenParamName ∈ {'point_meshing', 'angle_threshold', 'distance_in_front'}

▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *string* / real / integer
Wert des generischen Parameters.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'isosurface', 'none', 0.0, 0.1, 0.175, 0.524}

▷ **ObjectModel3DFusion** (output_control) ..... object_model_3d ⇝ *handle*
Handle des fusionierten 3D-Objektmodells.

Ergebnis

fuse_object_model_3d liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

read_object_model_3d, register_object_model_3d_pair, register_object_model_3d_global, surface_normals_object_model_3d, triangulate_object_model_3d, simplify_object_model_3d, get_object_model_3d_params

Nachfolger

write_object_model_3d, create_surface_model

Siehe auch

reconstruct_surface_stereo

Literatur

C. Zach, T. Pock, and H. Bischof: „A globally optimal algorithm for robust TV-L1 range image integration." Proceedings of IEEE International Conference on Computer Vision (ICCV 2007).

Modul

3D Metrology

intersect_plane_object_model_3d (: : ObjectModel3D, Plane : ObjectModel3DIntersection)

Schneiden eines 3D-Objektmodells mit einer Ebene.

intersect_plane_object_model_3d schneidet ein 3D-Objektmodell mit der Ebene, die der x-y Ebene der Pose entspricht, die im Parameter Plane übergeben wird. Die z-Achse der Pose entspricht der Normalen der Ebene. Jedes Dreieck im Modell, das die Ebene schneidet, erzeugt im Ausgabemodell zwei 3D Punkte und eine Linie zwischen diesen zwei Punkten. Diese Linien werden in ObjectModel3DIntersection zurückgegeben. Die sich ergebenden Linien sind koplanar.

Das Ergebnis kann mit disp_object_model_3d angezeigt werden und die Ergebnisse können mit get_object_model_3d_params und dem Parameter *'lines'* abgefragt werden.

Parameter

- ▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Objektmodells.
- ▷ **Plane** (input_control) . . . . . pose(-array) ⇝ *real* / integer
  Pose der Ebene.
  Defaultwert: [0,0,0,0,0,0,0]
- ▷ **ObjectModel3DIntersection** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Objektmodells mit allen Schnittkanten der Oberfläche mit der Ebene.

Beispiel

```
gen_object_model_3d_from_points (rand(20)-0.5, rand(20)-0.5,\
                                 rand(20)-0.5, ObjectModel3D)
convex_hull_object_model_3d (ObjectModel3D, ObjectModel3DConvexHull)
intersect_plane_object_model_3d (ObjectModel3DConvexHull, [0,0,0,0,0,0,0], \
                                 ObjectModel3DIntersection)
dev_get_window (WindowHandle)
visualize_object_model_3d (WindowHandle, [ObjectModel3DIntersection, \
                           ObjectModel3DConvexHull], [], [], \
                           ['alpha_1'], [0.5], [], [], [], PoseOut)
```

Ergebnis

`intersect_plane_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `select_points_object_model_3d`

Nachfolger

`connection_object_model_3d`

Siehe auch

`reduce_object_model_3d_by_view`

Modul

3D Metrology

object_model_3d_to_xyz (: X, Y, Z : ObjectModel3D, Type, CamParam, Pose :)

Transformieren eines 3D-Objektmodells in Bilder.

`object_model_3d_to_xyz` transformiert die Punkte des 3D-Objektmodells `ObjectModel3D` in die drei Bilder `X`, `Y` und `Z`.

Es gibt drei mögliche Transformationsmethoden, die über den Parameter `Type` ausgewählt werden. Zu beachten ist, dass nur im Modus *'cartesian_faces'* mehr als ein 3D-Objektmodell in `ObjectModel3D` übergeben werden kann. Alle anderen Modi verlangen genau ein 3D-Objektmodell.

'cartesian': Die 3D-Punkte des Modells werden zunächst mit der übergebenen `Pose` transformiert. Anschließend werden sie mittels der in `CamParam` übergebenen Kameraparameter in ein Bild projiziert.

Die internen Kameraparameter `CamParam` beschreiben die Art der Projektion (siehe Kalibrierung). Die `Pose` wird in der Form ${}^{ccs}\mathbf{P}_{mcs}$ erwartet. Dabei steht *ccs* für das Kamerakoordinatensystem und *mcs* für das Modellkoordinatensystem (ein 3D Weltkoordinatensystem), siehe auch Transformationen / Posen und

„Solution Guide III-C - 3D Vision“. Folglich beschreibt die Pose die Position und Orientierung des Modellkoordinatensystems zum Kamerakoordinatensystem.

Die X-, Y- und Z-Koordinaten der transformierten Punkte werden jeweils in das entsprechende Ausgabebild an die projizierte Stelle geschrieben. Werden mehrere Punkte auf die selbe Bildkoordinate projiziert, so wird nur der Punkt mit dem kleinsten Z-Wert in die Bilder geschrieben. Die Größe der Bilder wird durch die Kameraparameter festgelegt.

Die zurückgegebenen Bilder zeigen das Objekt so wie es von der Kamera unter der entsprechenden Pose gesehen werden würde.

'cartesian_faces': Diese Transformationsmethode ist nur verwendbar, wenn die 3D-Objektmodelle Flächen (Dreiecke oder Polygone) enthalten. Andernfalls wird das 3D-Objektmodell ohne Flächen unberücksichtigt. Wenn eines der 3D-Objektmodelle Polygone enthält, werden diese intern zunächst in Dreiecke umgewandelt. Diese Umwandlung kann schon vorher durchgeführt werden, um diesen Operator zu beschleunigen. Dafür kann `read_object_model_3d` mit `GenParamName` *'convert_to_triangles'* auf *'true'* gesetzt verwendet werden, um alle Polygone in Dreiecke zu konvertieren. Alternativ kann `triangulate_object_model_3d` zur Erstellung der Oberflächendreiecke aufgerufen werden.

Die Flächen der Modelle werden zunächst mit der übergebenen `Pose` transformiert. Anschließend werden sie mittels der in `CamParam` übergebenen Kameraparameter in ein Bild projiziert, wobei die 3D-Information (X-, Y- und Z-Koordinaten) in den Pixeln weiterhin enthalten bleibt. Mehr Details zu `CamParam` und `Pose` finden sich auch im Abschnitt *'cartesian'*. Werden mehrere Flächen auf dieselbe Bildkoordinate projiziert, so wird nur der Wert mit dem kleinsten Z-Wert in die Bilder geschrieben. Die Größe der Bilder wird durch die Kameraparameter festgelegt.

Die zurückgegebenen Bilder zeigen die Objekte so, wie sie von der Kamera unter der entsprechenden Pose gesehen werden würden.

Sind OpenGL 2.1, GLSL 1.2 und die OpenGL Extensions GL_EXT_framebuffer_object und GL_EXT_framebuffer_blit verfügbar, ist die Verarbeitungsgeschwindigkeit höher.

Dieser `Type` kann zur Erzeugung von Modellen mit 2D-Mapping verwendet werden, indem mit `xyz_to_object_model_3d` wieder ein Modell aus den zurückgegebenen Bildern erstellt wird. Die Verwendung eines 2D-Mappings ist oft vorteilhaft auf Grund besserer Robustheit und höherer Verarbeitungsgeschwindigkeit, z. B. für die Operatoren `sample_object_model_3d` und `surface_normals_object_model_3d`, sowie in der Vorbearbeitung eines 3D-Objektmodells für das oberflächenbasierte 3D-Matching (Glättung, Entfernung der Ausreißer, und Reduzieren der Domäne).

'from_xyz_map': Diese Transformationsmethode ist nur möglich, wenn das 3D-Objektmodell mit dem Operator `xyz_to_object_model_3d` erstellt wurde. Hier werden die 3D-Punkte an die Stelle der Bilder geschrieben von der sie ursprünglich kamen. Dabei wird die in `xyz_to_object_model_3d` erstellte Abbildung der Punkte in Bildkoordinaten verwendet.

Die Parameter `CamParam` und `Pose` werden ignoriert. Die Größe der Bilder ist gleich der Bildgröße der ursprünglich an `xyz_to_object_model_3d` übergebenen Bilder und kann über `get_object_model_3d_params` mit *'mapping_size'* abgefragt werden.

Diese Transformationsmethode ist schneller als *'cartesian'*. Sie ist z.B. nützlich um das Ergebnis einer mit `segment_object_model_3d` durchgeführten Segmentierung anzuzeigen.

Achtung

Kameras mit hyperzentrischen Objektiven werden nicht unterstützt. Falls `CamParam` Verzeichnungskoeffizienten ungleich null enthält ist bei der Projektion großer Dreiecke oder Polygone darauf zu achten, dass die Verzeichnung nur auf die Punkte des Objektmodells angewandt werden. In der Projektion werden diese wiederum mit geraden Linien verbunden. Für eine gute Approximation der verzeichneten Linien sollte eine Triangulierung mit ausreichend kleinen Dreiecken verwendet werden.

Parameter

- ▷ **X** (output_object) ..... singlechannelimage ⇝ *object :* real
  Bild mit den X-Koordinaten der 3D-Punkte.
- ▷ **Y** (output_object) ..... singlechannelimage ⇝ *object :* real
  Bild mit den Y-Koordinaten der 3D-Punkte.
- ▷ **Z** (output_object) ..... singlechannelimage ⇝ *object :* real
  Bild mit den Z-Koordinaten der 3D-Punkte.
- ▷ **ObjectModel3D** (input_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle des zu transformierenden 3D-Objektmodells.

▷ **Type** (input_control) . . . . . string ⇝ *string*
Transformationsmethode.
Defaultwert: 'cartesian'
Werteliste: Type ∈ {'cartesian', 'cartesian_faces', 'from_xyz_map'}

▷ **CamParam** (input_control) . . . . . campar ⇝ *real* / integer / string
Kameraparameter.

▷ **Pose** (input_control) . . . . . pose ⇝ *real* / integer
Pose des 3D-Objektmodells.
Parameteranzahl: Pose == 0 || Pose == 7 || Pose == 12

Ergebnis

object_model_3d_to_xyz liefert den Wert TRUE wenn alle Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d, triangulate_object_model_3d

Alternativen

project_object_model_3d

Siehe auch

xyz_to_object_model_3d, get_object_model_3d_params

Modul

3D Metrology

prepare_object_model_3d (: : ObjectModel3D, Purpose, OverwriteData, GenParamName, GenParamValue :)

Vorbereiten eines 3D-Objektmodells für nachfolgende Operationen.

Der Operator prepare_object_model_3d bereitet ein 3D-Objektmodell ObjectModel3D für eine nachfolgende Operation, die in Purpose festgelegt ist, vor. Der Operator berechnet Werte, die für die Operation erforderlich sind, und speichert diese in ObjectModel3D, um die Geschwindigkeit der nachfolgenden Operationen zu erhöhen. Es ist nicht notwendig den Operator prepare_object_model_3d aufzurufen. Wenn das 3D-Objektmodell mehrere Male von derselben Operation benutzt wird, kann es allerdings die Geschwindigkeit der Operation erhöhen.

Die folgenden Werte für den Parameter Purpose sind möglich:

'shape_based_matching_3d': Das 3D-Objektmodell wird für die Nutzung im Operator create_shape_model_3d vorbereitet. In diesem Fall können keine generischen Parameter gesetzt werden.

'segmentation': Das 3D-Objektmodell wird für die Nutzung im Operator segment_object_model_3d vorbereitet. Um die Vorbereitung durchführen zu können muss das 3D-Objektmodell das Attribut mit den Oberflächendreiecken und das Attribut mit den Punktkoordinaten beinhalten.

Wenn das 3D-Objektmodell kein Attribut mit den Oberflächendreiecken hat, wird eine einfache Dreiecksvermaschung durchgeführt. Hierfür muss das 3D-Objektmodell das Attribut mit den Punktkoordinaten und das Attribut mit der Zuordnung der 3D Punkte zu den Bildkoordinaten beinhalten. Es wird nur zwischen Punkten trianguliert, die von benachbarten Pixeln stammen. Zusätzlich können Löcher in der Bildregion mit einer Delaunay-Triangulierung gefüllt werden (siehe *'max_area_holes'* unten). Nur Löcher, die vollständig von der Bildregion eingeschlossen sind, werden geschlossen.

'distance_computation': Das 3D-Objektmodell wird für die Verwendung als Zielmodell in distance_object_model_3d vorbereitet.

'gen_xyz_mapping': Für ein 3D-Objektmodell, welches eine geordnete Punktwolke aufweist, wird ein XYZ-Mapping berechnet. Dabei werden jedem 3D-Punkt Bildkoordinaten zugeordnet. Hierfür muss entweder der generische Parameter *'xyz_map_width'* oder *'xyz_map_height'* gesetzt werden, um festzulegen ob die 3D-Punkte zeilen- oder spaltenweise vorliegen und um die Bildgröße zu definieren (siehe *'xyz_map_width'* und *'xyz_map_height'* unten).

Falls ein 2D-Mapping vorhanden ist, wird seine Verwendung auf Grund besserer Robustheit und höherer Verarbeitungsgeschwindigkeit empfohlen. Vor allem für die Operatoren `sample_object_model_3d` und `surface_normals_object_model_3d`, sowie in der Vorbearbeitung eines 3D-Objektmodells für das oberflächenbasierte 3D-Matching (Glättung, Entfernung der Ausreißer, und Reduzieren der Domäne) ist daher die Verwendung des 2D-Mappings vorteilhaft.

Der Parameter `OverwriteData` legt fest, ob vorhandene Daten eines bereits vorbereiteten 3D-Objektmodells gelöscht werden sollen. Falls `OverwriteData` auf *'true'* gesetzt ist, werden die vorbereiteten Daten, die durch den Parameter `Purpose` festgelegt sind, überschrieben. Falls `OverwriteData` auf *'false'* gesetzt ist, werden die bereits vorbereiteten Daten nicht überschrieben.

Der Parameter `OverwriteData` kann benutzt werden, wenn andere generische Parameter `GenParamName` und `GenParamValue` gesetzt werden. Der Parameter `OverwriteData` hat keinen Einfluss, wenn der Parameter `Purpose` auf *'shape_based_matching_3d'* gesetzt ist, da in diesem Fall keine generischen Parameter gesetzt werden können.

Die generischen Parameter können optional genutzt werden um die Vorbereitung zu beeinflussen. Falls gewünscht, können die generischen Parameter und deren Werte in `GenParamName` und `GenParamValue` übergeben werden. Die folgenden Werte für `GenParamName` und die dazugehörigen Werte für `GenParamValue` sind möglich:

'max_area_holes': Dieser Parameter ist nur gültig, falls `Purpose` auf *'segmentation'* gesetzt ist. Der Parameter legt fest, welche Fläche die Löcher in den Punktkoordinaten haben dürfen, damit diese mittels einer Delaunay-Triangulierung geschlossen werden. Nur Löcher, die vollständig von der Bildregion eingeschlossen sind, werden geschlossen. Falls *'max_area_holes'* auf *0* gesetzt ist, werden keine Löcher vermascht. Falls *'max_area_holes'* größer oder gleich *1* Pixel gesetzt ist, werden die Löcher mit einer Fläche kleiner oder gleich *'max_area_holes'* mittels einer Vermaschung geschlossen.

Wertevorschläge: 1, *10*, *100*.

Standardwert: 10.

'distance_to': Dieser Parameter ist nur gültig, falls `Purpose` auf *'distance_computation'* gesetzt ist. Dieser Parameter bestimmt die Zieldaten zu denen der Abstand berechnet werden soll. Der Parameter wird in der Dokumentation zu `distance_object_model_3d` genauer beschrieben.

Werteliste: 'auto', *'triangles'*, *'points'*, *'primitive'*.

Standardwert: 'auto'.

'method': Dieser Parameter ist nur gültig, falls `Purpose` auf *'distance_computation'* gesetzt ist. Dieser Parameter bestimmt die Methode mit welcher der Abstand berechnet werden soll. Der Parameter wird in der Dokumentation zu `distance_object_model_3d` genauer beschrieben.

Werteliste: 'auto', *'kd-tree'*, *'voxel'*, *'linear'*.

Standardwert: 'auto'.

'max_distance': Dieser Parameter ist nur gültig, falls `Purpose` auf *'distance_computation'* gesetzt ist. Dieser Parameter bestimmt den maximal interessanten Abstand. Wird der Parameter auf *0* gesetzt, so wird kein maximaler Abstand verwendet. Der Parameter wird in der Dokumentation zu `distance_object_model_3d` genauer beschrieben.

Wertevorschläge: 0, *0.1*, *1*, *10*.

Standardwert: 0.

'sampling_dist_rel': Dieser Parameter ist nur gültig, falls `Purpose` auf *'distance_computation'* gesetzt ist. Dieser Parameter setzt die relative Abtastdistanz falls der Abstand zu Dreiecken mit der Methode *'voxel'* berechnet wird. Der Parameter wird in der Dokumentation zu `distance_object_model_3d` genauer beschrieben.

Wertevorschläge: 0.03, *0.01*.

Standardwert: 0.03.

'sampling_dist_abs': Dieser Parameter ist nur gültig, falls `Purpose` auf *'distance_computation'* gesetzt ist. Dieser Parameter setzt die absolute Abtastdistanz falls der Abstand zu Dreiecken mit der Methode *'voxel'* berechnet wird. Der Parameter wird in der Dokumentation zu `distance_object_model_3d` genauer beschrieben.

Wertevorschläge: *1*, *5*, *10*.

Standardwert: None.

'xyz_map_width': Dieser Parameter ist nur gültig, falls `Purpose` auf *'gen_xyz_mapping'* gesetzt ist. Dieser Parameter legt fest, dass die Punktwolke zeilenweise geordnet ist. Der übergebene Wert wird als Breite des Bildes angenommen. Die Höhe des Bildes wird automatisch ermittelt. Nur einer der beiden Parameter *'xyz_map_width'* und *'xyz_map_height'* kann gesetzt werden.

Standardwert: None.

'xyz_map_height': Dieser Parameter ist nur gültig, falls `Purpose` auf *'gen_xyz_mapping'* gesetzt ist. Dieser Parameter legt fest, dass die Punktwolke spaltenweise geordnet ist. Der übergebene Wert wird als Höhe des Bildes angenommen. Die Breite des Bildes wird automatisch ermittelt. Nur einer der beiden Parameter *'xyz_map_width'* und *'xyz_map_height'* kann gesetzt werden.

Standardwert: None.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells.

▷ **`Purpose`** (input_control) . . . . . string ⇝ *string*
Zweck des 3D-Objektmodells.
Defaultwert: 'shape_based_matching_3d'
Wertevorschläge: `Purpose` ∈ {'shape_based_matching_3d', 'segmentation', 'distance_computation', 'gen_xyz_mapping'}

▷ **`OverwriteData`** (input_control) . . . . . string ⇝ *string*
Legt fest, ob schon vorhandene Daten überschrieben werden.
Defaultwert: 'true'
Werteliste: `OverwriteData` ∈ {'true', 'false'}

▷ **`GenParamName`** (input_control) . . . . . attribute.name-array ⇝ *string* / real / integer
Name des generischen Parameters.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'max_area_holes', 'distance_to', 'method', 'max_distance', 'sampling_dist_rel', 'sampling_dist_abs', 'xyz_map_width', 'xyz_map_height'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value-array ⇝ *string* / real / integer
Wert des generischen Parameters.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {0, 1, 100, 'auto', 'triangles', 'points', 'primitive', 'kd-tree', 'voxel', 'linear', 0.01, 0.03}

Beispiel

```
read_object_model_3d ('object_model_3d', 'm', [], [], ObjectModel3D, Status)
prepare_object_model_3d (ObjectModel3D, 'gen_xyz_mapping', 'true',\
                         'xyz_map_width', Width)
object_model_3d_to_xyz (X, Y, Z, ObjectModel3D, 'from_xyz_map', [], [])
```

Ergebnis

`prepare_object_model_3d` liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`create_shape_model_3d`, `create_surface_model`, `distance_object_model_3d`,

find_surface_model, fit_primitives_object_model_3d, refine_surface_model_pose, segment_object_model_3d, simplify_object_model_3d, sample_object_model_3d, surface_normals_object_model_3d

Modul

3D Metrology

project_object_model_3d (: ModelContours : ObjectModel3D, CamParam, Pose, GenParamName, GenParamValue :)

Projizieren eines 3D-Objektmodells in Bildkoordinaten.

Der Operator project_object_model_3d projiziert ein 3D-Objektmodell in das Bildkoordinatensystem und liefert die projizierten Konturen in ModelContours zurück. Dieser Operator ist vor allem für die Visualisierung von 3D-Objektmodellen hilfreich. Primitive werden nicht projiziert, sondern bei der Visualisierung ignoriert. Die Koordinaten des 3D-Objektmodells sind im Modellkoordinatensystem gegeben (*mcs*), einem 3D-Weltkoordinatensystem. Sie werden zuerst mittels der gegebenen Pose in Kamerakoordinaten (*ccs*) transformiert und dann basierend auf den internen Kameraparametern CamParam in das Bildkoordinatensystem projiziert. Dabei wird die Pose in der Form ${}^{ccs}\mathbf{P}_{mcs}$ erwartet, siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". Das heißt, die Pose beschreibt die Position und Lage des Weltkoordinatensystems im Kamerakoordinatensystem. Die internen Kameraparameter CamParam beschreiben dabei die Abbildungseigenschaften der Kamera (siehe Kalibrierung).

Es gibt einige generische Parameter, die optional angegeben werden können, um die Projektion zu beeinflussen. Falls erwünscht, können die generischen Parameter und deren Werte in GenParamName und GenParamValue übergeben werden. Die folgenden Werte für GenParamName sind möglich:

'data': Dieser Parameter gibt an, welche geometrischen Daten des 3D-Objektmodells projiziert werden sollen. Wenn für *'data' 'faces'* gewählt wird, werden die Flächen des 3D-Objektmodells projiziert. Die Flächen werden dabei in ModelContours durch ihre Randlinien repräsentiert. Wird für *'data' 'lines'* gewählt, werden die 3D-Linien des 3D-Objektmodells projiziert. Wenn für *'data' 'points'* gewählt wird, werden die Punkte des 3D-Objektmodells projiziert. Die projizierten Punkte können auf unterschiedliche Arten in ModelContours repräsentiert werden. Die Art der Punktrepräsentation kann mit dem generischen Parameter *'point_shape'* gewählt werden (siehe unten). Wird für *'data'* der Wert *'auto'* übergeben, wählt HALCON automatisch die für eine Visualisierung am besten geeigneten Geometriedaten aus.

Werteliste: 'auto', 'faces', 'points', 'lines'.

Standardwert: 'auto'.

'point_shape': Mit diesem Parameter kann gewählt werden, wie Punkte in der Ausgabekontur ModelContours repräsentiert werden. Daher wirkt sich dieser Parameter nur dann aus, wenn Punkte als zu projizierende Geometriedaten ausgewählt wurden (siehe oben). Wird *'point_shape'* auf *'circle'* gesetzt, dann werden Punkte durch Kreise repräsentiert, wird *'point_shape'* auf *'cross'* gesetzt, dann werden Punkte durch Kreuze repräsentiert. In beiden Fällen kann die Größe der Punkte (d.h. die Größe der Kreise bzw. der Kreuze) mit dem generischen Parameter *'point_size'* gewählt werden (siehe unten). Die Orientierung der Kreuze kann mit dem generischen Parameter *'point_orientation'* festgelegt werden (siehe unten).

Werteliste: 'circle', 'cross'.

Standardwert: 'circle'.

'point_size': Mit diesem Parameter kann die Größe der Punktrepräsentation in der Ausgabekontur ModelContours gewählt werden, d.h. die Größe der Kreise bzw. der Kreuze, je nach Wahl von *'point_shape'*. Dieser Parameter wirkt sich natürlich nur dann aus, wenn Punkte als zu projizierende Geometriedaten ausgewählt wurden (siehe oben). Die Größe muss in Pixeleinheiten angegeben werden. Wird für *'point_size'* der Wert *0* übergeben, dann wird jeder Punkt durch eine Kontur repräsentiert, die nur einen Punkt enthält.

Wertevorschläge: 0, 2, 4.

Standardwert: 4.

'point_orientation': Dieser Parameter legt die Orientierung der Kreuze im Bogenmaß fest. Der Parameter wirkt sich natürlich nur dann aus, wenn Punkte als zu projizierende Geometriedaten ausgewählt wurden und für *'point_shape'* der Wert *'cross'* ausgewählt wurde (siehe oben).

Wertevorschläge: 0, 0.39, 0.79.

Standardwert: 0.79.

'union_adjacent_contours': Dieser Parameter legt fest, ob nach der Projektion benachbarte Konturen verbunden werden sollen. Diese Option is äquivalent zu einem anschließenden Aufruf von `union_adjacent_contours_xld`, ist allerdings deutlich schneller.

Werteliste: 'true', 'false'.

Standardwert: 'true'.

'hidden_surface_removal': Mit diesem Parameter kann die Verdeckungsberechnung ein- bzw. ausgeschaltet werden. Wenn der Parameter *'hidden_surface_removal'* auf *'true'* gesetzt ist, werden nur die projizierten Kanten zurückgegeben, die nicht durch Flächen des 3D-Objektmodells verdeckt sind. Ist *'hidden_surface_removal'* auf *'false'* gesetzt, werden alle projizierten Kanten des 3D-Objektmodells zurückgegeben. Dieser Modus ist schneller als wenn *'hidden_surface_removal'* auf *'true'* gesetzt ist.

Ist die Systemvariable (siehe `set_system`) *'opengl_hidden_surface_removal_enable'* auf *'true'* gesetzt (Standard, sofern es verfügbar ist) und *'hidden_surface_removal'* auf *'true'* gesetzt, wird die Modellprojektion auf der Grafikkarte durchgeführt. Abhängig von der verwendeten Grafikkarte ist dies signifikant schneller als die analytische Projektion. Es ist zu beachten, dass die Ergebnisse der OpenGL Projektion leicht verschieden zu denen der analytischen Projektion sind. Insbesondere werden in diesem Modus nur die Kanten projiziert die im Bild der Kamera (`CamParam`) sichtbar sind.

Werteliste: 'true', 'false'.

Standardwert: 'true'.

'min_face_angle': Mit diesem Parameter wird der minimale Winkel angegeben, den zwei benachbarte Flächen einschließen müssen, damit die Kante zwischen ihnen projiziert wird. Wird *'min_face_angle'* auf *0.0* gesetzt, werden alle Kanten projiziert. Wird *'min_face_angle'* auf π (entspricht 180 Grad) gesetzt, wird nur die Silhouette des 3D-Objektmodells zurückgegeben. Dieser Parameter kann insbesondere dafür verwendet werden, um Kanten in gekrümmten Flächen, z.B. einer Zylinder- oder Kegeloberfläche, zu unterdrücken.

Wertevorschläge: 0.17, 0.26, 0.35, 0.52.

Standardwert: 0.52.

Achtung

Kameras mit hyperzentrischen Objektiven werden nicht unterstützt. Falls `CamParam` Verzeichnungskoeffizienten ungleich null enthält ist bei der Projektion großer Dreiecke oder Polygone darauf zu achten, dass die Verzeichnung nur auf die Punkte des Objektmodells angewandt werden. In der Projektion werden diese wiederum mit geraden Linien verbunden. Für eine gute Approximation der verzeichneten Linien sollte eine Triangulierung mit ausreichend kleinen Dreiecken verwendet werden.

Parameter

- ▷ **`ModelContours`** (output_object) ..... xld_cont(-array) ⇝ *object*
  Projizierte Modellkonturen.
- ▷ **`ObjectModel3D`** (input_control) ..... object_model_3d ⇝ *handle*
  Handle des 3D-Objektmodells.
- ▷ **`CamParam`** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **`Pose`** (input_control) ..... pose ⇝ *real* / integer
  3D-Lage des Weltkoordinatensystems in Kamerakoordinaten.
  Parameteranzahl: Pose == 7
- ▷ **`GenParamName`** (input_control) ..... string(-array) ⇝ *string*
  Name des generischen Parameters.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'true', 'false', 'hidden_surface_removal', 'min_face_angle', 'data', 'point_shape', 'point_size', 'point_orientation', 'union_adjacent_contours'}
- ▷ **`GenParamValue`** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Wert des generischen Parameters.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {0.17, 0.26, 0.35, 0.52, 'true', 'false', 'auto', 'points', 'faces', 'lines', 'circle', 'cross', 1, 2, 3, 4, 0.785398}

Ergebnis

`project_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine

Fehlerbehandlung durchgeführt. Wenn die gewählten zu projizierenden Geometriedaten nicht im 3D-Objektmodell vorhanden sind, wird die Fehlermeldung 9514 zurückgegeben.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `affine_trans_object_model_3d`, `prepare_object_model_3d`

Nachfolger

`clear_object_model_3d`

Siehe auch

`project_shape_model_3d`, `object_model_3d_to_xyz`

Modul

3D Metrology

`projective_trans_object_model_3d` (: : `ObjectModel3D`, `HomMat3D` : `ObjectModel3DProjectiveTrans`)

Anwenden einer beliebigen projektiven 3D-Transformation auf 3D-Objektmodelle.

`projective_trans_object_model_3d` wendet eine beliebige projektive 3D-Transformation auf 3D-Objektmodelle an und liefert die Handle der transformierten 3D-Objektmodelle zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in `HomMat3D` übergeben wird (siehe `projective_trans_point_3d`).

Die Transformationsmatrix kann z.B. mit Hilfe des Operators `vector_to_hom_mat3d` aufgebaut werden.

Achtung

`projective_trans_object_model_3d` transformiert nur Attribute vom Typ 3D-Punkt. Attribute vom Typ 3D-Formmodell für das formbasierte 3D-Matching, vom Typ 3D-Primitiv und vom Typ Normale werden nicht transformiert. Daher sind diese Attribute nicht im transformierten 3D-Objektmodell vorhanden. Alle anderen Attribute werden kopiert ohne sie zu verändern. Um 3D-Primitive zu transformieren, muss der Operator `rigid_trans_object_model_3d` verwendet werden.

Parameter

- ▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle der 3D-Objektmodelle.
- ▷ **`HomMat3D`** (input_control) . . . . . hom_mat3d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **`ObjectModel3DProjectiveTrans`** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle der transformierten 3D-Objektmodelle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `projective_trans_object_model_3d` den Wert TRUE. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`project_object_model_3d`, `object_model_3d_to_xyz`

Siehe auch

affine_trans_point_3d, rigid_trans_object_model_3d, affine_trans_object_model_3d

Modul

3D Metrology

register_object_model_3d_global (: : ObjectModels3D, HomMats3D, From, To, GenParamName, GenParamValue : HomMats3DOut, Scores)

Verfeinern der relativen Lagen der eingegebenen 3D-Objektmodelle zueinander, basierend auf deren Überlappung.

register_object_model_3d_global verfeinert die relative Lage der angegebenen 3D-Objektmodelle zueinander, dieser Vorgang wird auch globale Registrierung genannt. Ausgehend von den mit HomMats3D angegebenen genäherten relativen Lagen der Modelle aus ObjectModels3D zueinander werden paarweise Überlappungsbereiche gesucht. Alle Eingabemodelle mit erkannten Überlappungen werden im Rahmen einer Ausgleichsrechnung so gegeneinander verschoben, dass die Überlappung maximal ist. Dieser Vorgang wird dann iterativ wiederholt. Die Lagen der Modelle mit der höchsten erreichten Überlappung werden in HomMats3DOut relativ zu einem globalen Referenzsystem zurückgegeben. Die entsprechenden Transformationen können mit dem Operator affine_trans_object_model_3d auf die angegebenen 3D-Objektmodelle angewandt werden, um sie in ein gemeinsames Koordinatensystem zu überführen. Der Parameter Scores enthält pro 3D-Objektmodell die Anzahl an gefundenen Nachbarn mit ausreichender Überlappung. Wenn für mindestens ein 3D-Objektmodell keine Überlappung gefunden wird, wird eine Fehlerbehandlung durchgeführt.

Die genäherten relativen Lagen der Modelle müssen mithilfe der Parameter From und To konfiguriert und in HomMats3D angegeben werden. Es gibt drei mögliche Wege die relative Lage anzugeben:

Erstens kann durch das Setzen von From auf *'global'* die Interpretation des Parameters HomMats3D als ein Satz von globalen Transformationen zu einem gemeinsamen Koordinatenursprung festgelegt werden. Die Anzahl der Transformationen in HomMats3D (je 12 Werte pro Transformation) muss in diesem Fall gleich der Anzahl der 3D-Objektmodelle sein. Der Parameter To darf in diesem Fall nicht gesetzt sein. Diese Parametrisierung ist für Fälle gedacht, in denen entweder das aufgenommene Objekt oder die Kamera statisch bleibt und auf das jeweils Andere eine bekannte Transformation angewandt wird, z.B. durch einen Drehteller oder einen Roboter. In diesem Fall werden alle Nachbarschaften, die aufgrund der angegebenen Startlösung denkbar sind, für die Optimierung berücksichtigt.

Zweitens kann durch das Setzen von From auf *'previous'* die Interpretation des Parameters HomMats3D als ein Satz von relativen Verschiebungen zum jeweils vorhergehenden 3D-Objektmodell festgelegt werden. Hierfür wird eine Transformation weniger als die Anzahl der 3D-Objektmodelle benötigt, da die erste Transformation sich auf die Transformation vom ersten 3D-Objektmodell in das Referenzsystem des zweiten bezieht, usw. Der Parameter To darf in diesem Fall nicht gesetzt sein. Der Fall sollte angewandt werden, wenn die Kamera und das aufgenommene Objekt frei gegeneinander bewegt werden und die Transformation jeweils nur zur vorherigen Aufnahme ermittelt wird, z.B. mit dem Operator register_object_model_3d_pair. In diesem Fall werden alle Nachbarschaften, die aufgrund der angegebenen Startlösung denkbar sind, für die Optimierung berücksichtigt.

Drittens kann durch das Angeben von Indexpaaren in From und To jede einzelne Transformation in HomMats3D beschrieben werden. Die angegebenen Transformationen sind demnach immer die benötigten Transformationen, um das 3D-Objektmodell mit Index From in das Koordinatensystem des 3D-Objektmodells mit Index To zu verschieben. Die Indizes beziehen sich auf die Eingabeparameter von ObjectModels3D. In diesem Fall werden nur die Nachbarschaften berücksichtigt, die auch angegeben sind, somit sollten alle nötigen Überlappungen auch angegeben werden. Dies muss so geschehen, dass zwischen jedem Paar von Eingabeobjekten ein entsprechender Pfad von Transformationen vorhanden ist.

Wenn ObjectModels3D 3D-Primitive enthält, werden diese intern in Punktwolken umgewandelt und anschließend als solche behandelt.

Die numerisch maximal erreichbare Positionsgenauigkeit liegt bei etwa 0.1% der Punktwolkengröße. Die Genauigkeit hängt weiter vom Rauschen der Daten, von der Anzahl der Datenpunkte und von der Form der Punktwolken ab.

Der Vorgang der globalen Registrierung kann mit den Parametern GenParamName und GenParamValue weiter reguliert werden.

'default_parameters': Es gibt zwei Standardparametersätze, die zwischen schneller Konvergenz (*'fast'*) und genauer Überlappung (*'accurate'*) abwägen.

Werteliste: 'fast', 'accurate'.

Standardwert: 'accurate'.

'rel_sampling_distance': Die relative Abtastrate der Eingabemodelle. Der Wert wird relativ zum Durchmesser der jeweiligen 3D-Objektmodelle interpretiert. Ein höherer Wert führt zu schnelleren Ausführungszeiten, während ein niedrigerer Wert zu genaueren Ergebnissen führt.

Wertevorschläge: 0.03, 0.05, 0.07.

Standardwerte: 0.05 ('default_parameters' = 'accurate'), 0.07 ('default_parameters' = 'fast').

Restriktion: 0 < *'rel_sampling_distance'* < 1

'pose_ref_sub_sampling': Anzahl an Punkten, die für jeden zugeordneten Punkt bei der Berechnung der Überlappung ausgelassen werden. Ein hoher Wert erlaubt eine schnellere Berechnung bei etwas ungenaueren Ergebnissen.

Wertevorschläge: 1, 2, 20.

Standardwerte: 2 ('default_parameters' = 'accurate'), 10 ('default_parameters' = 'fast').

Restriktion: 'pose_ref_sub_sampling' > 0

'max_num_iterations': Anzahl der Iterationen zur Verbesserung der relativen Lagen. Je besser die initiale Schätzung der Lagen ist, desto weniger Iterationen werden benötigt.

Wertevorschläge: 1, 3, 10.

Standardwert: 3.

Parameter

- ▷ **ObjectModels3D** (input_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle mehrerer 3D-Objektmodelle.
- ▷ **HomMats3D** (input_control) ..... hom_mat3d-array ⇝ *real* / integer
  Genäherte relative Lagen der 3D-Objektmodelle.
- ▷ **From** (input_control) ..... number(-array) ⇝ *string* / integer
  Typ der genäherten relativen Lagen in `HomMats3D`.
  Defaultwert: 'global'
  Werteliste: `From` ∈ {'global', 'previous', 0, 1, 2, 3, 4}
- ▷ **To** (input_control) ..... number(-array) ⇝ *integer*
  Index des Ziel-3D-Objektmodells für die Angabe der genäherten relativen Lagen in `HomMats3D`, falls `From` Indizes enthält, andernfalls bleibt dieser Parameter leer.
  Defaultwert: []
  Werteliste: `To` ∈ {0, 1, 2, 3, 4}
- ▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
  Namen der generischen Parameter zum Steuern der globalen Registrierung.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'default_parameters', 'rel_sampling_distance', 'pose_ref_sub_sampling', 'max_num_iterations'}
- ▷ **GenParamValue** (input_control) ..... number-array ⇝ *real* / integer / string
  Werte der generischen Parameter zum Steuern der globalen Registrierung.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {0.03, 0.05, 0.07, 0.1, 0.25, 0.5, 1, 2, 5, 10, 20, 'fast', 'accurate'}
- ▷ **HomMats3DOut** (output_control) ..... hom_mat3d-array ⇝ *real* / integer
  Resultierende Transformationen.
- ▷ **Scores** (output_control) ..... number-array ⇝ *real*
  Anzahl der gefundenen Nachbarn mit ausreichender Überlappung pro eingegebenem 3D-Objektmodell.

Ergebnis

`register_object_model_3d_global` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `xyz_to_object_model_3d`, `register_object_model_3d_pair`, `gen_object_model_3d_from_points`

Nachfolger

`affine_trans_object_model_3d`, `union_object_model_3d`, `sample_object_model_3d`, `triangulate_object_model_3d`

Siehe auch

`register_object_model_3d_pair`, `find_surface_model`, `refine_surface_model_pose`

Modul

3D Metrology

register_object_model_3d_pair (: : ObjectModel3D1, ObjectModel3D2, Method, GenParamName, GenParamValue : Pose, Score)

Suchen einer Transformation zwischen zwei 3D-Objektmodellen.

`register_object_model_3d_pair` sucht eine Transformation zwischen zwei 3D-Objektmodellen. Dieser Vorgang wird auch Registrierung genannt. Die in `Pose` zurückgegebene Pose beschreibt die Transformation, die nötig ist, um `ObjectModel3D1` in das Koordinatensystem des zweiten Objekts `ObjectModel3D2` zu überführen. Im Parameter `Score` wird der Anteil der überlappenden Oberflächen zurückgegeben (als Wert zwischen *0.0* und *1.0*). Sollte keine Überlappung vorhanden sein, wird auch keine Pose zurückgegeben. Es kann als Methode im Parameter `Method` entweder eine volle Suche ohne Einschränkung der relativen Lage mit *'matching'* oder eine Verfeinerung der momentanen relativen Lage zum als gemeinsam angenommenen globalen Koordinatenursprung mit *'icp'* ausgewählt werden. Die Verfeinerung findet auch nach dem Matching statt.

Die numerisch maximal erreichbare Positionsgenauigkeit liegt bei etwa 0.1% der Punktwolkengröße. Die Genauigkeit hängt weiter vom Rauschen der Daten, von der Anzahl der Datenpunkte und von der Form der Punktwolken ab.

Mit den Parametern `GenParamName` and `GenParamValue` können das Matching und die Verfeinerung parametrisiert werden:

'default_parameters': Um eine einfache Parametrisierung zu ermöglichen stehen einige Standardparametersätze zur Verfügung. Der Parametersatz *'fast'* zielt auf eine relativ schnelle Verarbeitung. Der Parametersatz *'accurate'* zielt auf genauere Ergebnisse. Der Parametersatz *'robust'* zielt neben einer höheren Genauigkeit auch auf eine exaktere Berechnung der Bewertung in `Score`, ist aber dafür langsamer.

Werteliste: 'fast', 'accurate', 'robust'.

Standardwert: 'accurate'.

'rel_sampling_distance': Legt die relative Abtastrate fest, mit der Oberflächen der 3D-Objektmodelle abgetastet werden. Der Wert wird relativ zum Durchmesser des jeweiligen Objektes angegeben und gibt den minimalen Abstand von zwei während der Abtastung ausgewählten Punkten an. Umso höher der Wert ist, desto schneller und ungenauer ist die Berechnung der relativen Lage. Dieser Parameter kann mit Hilfe von *'rel_sampling_distance_obj1'* und *'rel_sampling_distance_obj2'* für beide Objekte unabhängig gesetzt werden.

Wertevorschläge: 0.03, 0.05, 0.07.

Standardwert: 0.05.

'key_point_fraction': Der Anteil an Punkten, die beim Matching als Referenzpunkte gewählt werden. Dieser Wert kann für das erste Objekt mit *'key_point_fraction_obj1'* und für das zweite mit *'key_point_fraction_obj2'* separat gesetzt werden. Die Erhöhung dieses Wertes führt zu einer längeren Berechnung, erlaubt aber kleinere Überlappungen zu finden.

Wertevorschläge: 0.2, 0.3, 0.4.

Standardwert: 0.3.

'pose_ref_num_steps': Die Anzahl der Iterationsschritte bei der Verfeinerung der Position.

Wertevorschläge: 5, 7, 10.

Standardwert: 5.

'pose_ref_sub_sampling': Anzahl der Punkte, die bei der Positionsverfeinerung pro ausgewertetem Punkt zu überspringen sind.

Wertevorschläge: 1, *2*, *20*.

Standardwert: 2.

'pose_ref_dist_threshold_rel': Der maximale Abstand, den zwei Flächen haben dürfen, um als Überlappungskandidaten bei der Positionsverfeinerung in Frage zu kommen. Dieser Wert wird relativ zum Durchmesser des größeren Objektes angegeben.

Wertevorschläge: 0.05, *0.1*, *0.15*.

Standardwert: 0.1.

'pose_ref_dist_threshold_abs': Der maximale Abstand, den zwei Flächen haben dürfen, um als Überlappungskandidaten bei der Positionsverfeinerung in Frage zu kommen. Dieser Wert wird absolut angegeben.

'model_invert_normals': Invertiert die Normalen des kleineren der zwei Objekte, sollten diese inkompatible Normalenausrichtungen haben.

Werteliste: 'true', *'false'*.

Standardwert: 'false'.

Parameter

▷ **ObjectModel3D1** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des ersten 3D-Objektmodells.

▷ **ObjectModel3D2** (input_control) . . . . . object_model_3d ⇝ *handle*
Handle des zweiten 3D-Objektmodells.

▷ **Method** (input_control) . . . . . string ⇝ *string*
Methode zur Registrierung.
Defaultwert: 'matching'
Werteliste: `Method` ∈ {'matching', 'icp'}

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'default_parameters', 'rel_sampling_distance', 'rel_sampling_distance_obj1', 'rel_sampling_distance_obj2', 'key_point_fraction', 'key_point_fraction_obj1', 'key_point_fraction_obj2', 'pose_ref_num_steps', 'pose_ref_sub_sampling', 'pose_ref_dist_threshold_rel', 'pose_ref_dist_threshold_abs', 'model_invert_normals'}

▷ **GenParamValue** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'fast', 'accurate', 'robust', 0.1, 0.25, 0.5, 1, 'true', 'false'}

▷ **Pose** (output_control) . . . . . pose ⇝ *real* / integer
Pose, mit der `ObjectModel3D1` in das Referenzsystem von `ObjectModel3D2` verschoben werden kann.

▷ **Score** (output_control) . . . . . number-array ⇝ *real*
Überlappung der zwei Objekte.

Beispiel

```
* Generate two boxes
gen_box_object_model_3d ([0,0,0,0,0,0,0],3,2,1, ObjectModel3D1)
gen_box_object_model_3d ([0,0,0.5,15,0,0,0],3,2,1, ObjectModel3D2)
* Match them
register_object_model_3d_pair (ObjectModel3D1, ObjectModel3D2, 'matching',\
                               [], [], Pose, Score)
```

Ergebnis

`register_object_model_3d_pair` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `gen_object_model_3d_from_points`, `xyz_to_object_model_3d`

Nachfolger

`register_object_model_3d_global`, `affine_trans_object_model_3d`, `union_object_model_3d`

Siehe auch

`register_object_model_3d_global`, `find_surface_model`

Modul

3D Metrology

render_object_model_3d (: Image : ObjectModel3D, CamParam, Pose, GenParamName, GenParamValue :)

Berechnen eines Bildes durch Rendern von 3D-Objektmodellen.

`render_object_model_3d` rendert die 3D-Objektmodelle `ObjectModel3D` und gibt das Ergebnis im Bild `Image` zurück. Die Szene wird über die Kameraparameter `CamParam` und die Posen `Pose` der Objekte definiert. `Pose` kann entweder eine Pose enthalten, die für alle Objekte gilt, oder eine individuelle Pose für jedes Objekt.

Die Ansicht der Szene ist identisch zu der von `disp_object_model_3d` erzeugten Ansicht. Die Parameter und weitere Details sind in `disp_object_model_3d` dokumentiert. Anders als in `disp_object_model_3d` sind aber die Parameter *'object_index_persistence'* und *'disp_background'* nicht einstellbar.

`render_object_model_3d` benötigt OpenGL 2.1, GLSL 1.2 und die OpenGL Extensions GL_EXT_framebuffer_object und GL_EXT_framebuffer_blit. Andernfalls wird automatisch der Parameter *'opengl_compatibility_mode_enable'* der Systemvariable (siehe `set_system`) auf *'true'* gesetzt und die Visualisierung mit geringeren Anforderungen durchgeführt. Hierfür wird OpenGL 1.1 benötigt.

Achtung

Kameras mit hyperzentrischen Objektiven werden nicht unterstützt.

Parameter

- ▷ **Image** (output_object) ..... multichannel-image ⇝ *object :* byte
  Gerenderte Szene.
- ▷ **ObjectModel3D** (input_control) ..... object_model_3d(-array) ⇝ *handle*
  Handles der 3D Objektmodelle.
- ▷ **CamParam** (input_control) ..... campar ⇝ *real* / integer / string
  Kameraparameter.
- ▷ **Pose** (input_control) ..... pose(-array) ⇝ *real* / integer
  3D Posen der Objekte im Kamerakoordinatensystem.
- ▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'alpha', 'attribute', 'color', 'colored', 'disp_lines', 'disp_pose', 'disp_normals', 'light_position', 'line_color', 'normal_color', 'quality', 'compatibility_mode_enable', 'point_size', 'color_attrib', 'color_attrib_start', 'color_attrib_end', 'red_channel_attrib', 'blue_channel_attrib', 'green_channel_attrib', 'rgb_channel_attrib_start', 'rgb_channel_attrib_end', 'lut'}
- ▷ **GenParamValue** (input_control) ..... string-array ⇝ *string* / integer / real
  Werte der generischen Parameter.
  Defaultwert: []
  Werteliste: `GenParamValue` ∈ {'true', 'false', 'coord_x', 'coord_y', 'coord_z', 'normal_x', 'normal_y', 'normal_z', 'red', 'green', 'blue', 'auto', 'faces', 'primitive', 'points', 'lines'}

Ergebnis

`render_object_model_3d` liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: mutually exclusive (läuft parallel zu anderen nicht-exklusiven Operatoren mit Ausnahme von sich selbst).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_surface_model`, `fit_primitives_object_model_3d`, `segment_object_model_3d`, `read_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`disp_obj`

Siehe auch

`disp_object_model_3d`, `project_shape_model_3d`, `object_model_3d_to_xyz`

Modul

3D Metrology

rigid_trans_object_model_3d (: : ObjectModel3D, Pose : ObjectModel3DRigidTrans)

Anwenden einer starren 3D-Transformation auf 3D-Objektmodelle.

`rigid_trans_object_model_3d` wendet starre 3D-Transformationen auf 3D-Objektmodelle an und liefert den Handle der transformierten 3D-Objektmodelle zurück. Die Transformationen werden durch eine oder mehrere Posen beschrieben, die in `Pose` übergeben werden. Die Posen werden dabei in der Form ${}^{cst}\mathbf{P}_{mcsi}$ erwartet. Dabei steht *mcsi* für das Koordinatensystem des Eingabe-3D-Objektmodells und *cst* für das Koordinatensystem des transformierten Models, z.B. das Koordinantensystem einer Szene (siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision"). Posen können mit Hilfe der Operatoren `create_pose`, `pose_invert`, etc. aufgebaut werden oder das Resultat von `get_object_model_3d_params` sein.

Der Operator unterstützt ein oder mehrere 3D-Objektmodelle mit einer Pose (N:1), ein 3D-Objektmodell mit ein oder mehreren Posen (1:N) sowie eine Pose pro übergebenem 3D-Objektmodell, die dann nur auf das jeweilige 3D-Objektmodell angewandt wird (N:N). Dabei kann N auch für 0 stehen, d.h. es kann keine Pose oder kein Objektmodell übergeben werden, in welchem Fall in `ObjectModel3DRigidTrans` ein leeres Tupel zurückgegeben wird. Dies kann z.B. verwendet werden, um die Ergebnisse von `find_surface_model` zu transformieren, ohne vorher zu prüfen ob tatsächlich mindestens ein Ergebnis gefunden wurde.

Achtung

`rigid_trans_object_model_3d` kopiert Attribute vom Typ Oberflächendreiecke, Oberflächen, Polygone, 3D Linienzüge, Zuordnung der 3D Punkte zu Bildkoordinaten und erweitertes Attribut ohne sie zu verändern.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle der 3D-Objektmodelle.

▷ **Pose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
Posen.

▷ **ObjectModel3DRigidTrans** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle der transformierten 3D-Objektmodelle.

Ergebnis

`rigid_trans_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

read_object_model_3d, xyz_to_object_model_3d, fit_primitives_object_model_3d

Nachfolger

project_object_model_3d, object_model_3d_to_xyz, get_object_model_3d_params

Siehe auch

affine_trans_point_3d, affine_trans_object_model_3d

Modul

3D Metrology

sample_object_model_3d (: : ObjectModel3D, Method, SampleDistance, GenParamName, GenParamValue : SampledObjectModel3D)

Abtasten eines 3D-Objektmodells.

sample_object_model_3d erzeugt eine abgetastete Variante des 3D-Objektmodells ObjectModel3D, welches aus Punkten besteht, die zueinander den Mindestabstand SampleDistance haben. Das erzeugte 3D-Objektmodell wird in SampledObjectModel3D zurückgegeben.

Der Gebrauch von sample_object_model_3d ist empfohlen, wenn komplexe Punktwolken zur Nachverarbeitung ausgedünnt werden oder Primitive oder triangulierte 3D-Objektmodelle in Punktwolken umgewandelt werden sollen. Wenn das 3D-Objektmodell trianguliert ist und unter bestmöglicher Beibehaltung seiner ursprünglichen Geometrie vereinfacht werden soll, sollte stattdessen simplify_object_model_3d verwendet werden.

Besteht das Eingabemodell ObjectModel3D nur aus Punkten, so stehen einige Methoden zur Verfügung die über den Parameter Method ausgewählt werden können:

'fast': Die Standardmethode *'fast'* fügt der Reihe nach alle Punkte, die bisher keinen Nachbarpunkt mit einem Abstand unter der angegebenen Schwelle haben, in das Ausgabeobjekt ein. In diesem Fall werden, soweit vorhanden, die Punktnormalen, das Mapping sowie erweiterte Punktattribute kopiert.

'fast_compute_normals' : Die Methode *'fast_compute_normals'* berechnet die Normalen von allen Punkten die wie bei der Methode *'fast'* ausgewählt wurden. Dazu muss das Eingabeobjektmodell entweder Normalen aufweisen, die kopiert werden, oder ein XYZ-Mapping aus dem die Normalen berechnet werden. Die z-Komponente der berechneten Normalenvektoren ist immer positiv. Das XYZ-Mapping wird beispielsweise von xyz_to_object_model_3d erzeugt.

'accurate': Die Methode *'accurate'* geht alle Punkte von ObjectModel3D durch und berechnet, ob innerhalb einer Kugel mit dem Radius SampleDistance um den untersuchten Punkt noch andere Punkte liegen. Wenn keine anderen Punkte in der Kugel enthalten sind, wird der ursprüngliche Punkt in SampledObjectModel3D gespeichert. Falls noch andere Punkte in der Kugel sind, wird ihr Schwerpunkt (inklusive dem ursprünglichen Punkt) berechnet und der resultierende Punkt in SampledObjectModel3D gespeichert. Dieser Vorgang wird mit den restlichen Punkten wiederholt, bis keine Punkte mehr übrig sind. Bei der Methode gehen erweiterte Attribute verloren, Normalen und XYZ-Mapping werden kopiert. Punkte, die keine oder wenig Nachbarn in der Originalpunktwolke haben, können als Ausreißer erkannt und entfernt werden, indem die Mindestanzahl an Nachbarn mit Hilfe von *'min_num_points'* in GenParamName und einer Anzahl in GenParamValue angegeben wird.

'accurate_use_normals': Die Methode *'accurate_use_normals'* benötigt bereits berechnete Normalen, und interpoliert benachbarte Punkte nur dann, wenn sie eine ähnliche Normale aufweisen. Der Schwellenwert auf diese Ähnlichkeit kann mit *'max_angle_diff'* in GenParamName und einem Winkel in GenParamValue angegeben werden. Der Standardwert für den Schwellenwert ist *'180'* Grad. Es kann außerdem das selbe Ausreißerkriterium wie bei der Methode *'accurate'* angewandt werden.

'xyz_mapping': Die Methode *'xyz_mapping'* nur auf Punktwolken angewendet werden, die ein XYZ-Mapping enthalten (z.B. wenn sie mittels xyz_to_object_model_3d erstellt wurden). Dieses Mapping speichert für jeden Punkt die ursprünglichen Bildkoordinaten. Die Methode *'xyz_mapping'* unterteilt diese Bilder in Quadrate, deren Seitenlängen dem Parameter SampleDistance (in Pixeln) entsprechen, und wählt einen Punkt pro Quadrat aus.

Die Methode verhält sich daher ähnlich zu einer Anwendung von zoom_image_factor auf die ursprünglichen XYZ-Bilder. Diese Methode ignoriert die 3D-Koordinaten bei der Auswahl der Punkte und verwendet lediglich die 2D Bildkoordinaten der Punkte.

Wichtig ist, dass anders als bei den anderen Methoden `SampleDistance` bei dieser Methode einem Abstand in Pixeln entspricht, nicht einem Abstand im 3D-Raum.

'xyz_mapping_compute_normals': Die Methode *'xyz_mapping_compute_normals'* berechnet die Normalen von allen Punkten die wie bei der Methode *'xyz_mapping'* ausgewählt wurden. Die z-Komponente der berechneten Normalenvektoren ist immer positiv. Enthält das Eingabeobjektmodell bereits Normalen, werden diese kopiert. Andernfalls werden die Normalen aus den XYZ-Mapping berechnet.

'furthest_point': Die Methode *'furthest_point'* fügt iterativ jeweils den Punkt des Eingabeobjekts dem Ausgabeobjekt hinzu, der am weitesten von allen bisherigen Punkten des Ausgabeobjekts entfernt ist. Für diese Methode wird in `SampleDistance` die gewünschte Anzahl der Punkte des Ausgabeobjekts übergeben. Ist diese größer oder gleich der Anzahl der Punkte des Eingabeobjekts, so werden alle Punkte des Eingabeobjekts zurückgegeben.

Der erste dem Ausgabeobjekt hinzugefügte Punkt ist derjenige, der am weitesten vom Zentrum des achsenparallelen Quaders um alle Punkte des Eingabeobjekts entfernt ist.

'furthest_point_compute_normals': Die Methode *'furthest_point_compute_normals'* berechnet die Normalen von allen Punkten, die wie bei der Methode *'furthest_point'* ausgewählt wurden. Auch bei dieser Methode wird in `SampleDistance` die Anzahl der gewünschten Punkte im Ausgabeobjekt übergeben.

Für die Normalenberechnung muss das Eingabeobjektmodell entweder Normalen aufweisen, die kopiert werden, oder ein XYZ-Mapping aus dem die Normalen berechnet werden. Die z-Komponente der berechneten Normalenvektoren ist immer positiv. Das XYZ-Mapping wird beispielsweise von `xyz_to_object_model_3d` erzeugt.

Besteht das Eingabeobjektmodell aus Flächen (Dreiecke oder Polygone) oder aus einem 3D-Primitiv, so wird die Oberfläche mit der angegebenen Distanz abgetastet. In diesem Fall wird die in `Method` angegebene Methode ignoriert. Die Orientierung der berechneten Normalen hängt vom Umlaufsinn der Flächen des Modells ab. Üblicherweise variiert der Umlaufsinn der Flächen innerhalb eines CAD-Modells nicht, wodurch die ermittelten Normalen entweder alle nach innen oder nach außen gerichtet sind. 3D-Primitive vom Typ Ebene oder Zylinder dürfen nur endliche Ausdehnung haben. Besteht das Eingabeobjektmodell aus Linien, so werden die Linien mit der angegebenen Distanz abgetastet.

Der Abtastprozess approximiert Oberflächen, indem neue Punkte im Ausgabeobjektmodell erzeugt werden. Daher gehen erweiterte Attribute des Eingabeobjektmodell verloren.

Bei gemischten Eingabeobjektmodellen entscheidet die Abtastpriorität (von hoch bis niedrig): Flächen, Linien, Primitive, und Punkte, d.h. nur die Objekte der höchsten Priorität werden abgetastet.

Für die Abtastdistanz in `SampleDistance` kann entweder ein Wert übergeben werden, der für alle in `ObjectModel3D` übergebenen 3D-Objektmodelle gültig ist, oder für jedes Eingabemodell ein eigener Wert. Die Einheit der Abtastdistanz ist die HALCON-übliche Einheit *'m'*.

Parameter

▷ **`ObjectModel3D`** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des abzutastenden 3D-Objektmodells.

▷ **`Method`** (input_control) . . . . . string ⇝ *string*
Verwendete Methode zum Abtasten.
Defaultwert: 'fast'
Werteliste: `Method` ∈ {'fast', 'fast_compute_normals', 'accurate', 'accurate_use_normals', 'xyz_mapping', 'xyz_mapping_compute_normals', 'furthest_point', 'furthest_point_compute_normals'}

▷ **`SampleDistance`** (input_control) . . . . . real(-array) ⇝ *real* / integer
Abtastdistanz.
Parameteranzahl: SampleDistance == 1 || SampleDistance == ObjectModel3D
Defaultwert: 0.05

▷ **`GenParamName`** (input_control) . . . . . string-array ⇝ *string*
Namen der generischen Parameter, die für die Abtastung eingestellt werden sollen.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'min_num_points', 'max_angle_diff'}

▷ **`GenParamValue`** (input_control) . . . . . number-array ⇝ *real* / integer / string
Werte der generischen Parameter, die für die Abtastung eingestellt werden sollen.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {1, 2, 5, 10, 20, 0.1, 0.25, 0.5}

▷ **SampledObjectModel3D** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells mit den abgetasteten Punkten.
Parameteranzahl: SampledObjectModel3D == ObjectModel3D

Beispiel

```
gen_box_object_model_3d ([0,0,0,0,0,0,0],3,2,1, ObjectModel3D)
sample_object_model_3d (ObjectModel3D, 'fast', 0.05, [], [], \
                        SampledObjectModel3D)
dev_get_window (WindowHandle)
visualize_object_model_3d (WindowHandle, SampledObjectModel3D, \
                           [], [], [], [], [], [], [], PoseOut)
```

Ergebnis

`sample_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_object_model_3d`, `gen_plane_object_model_3d`, `gen_sphere_object_model_3d`, `gen_cylinder_object_model_3d`, `gen_box_object_model_3d`, `gen_sphere_object_model_3d_center`, `xyz_to_object_model_3d`

Nachfolger

`get_object_model_3d_params`, `clear_object_model_3d`

Alternativen

`simplify_object_model_3d`, `smooth_object_model_3d`

Modul

3D Metrology

simplify_object_model_3d (: : ObjectModel3D, Method, Amount, GenParamName, GenParamValue : SimplifiedObjectModel3D)

Vereinfacht ein trianguliertes 3D-Objektmodell.

`simplify_object_model_3d` vereinfacht das triangulierte 3D-Objektmodell `ObjectModel3D`, indem sukzessiv Modellpunkte eliminiert werden und gibt das Ergebnis in `SimplifiedObjectModel3D` zurück. Im Gegensatz zu `sample_object_model_3d` wird dabei die ursprüngliche Geometrie des 3D-Objektmodells so gut wie möglich bewahrt. Dies bedeutet typischerweise, dass Kanten erhalten bleiben, die Punktdichte in glatten Objektteilen aber gleichzeitig stark reduziert wird. Dies ist z.B. hilfreich, um nachfolgende Operatoren durch die reduzierte Modellkomplexität zu beschleunigen.

Die Vermaschung des Eingabe-3D-Objektmodells bleibt im Gegensatz zum Operator `sample_object_model_3d` erhalten, der Oberflächen abtastet und dabei äquidistante, unverbundene Punkte erzeugt.

Aktuell wird nur eine Vereinfachungsmethode (*'preserve_point_coordinates'*) unterstützt, die in `Method` ausgewählt werden kann. Bei dieser Methode haben die Punkte im vereinfachten Objektmodell `SimplifiedObjectModel3D` die gleichen Koordinaten wie die entsprechenden Punkte im Eingabe-Objektmodell `ObjectModel3D`.

`simplify_object_model_3d` setzt voraus, dass die Punkte im Eingabe-Objektmodell trianguliert sind. Dies kann mit `get_object_model_3d_params` (`GenParamName`=*'has_triangles'*) abgefragt werden. Objektmodelle, die keine Triangulierung enthalten, können mit `triangulate_object_model_3d` oder `prepare_object_model_3d` (`Purpose`=*'segmentation'*) zuvor trianguliert werden.

Der Grad der Vereinfachung kann mit `Amount` festgelegt werden. Standardmäßig wird in `Amount` der Prozentsatz der Punkte des Eingabe-Objektmodells übergeben, die im Ausgabe-Objektmodell enthalten sein sollen. Je kleiner der Wert für `Amount` in diesem Fall gewählt wird, desto stärker wird das Objektmodell vereinfacht.

Alternativ kann die Bedeutung des Parameters `Amount` verändert werden. Dazu kann der generische Parameter *'amount_type'* auf die folgenden Werte gesetzt werden:

'percentage_remaining' (default): `Amount` beschreibt den Prozentsatz der Punkte des Eingabe-Objektmodells, die im Ausgabe-Objektmodell enthalten sein sollen.
Wertebereich: [*0.0* ... *100.0*].

'percentage_to_remove' : `Amount` beschreibt den Prozentsatz der Punkte des Eingabe-Objektmodells, die eliminiert werden sollen.
Wertebereich: [*0.0* ... *100.0*].

'num_points_remaining' : `Amount` beschreibt die Anzahl der Punkte des Eingabe-Objektmodells, die im Ausgabe-Objektmodell enthalten sein sollen.
Wertebereich: [*0* ... Anzahl der Punkte im Eingabe-Objektmodell].

'num_points_to_remove' : `Amount` beschreibt die Anzahl der Punkte des Eingabe-Objektmodells, die eliminiert werden sollen.
Wertebereich: [*0* ... Anzahl der Punkte im Eingabe-Objektmodell].

Manchmal kehren sich Dreiecke der Vermaschung während der Vereinfachung um, d.h. die Richtung ihrer Normalenvektoren ändert sich um 180 Grad. Dies tritt insbesondere bei künstlich generierten CAD-Modellen auf, die ebenen Teilen enthalten. Um die Umkehr der Dreiecke zu verhindern, kann der generische Parameter *'avoid_triangle_flips'* auf *'true'* gesetzt werden (der Default ist *'false'*). In diesem Fall erhöht sich allerdings die Laufzeit von `simplify_object_model_3d`.

Es ist zu beachten, dass mehrfaches Aufrufen von `simplify_object_model_3d` mit geringem Vereinfachungsgrad zu anderen Ergebnissen führen kann als ein einmaliger Aufruf mit stärkerem Vereinfachungsgrad. Isolierte (d.h. nicht-triangulierte) Punkte werden bei der Vereinfachung gelöscht. Dies kann dazu führen, dass die Anzahl der Punkte in `SimplifiedObjectModel3D` leicht vom in `Amount` gewählten Vereinfachungsgrad abweicht.

Parameter

- ▷ **`ObjectModel3D`** (input_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Objektmodells, das vereinfacht werden soll.
- ▷ **`Method`** (input_control) ..... string ⇝ *string*
  Methode, die zum Vereinfachen verwendet werden soll.
  Defaultwert: 'preserve_point_coordinates'
  Werteliste: `Method` ∈ {'preserve_point_coordinates'}
- ▷ **`Amount`** (input_control) ..... number ⇝ *real* / integer
  Maß der Vereinfachung (Default: Prozentsatz der verbleibenden Modellpunkte).
- ▷ **`GenParamName`** (input_control) ..... attribute.name-array ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'amount_type', 'avoid_triangle_flips'}
- ▷ **`GenParamValue`** (input_control) ..... attribute.value-array ⇝ *string* / real
  Werte der generischen Parameter.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {'percentage_remaining', 'percentage_to_remove', 'num_points_remaining', 'num_points_to_remove', 'true', 'false'}
- ▷ **`SimplifiedObjectModel3D`** (output_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle des vereinfachten 3D-Objektmodells.

Beispiel

```
read_object_model_3d ('mvtec_bunny.om3', 'm', [], [], ObjectModel3D, Status)
visualize_object_model_3d (WindowHandle, ObjectModel3D, [], [], [], [], \
                           [], [], [], Pose)
simplify_object_model_3d (ObjectModel3D, 'preserve_point_coordinates', \
```

```
                          5.0, 'amount_type', 'percentage_remaining', \
                          SimplifiedObjectModel3D)
visualize_object_model_3d (WindowHandle, SimplifiedObjectModel3D, [], \
                           Pose, [], [], [], [], [], Pose)
```

Ergebnis

`simplify_object_model_3d` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`prepare_object_model_3d`, `read_object_model_3d`, `triangulate_object_model_3d`, `xyz_to_object_model_3d`

Nachfolger

`disp_object_model_3d`, `smallest_bounding_box_object_model_3d`

Alternativen

`sample_object_model_3d`, `smooth_object_model_3d`

Literatur

Michael Garland, Paul S. Heckbert: Surface Simplification Using Quadric Error Metrics, Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97), 209-216, ACM Press, 1997

Modul

3D Metrology

smooth_object_model_3d

```
smooth_object_model_3d ( : : ObjectModel3D, Method, GenParamName,
    GenParamValue : SmoothObjectModel3D )
```

Glätten von 3D-Punkten eines 3D-Objektmodells.

Der Operator `smooth_object_model_3d` glättet die 3D-Punkte des in `ObjectModel3D` eingegebenen 3D-Objektmodells. Zurzeit bietet der Operator drei Verfahren zur Glättung, die im Parameter `Method` ausgewählt werden: *'mls'*, *'xyz_mapping'* und *'xyz_mapping_compute_normals'*. *'mls'* verwendet ein Moving Least Squares (MLS) Verfahren. Als Nebeneffekt des Glättens erweitert die Methode das Objektmodell `SmoothObjectModel3D` um Normalen. *'xyz_mapping'* glättet die Punktkoordinaten mit Hilfe eines 2D-Filters und des in `ObjectModel3D` enthaltenen 2D-Mappings. *'xyz_mapping_compute_normals'* glättet die Punkte wie *'xyz_mapping'* und erweitert `SmoothObjectModel3D` zusätzlich um Normalen.

Mit `GenParamName` und `GenParamValue` können zusätzliche Parameternamen und -werte eingestellt werden. Für `Method`=*'mls'* setzbare Parameternamen haben *'mls'* vorangestellt. Analog dazu besitzen die für `Method`=*'xyz_mapping'* und `Method`=*'xyz_mapping_compute_normals'* zulässigen Parameternamen das Präfix *'xyz_mapping'*.

MLS Glättung

Für `Method`=*'mls'* berechnet der MLS-Algorithmus für jeden Punkt P eine Ebene oder eine Oberfläche höherer Ordnung, welche optimal an die k-Nachbarschaft des Punktes (d.h. die nächsten k Nachbarn) angepasst wird. Das Oberflächenfitting ist im Wesentlichen als *gewichtete Methode der kleinsten Quadrate* für die Schätzung der Ebenen- bzw. Oberflächenparameter implementiert. Die nächsten Nachbarn von P haben einen größeren Einfluss auf die Parameterschätzung als die ferneren Punkte. Dieser Einfluss wird durch die folgende *Gewichtungsfunktion* und den Parameter σ gesteuert:

$$w(P') = exp\left(-\frac{\|P' - P\|^2}{\sigma^2}\right)$$

Der Punkt wird dann auf die Oberfläche projiziert. Dieses Verfahren wird für jeden einzelnen Punkt wiederholt, was in einem geglätteten Punktsatz resultiert. Die Punkte auf der gefitteten Oberfläche haben leicht zu berechnende Normalen (d.h. die Normalen können direkt aus den jeweiligen Oberflächenparametern berechnet werden). Folglich werden die Punkte nicht nur geglättet, sondern als Nebeneffekt des Glättens bekommen sie eine Schätzung für ihre Normalen.

Mit `GenParamName` und `GenParamValue` können folgende zusätzliche Parameter der *MLS Glättung* eingestellt werden:

'mls_kNN': Setzt die Anzahl der nächsten k Nachbarn, welche beim Fitten einer Ebene, bzw. Oberfläche zu jedem Punkt betrachtet werden.

Wertevorschläge: 40, *60*, *80*, *100*, *400*.

Standardwert: *60*

'mls_order': Setzt die Ordnung der MLS-Oberfläche. Für *'mls_order'=1* ist die Oberfläche eine einfache Ebene.

Wertevorschläge: 1, *2*, *3*.

Standardwert: *2*

'mls_abs_sigma': Setzt den Gewichtungsparameter σ als Festwert in Meter. Der auszuwählende Wert hängt stark vom Maßstab der eingegebenen Punktdaten ab. Als Faustregel sollte man einen Wert für σ wählen, der dem typischen Abstand jedes Punktes P zu seinem $k/2$. (mittleren) Nachbarn $P_{k/2}$ entspricht. Es ist zu beachten, dass, wenn die Daten eine variierende Dichte haben, das Setzen eines Festwerts für σ zu ungleichen Glättungsergebnissen für Bereiche unterschiedlicher Punktdichte führt. Dieses Problem kann umgegangen werden, indem man statt *'mls_abs_sigma'* den Parameter *'mls_relative_sigma'* benutzt. Letzterer ist unabhängig vom Maßstab und deswegen auch einfacher zu bedienen. Es sollte allerdings auch beachtet werden, dass der Parameter *'mls_relative_sigma'* ignoriert wird, wenn der Parameter *'mls_abs_sigma'* gesetzt ist.

Wertevorschläge: 0.0001, *0.001*, *0.01*, *0.1*, *1.0*.

'mls_relative_sigma': Setzt einen Multiplikationsfaktor σ_{rel}, der verwendet wird, um σ_P für jeden Punkt P nach der Formel zu berechnen:

$$\sigma_P = \sigma_{rel} \| P_{k/2} - P \|,$$

wobei $P_{k/2}$ der $k/2$. Nachbar von P ist. Anders als der global für alle Punkte wirkende Parameter σ wird σ_P allein für den jeweiligen Punkt P berechnet. Dadurch werden Probleme vermieden, welche durch das Verwenden eines festen Wertes für Daten mit variierender Punktdichte entstehen. Es sollte allerdings auch beachtet werden, dass der Parameter *'mls_relative_sigma'* ignoriert wird, wenn der Parameter *'mls_abs_sigma'* gesetzt ist.

Wertevorschläge: 0.1, *0.5*, *1.0*, *1.5*, *2.0*.

Standardwert: *1.0*.

'mls_force_inwards': Ist dieser Parameter auf den Wert *'true'* gesetzt, so werden alle Normalen so orientiert, dass sie „zum Koordinatenursprung zeigen“. Mathematisch ausgedrückt bedeutet das, dass das Skalarprodukt zwischen dem Normalenvektor und dem Vektor, der vom entsprechenden Oberflächenpunkt zum Koordinatenursprung zeigt, positiv ist. Dies kann notwendig sein, wenn das resultierende `SmoothObjectModel3D` für das oberflächenbasierte Matching verwendet werden soll — entweder als Modell im Operator `create_surface_model` oder als 3D-Szene im Operator `find_surface_model` — da hier die konsistente Orientierung der Normalen wichtig ist. Wird der Parameter *'mls_force_inwards'* auf den Wert *'false'* gesetzt, so ist die Orientierung der Normalen beliebig.

Werteliste: 'true', *'false'*.

Standardwert: *'true'*.

Glättung des 2D-Mappings

Für `Method`=*'xyz_mapping'* oder `Method`=*'xyz_mapping_compute_normals'* werden die Punkte des 3D-Objektmodells durch die Anwendung eines 2D-Filters auf dem enthaltenen Tiefenbild geglättet. Für `Method`=*'xyz_mapping_compute_normals'* wird `SmoothObjectModel3D` zusätzlich um Normalen erweitert, die anhand des XYZ-Mappings berechnet werden. Enthält `ObjectModel3D` kein 2D-Mapping, wird ein Fehler geworfen. Da das 3D-Objektmodell anhand des 2D-Mappings geglättet wird, ist die Verwendung von `Method`=*'xyz_mapping'* oder `Method`=*'xyz_mapping_compute_normals'* üblicherweise schneller als `Method`=*'mls'*. Invalide Punkte (z.B. duplizierte Punkte mit Koordinaten [0,0,0]) sollten vor Anwendung des Operators aus dem 3D-Objektmodell entfernt werden, beispielsweise mit `select_points_object_model_3d` und Attribut *'point_coord_z'* oder *'num_neighbors_fast X'*.

Mit GenParamName und GenParamValue können folgende zusätzliche Parameter der *2D Glättung* eingestellt werden:

'xyz_mapping_filter': Setzt den verwendeten Filter zur Glättung des 2D-Mappings. Die Größe der Filtermaske kann mit *'xyz_mapping_mask_width'* und *'xyz_mapping_mask_height'* gesetzt werden.
Im Standardmodus *'median_separate'* ist der verwendete Algorithmus vergleichbar mit median_separate. Dieser Modus ist häufig schneller als *'median'*, kann jedoch auch zu weniger präzisen Ergebnissen oder Artefakten an Oberflächendiskontinuitäten führen.
Im Modus *'median'* ist der Algorithmus vergleichbar mit median_image.
Werteliste: 'median_separate', 'median'.
Standardwert: 'median_separate'.

'xyz_mapping_mask_width', *'xyz_mapping_mask_height'*: Setzen Breite und Höhe der verwendeten Filtermaske.
Für *'xyz_mapping_filter'='median_separate'* und *'xyz_mapping_filter'='median'* werden gerade Werte für *'xyz_mapping_mask_width'* oder *'xyz_mapping_mask_height'* automatisch auf den nächstgrößeren ungeraden Wert gesetzt.
Für *'xyz_mapping_filter'='median'* muss die verwendete Filtermaske quadratisch sein (*'xyz_mapping_mask_width'* = *'xyz_mapping_mask_height'*). Wenn nur *'xyz_mapping_mask_width'* oder *'xyz_mapping_mask_height'* gesetzt wird, wird der andere Parameter automatisch auf den gleichen Wert gesetzt. Werden zwei verschiedene Werte gesetzt, wird ein Fehler geworfen.
Wertevorschläge: 3, 5, 7, 9.
Standardwert: 3.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle eines 3D-Objektmodells mit 3D-Punktdaten.

▷ **Method** (input_control) . . . . . string ⇝ *string*
Glättungsmethode.
Defaultwert: 'mls'
Werteliste: Method ∈ {'mls', 'xyz_mapping', 'xyz_mapping_compute_normals'}

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'mls_kNN', 'mls_order', 'mls_abs_sigma', 'mls_relative_sigma', 'mls_force_inwards', 'xyz_mapping_filter', 'xyz_mapping_mask_width', 'xyz_mapping_mask_height'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {10, 20, 40, 60, 0.1, 0.5, 1.0, 2.0, 0, 1, 2, 3, 5, 7, 9}

▷ **SmoothObjectModel3D** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des geglätteten 3D-Objektmodells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

surface_normals_object_model_3d, sample_object_model_3d, simplify_object_model_3d

Modul

3D Metrology

surface_normals_object_model_3d (: : ObjectModel3D, Method, GenParamName, GenParamValue : ObjectModel3DNormals)

Berechnet die 3D-Oberflächennormalen eines 3D-Objektmodells.

Der Operator surface_normals_object_model_3d berechnet die 3D-Oberflächennormalen eines Objekts ObjectModel3D. Die Berechnung wird mit dem in Method gewählten Verfahren durchgeführt.

Für Method='mls' wird das Verfahren Moving Least Squares (MLS) angewandt. Das MLS-Verfahren für Normalenberechnung ist identisch zu dem MLS-Verfahren für Punktglättung, das im Operator smooth_object_model_3d implementiert ist. Anders als smooth_object_model_3d glättet der Operator surface_normals_object_model_3d die Punkte allerdings nicht, d.h. die ursprünglichen 3D-Punkte von ObjectModel3D bleiben unverändert. Weitere Information über das MLS-Verfahren und seine Parameter stehen mit der Beschreibung des Operators smooth_object_model_3d zur Verfügung.

Beinhaltet das Objekt ObjectModel3D Dreiecke, können mit dem Verfahren Method='triangles' die Punktnormalen aus den Flächennormalen der angrenzenden Dreiecke berechnet werden. Die Flächennormalen werden dabei mit dem Winkel gewichtet, den die Dreiecke an dem Punkt umschließen.

Beinhaltet das Objekt ObjectModel3D ein 2D-Mapping (wie z.B. 3D-Objektmodelle, die mittels xyz_to_object_model_3d erstellt wurden), können mit dem Verfahren Method='xyz_mapping' die Punktnormalen aus der Umgebung der Punkte im 2D-Mapping berechnet werden. Dazu wird in einer 5x5-Nachbarschaft der Punkte im 2D-Mapping eine Ebene durch die zugehörigen 3D-Punkte gefittet. Die Richtung der Normale dieser Ebene wird anschließend konsistent zum 2D-Mapping gewählt, z.B. in oder entgegen der Blickrichtung des Sensors.

Kann der Normalenvektor eines Punktes nicht bestimmt werden, so wird dieser auf 0 gesetzt. Dies ist beispielsweise der Fall, wenn das 3D-Objektmodell den gleichen Punkt mehr als 'mls_kNN' mal enthält.

Parameter

▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle eines 3D-Objektmodells mit 3D-Punktdaten.

▷ **Method** (input_control) . . . . . string ⇝ *string*
Methode für Normalenberechnung.
Defaultwert: 'mls'
Werteliste: Method ∈ {'mls', 'triangles', 'xyz_mapping'}

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'mls_kNN', 'mls_order', 'mls_abs_sigma', 'mls_relative_sigma', 'mls_force_inwards'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {10, 20, 40, 60, 0.1, 0.5, 1.0, 2.0, 0, 1, 2, 'true', 'false'}

▷ **ObjectModel3DNormals** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
Handle des 3D-Objektmodells mit berechneten 3D-Normalen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

sample_object_model_3d

Nachfolger

create_surface_model, fuse_object_model_3d

Alternativen

smooth_object_model_3d

Modul

3D Metrology

```
triangulate_object_model_3d ( : : ObjectModel3D, Method,
    GenParamName, GenParamValue : TriangulatedObjectModel3D,
    Information )
```

Erstellen einer Oberflächentriangulierung eines 3D-Objekt-Modells.

Der Operator `triangulate_object_model_3d` erstellt eine Oberfläche aus dreieckigen Flächen für das 3D-Objekt-Modell `ObjectModel3D` und liefert die dadurch entstandene Oberfläche in `TriangulatedObjectModel3D` zurück. Zur Zeit bietet der Operator vier Verfahren zur Triangulierung, welche im Parameter `Method` ausgewählt werden: *'polygon_triangulation'*, *'xyz_mapping'*, *'greedy'* und *'implicit'*. *'polygon_triangulation'* ist eine einfache Routine, welche Polygon-Flächen der bestehenden Oberfläche in dreieckige Flächen umwandelt. *'xyz_mapping'* trianguliert die Punkte in 2D anhand eines 2D-Mapping. Die anderen zwei Verfahren sind eher komplexer. Bei ihnen wird eine Oberfläche aus reinen 3D-Punktdaten berechnet, wobei die Oberflächentopologie unbekannt ist. Ein detaillierter Vergleich der *'greedy'* und *'implicit'* Algorithmen kann weiter unten im Paragraph *"Vergleich der Triangulierungsmethode"* nachgelesen werden.

Triangulierung von Polygonen

Wenn `Method`=*'polygon_triangulation'*, dann werden alle Polygonen des Objekts `ObjectModel3D` trianguliert. Für dieses Verfahren werden keine generische Parameter unterstützt.

Triangulierung eines 2D-Mapping

Wenn `Method`=*'xyz_mapping'*, dann werden die Punkte in 2D anhand eines 2D-Mapping des Objekts `ObjectModel3D` trianguliert. Dabei wird dieselbe Methode benutzt wie in `prepare_object_model_3d` für Purpose='segmentation'. Enthält `ObjectModel3D` kein 2D-Mapping, wird ein Fehler geworfen.

Als Nachverarbeitungsschritt können Dreiecke entfernt werden, deren Normalenrichtung sich stark von einer festgelegten Richtung unterscheiden. Genauere Informationen zur Anwendung sind in der Beschreibung von `GenParamName` zu finden. Dies ist dann hilfreich falls die 2D-Nachbarschaft, die zur Triangulierung verwendet wird, die 3D-Nachbarschaft nicht gut wiedergibt, beispielsweise wenn entlang der Blickrichtung des Sensors Teile der Oberfläche verdeckt sind oder die entlang Blickrichtung typischen Störungen auftreten.

Mit `GenParamName` und `GenParamValue` kann der folgende zusätzliche Parameter der Map-Triangulierung eingestellt werden:

'xyz_mapping_max_area_holes' legt fest, welche Fläche die Löcher in den Punktkoordinaten haben dürfen, damit diese mittels einer Delaunay-Triangulierung geschlossen werden. Nur Löcher, die vollständig von der Bildregion eingeschlossen sind, werden geschlossen. Falls *'xyz_mapping_max_area_holes'* auf *0* gesetzt ist, werden keine Löcher trianguliert. Der Parameter entspricht dem `GenParamName` *'max_area_holes'* von `prepare_object_model_3d`.

Wertevorschläge: 1, *10*, *100*.

Default: 10.

'xyz_mapping_max_view_angle' legt die maximal zulässige Winkeldifferenz zwischen der Normalen der Dreiecken und der Blickrichtung des Sensors fest. Umso kleiner dieser Wert gesetzt wird, desto weniger Dreiecke werden zurückgegeben. Als Blickrichtung des Sensors wird die z-Achse des Koordinatensystems von `ObjectModel3D` angenommen, soweit nicht mittels `GenParamName` eine andere Blickrichtung (*'xyz_mapping_max_view_dir_x'*, *'xyz_mapping_max_view_dir_y'*, *'xyz_mapping_max_view_dir_z'*) spezifiziert wird. Es muss ein Winkel zwischen 0 und 90 Grad gesetzt werden.

Wertevorschläge: 'rad(60)', *'rad(85)'*, *'rad(90)'*.

Default: 'rad(90)'.

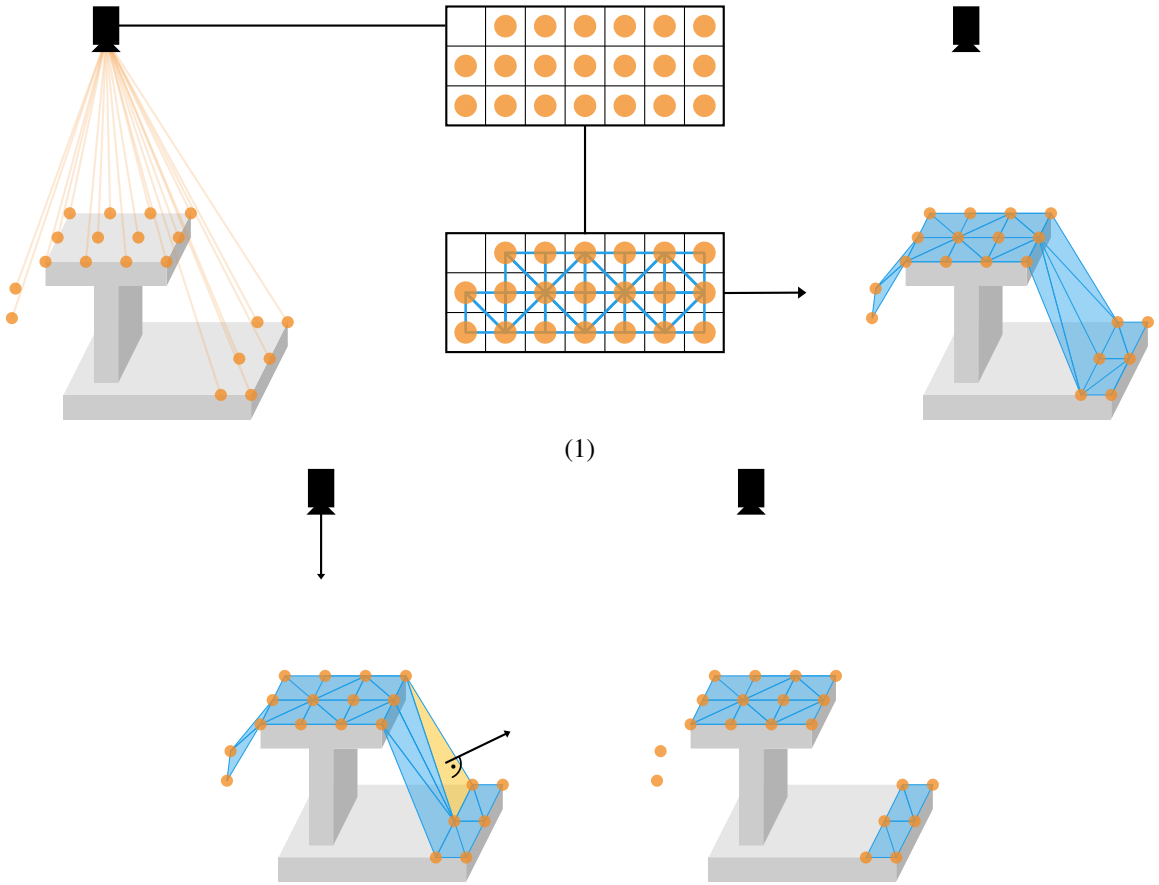
'xyz_mapping_max_view_dir_x', *'xyz_mapping_max_view_dir_y'*, *'xyz_mapping_max_view_dir_z'* legen die verwendete Blickrichtung des Sensors im Koordinatensystem von `ObjectModel3D` fest, wenn *'xyz_mapping_max_view_angle'* für `GenParamName` gesetzt wird. Falls nicht alle drei Richtungen gleichzeitig gesetzt werden oder die Richtung dem Nullvektor entspricht wird ein Fehler geworfen.

Wertevorschläge: [1, 0, 0], *[0, 0, 1]*.

Default: [0, 0, 1].

'xyz_mapping_output_all_points' legt fest, ob alle Eingabepunkte zurückgegeben werden, unabhängig davon ob sie für die Triangulierung verwendet wurden. Diese Option wird hauptsächlich aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Wird *'xyz_mapping_output_all_points'* auf *'false'* gesetzt, werden die alten Punktindizes als das erweiterte Attribut *'original_point_indices'* im 3D-Objekt-Modell `TriangulatedObjectModel3D` abgespeichert. Dieses Attribut kann daraufhin mit

get_object_model_3d_params abgefragt werden, oder mit Operatoren weiterverarbeitet werden, welche erweiterte Attribute verwenden.
Werteliste: 'false', 'true'.
Default: 'false'.



(1) (2)

(1) Zur Triangulierung des 3D-Objekt-Modells wird ein 2D-Mapping des Modells genutzt. Die Triangulierung erfolgt dann auf Basis der entsprechenden 2D-Nachbarschaft. Dabei können ungewollte Dreiecke entlang der Sensorblickrichtung entstehen, beispielsweise weil Teile des Objekts verdeckt sind, oder durch Rauschen. (2) Durch einen Vergleich der Normalenrichtung der Dreiecke und der Blickrichtung des Sensors wird entschieden, ob ein Dreieck zurückgegeben wird. Die maximale Abweichung ist in *'xyz_mapping_max_view_angle'* festgelegt.

Greedy-Triangulierung

Wenn Method=*'greedy'*, dann wird der so genannte 'greedy' (gierig) Triangulierungsalgorithmus aufgerufen. Dieser Algorithmus braucht 3D-Punktdaten mit Normalen. Wenn die Normalen nicht im ObjectModel3D vorhanden sind, werden sie vorher intern berechnet. Die Durchführung erfolgt mit der gleichen Methode wie in surface_normals_object_model_3d unter Verwendung der Defaultwerte. Der Triangulierungsalgorithmus baut eine Oberfläche, welche durch die einzelnen Punkte verläuft und deren Orientierung mit der Orientierung der entsprechenden Punktnormale bis auf eine bestimmte Toleranz übereinstimmt. Die Oberfläche wird aus einzelnen dreieckigen Flächen zusammengesetzt, welche Dreiergruppen von *benachbarten* Punkten verbinden. Um bestimmen zu können, welche Dreiergruppen sich für ein Oberflächendreieck eignen, verwendet der Algorithmus einen lokalen Nachbarschaftstest. Er wird **surface neighborhood criteria (SNC)**, bzw. **Nachbarschaftsbedingungen innerhalb einer Oberfläche** genannt:

Liegt ein Punkt P auf einer Oberfläche mit Orientierung (Normale) N, dann kann angenommen werden, dass der Punkt P' mit der Normale N' auch auf dieser Oberfläche liegt, wenn:

1. Der Abstand zwischen beiden Punkten ist kleiner gleich r, d.h., $\Delta(P, P') \leq r$
2. Beide Normalen sind ähnlich orientiert, d.h. der Winkel dazwischen ist $\angle(N, N') \leq \alpha$ oder auch, wenn keine strenge Konsistenz der Normalen erforderlich ist, $\angle(N, -N') \leq \alpha$

3. Der Vektor $\delta P = P' - P$ ist möglichst orthogonal zu der Normale N, d.h. der Winkel ist $|90° - \angle(N, \delta P)| \leq \beta$
4. Wenn P' die 3. Regel nicht erfüllt dafür aber nicht weiter als d von der durch $[P, N]$ definierten Ebene liegt, wird der Punkt trotzdem akzeptiert.

Die vier Parameter r (siehe *'greedy_radius_type'* und *'greedy_radius_value'*), α (siehe *'greedy_neigh_orient_tol'*), β (siehe *'greedy_neigh_latitude_tol'*) und d (siehe *'greedy_neigh_vertical_tol'*) steuern die Nachbarschaftsbedingungen SNC und haben die folgenden Bedeutungen:

Der Parameter α steuert im Grunde genommen die Krümmung der generierten Oberfläche: niedrigere α-Werte ergeben eine - lokal betrachtet - flachere Oberfläche.

Die anderen drei Parameter definieren den Anteil einer Kugel, in deren innerem Raum alle Punkte gültige SNC Nachbarn des Kugelzentrums sind. Die Kugel hat einen Radius von r, ihr Zentrum liegt in P und ihr Äquatorebene entspricht der Ebene $[P, N]$. Nur Punkte innerhalb r können als Nachbarn betrachtet werden (1. SNC-Regel). Die weiteren Voraussetzungen diese Punkte als gültige SNC-Nachbarn zu erkennen sind: sie müssen entweder einen Breitengrad (im Sinne einer "geographischen"Breite) innerhalb des Bereichs [-β; β] relativ zu der Äquator haben (2. SNC-Regel), oder sie müssen innerhalb einer Schicht beiderseits des Äquators, bzw. nicht weiter als d von ihm entfernt liegen (4. SNC-Regel).

Der Parameter r verhindert, dass der Algorithmus zu große Dreiecke erzeugt. Das ist insbesondere dann wichtig, wenn die Punkte entweder mehrere einzelne, getrennte Oberflächen, oder Oberflächen mit nicht zu schließenden Lücken darstellen. Das von β definierte Breitengradfenster erlaubt, dass Punkte, die auf Grund von Rauschen oder Oberflächenkrümmung außerhalb der Ebene $[P, N]$ liegen, auch als Nachbarn in Betracht gezogen werden. Ähnlich erlaubt der Parameter d, dass Punkte, welche knapp ober- oder unterhalb der Äquatorebene liegen, auch als Nachbarn betrachtet werden können. Letzteres ist wichtig, um Probleme auf Grund von Datenrauschen zu vermeiden.

Hier werden einige Hinweise angeboten, wie man die einzelne Parameter einstellen kann:

- Wenn die resultierende Oberfläche zu unterbrochen oder lückenhaft aussieht, dann könnte das ein Hinweis dafür sein, dass r zu niedrig ist. Ein höherer Wert von r sollte diese Probleme lösen.
- Wenn die Normalen im `ObjectModel3D` verrauscht sind (d.h. die Normalen benachbarter Punkten weichen stark voneinander ab), dann ist normalerweise der Wert von α zu erhöhen. Es gibt zwei mögliche und typische Quellen für verrauschte Normalen. Die eine liegt an Ungenauigkeiten verursacht von dem ursprünglichen Sensor, von dem die 3D-Punkte, bzw. -Normalen stammen. Die zweite ist eine ungenaue Normalenberechnung in einer Routine, die verwendet wurde, um die Normalen aus den reinen Punktdaten zu berechnen.
- Die Punkte können eine Oberfläche mit sehr starker Krümmung darstellen, d.h. die Oberfläche weist eine sehr feine Struktur auf, z.B., sind kleine Dellen, feine Wellen, Falten oder scharfe Kanten zu erkennen. Dann sollen typischerweise die Parameter α und/oder β erhöht werden.
- Umgekehrt kann die Oberfläche eher flach aussehen, dafür aber viele Ausreißer haben (d.h. Punkte welche nah an der Oberfläche liegen, dafür aber eine komplett falsche Orientierung haben und deswegen nicht zu der Oberfläche gehören). Dann soll typischerweise der Parameter α niedriger gestellt werden, um die Ausreißer aus der Oberflächengenerierung auszuschließen.
- Wenn die Punkte sehr verrauscht sind und eher eine breite Schicht formen als eine eindeutige glatte Oberfläche, dann sollen β und/oder d erhöht werden. Damit können auch Punkte außerhalb der optimalen $[P, N]$-Ebene immer noch als Nachbarn von P angenommen werden.
- Umgekehrt, wenn die Daten eher rauschfrei sind, dafür aber zwei Oberflächen sehr nah parallel aneinander (z.B. Oberflächen welche beide Seiten eines flachen Objekts darstellen) verlaufen, dann sollen typischerweise β und d niedriger eingestellt werden, sodass sich beide Oberflächen nicht gegenseitig stören.

Der **Greedy-Triangulalierungs-Algorithmus** fängt mit der Initialisierung einer Oberfläche an, indem er ein erstes Dreieck aus drei SNC-geeigneten, benachbarten Punkten bildet. Weisen alle gültigen Nachbarschaften lokale Inkonsistenzen wie kollineare oder 'doppelte' Punkte auf, wird ein Fehler geworfen. Ein vorangehender Aufruf von `sample_object_model_3d` mit Method 'fast' und kleiner SampleDistance wird die lokalen Inkonsistenzen von `ObjectModel3D` in den meisten Fällen beseitigen. Ist ein erstes Dreieck gebildet, setzt der Algorithmus "gierig"immer ein neues Dreieck dazu, wenn er von einem beliebigen Punkt am Oberflächenrand einen neuen Punkt nach dem SNC-Regeln erreichen kann. Wenn keine neue Punkte nach diesen Regeln erreicht werden können, es aber immer noch Punkte gibt, welche noch gar nicht angefasst worden sind, dann wird eine neue Oberfläche initialisiert. Da die SNC-Regeln lediglich nur in für eine vergleichsweise kleine lokale Nachbarschaft einzelner Punkte definiert sind, kann die resultierende Oberfläche globale, topologische Artefakte aufweisen wie,

z.B., Lücken oder Flips. Letztere entstehen, wenn die Oberfläche sich selber trifft während sie wächst, allerdings mit invertierter Orientierung beiderseits der aufeinandertreffenden Fronten (die Oberfläche ist irgendwo entlang ihres Wachstumsverlaufs "geflippt"). Diese Artefakte werden in zwei Nachbearbeitungsstufen behandelt: *hole filling*, bzw. *flip resolving*.

Zum Schluss kann noch zusätzlich *mesh morphology* auf die resultierende Oberfläche angewendet werden, um Artefakte am Oberflächenrand zu eliminieren. Die Mesh-Morphologie besteht aus einigen Erosions- und einigen Dilatationsschleifen. Jede Erosionsschleife trägt alle Dreiecke ab, die von allen Oberflächenrändern direkt erreichbar sind. Damit schrumpft die Oberfläche zusammen. Im Nachhinein werden mit jeder Dilatationsschleife alle gerade abgetragenen Dreiecke außerhalb der zusammengeschrumpften Ränder, die direkt von den Rändern erreichbar sind, wieder angeschlossen. Es ist zu beachten, dass Dreiecke welche vor dem Anfang der Morphologie (d.h. vor der ersten Erosionsschleife) gar nicht existiert haben, von einer Dilatationsschleife auch nicht generiert werden können. Werden die ursprünglichen Oberflächenränder einmal von der Dilatation erreicht, so kann diese keine weitere Schleifen erzeugen. Deswegen kann die Anzahl der Dilatationsschleifen nicht höher sein als die der Erosionsschleifen. Die Anwendung von Mesh-Erosion und anschließend Mesh-Dilatation ist analog zu den HALCON üblichen Region-Opening. Zusätzlich gehört zu der Mesh-Morphologie auch ein Schritt, in welchem alle Oberflächen beseitigt werden, wenn die Anzahl ihrer Dreiecke unter einer bestimmten Schwelle liegt.

Die einzelnen Schritte des Algorithmus sind hier zusammengefasst.

1. Triangulierung aller Punkte, die durch SNC erreichbar sind
2. Hole filling (siehe *'greedy_hole_filling'*)
3. Flip resolving (siehe *'greedy_fix_flips'*)
4. Mesh-Morphologie (siehe *'greedy_mesh_erosion'*, *'greedy_mesh_dilation'* und *'greedy_remove_small_surfaces'*)

Mit `GenParamName` und `GenParamValue` können die folgenden zusätzlichen Parameternamen und -werte des Greedy-Algorithmus eingestellt werden:

'greedy_kNN' setzt die Anzahl k der zu betrachtenden Nachbarn. Der Algorithmus betrachtet nur die nächsten k-Nachbarn eines Punkts, um von diesen die SNC-Nachbarn zu ermitteln.

Wertevorschläge: 20, *30*, *40*, *50*, *60*.

Default: 40

'greedy_radius_type': wenn *'greedy_radius_type'='fixed'*, dann stellt *'greedy_radius_value'* den SNC-Radius r in Meter ein.

Wenn *'greedy_radius_type'='z_factor'*, dann wird r für jeden Punkt P berechnet, indem die Z-Koordinate des Punkts mit dem Wert des Parameters *'greedy_radius_value'* multipliziert wird. Diese Darstellung von r ist besser für Daten geeignet, bei welchen die Punktdichte mit dem Abstand von dem Sensor, mit welchem die Punkte aufgenommen wurden, korreliert. Das ist typischerweise der Fall bei der Verwendung von TOF-Kameras.

Wenn *'greedy_radius_type'='auto'*, dann wird intern vom Algorithmus entschieden, ob ein Radius vom Typ *'fixed'* oder *'z_factor'* besser für die Daten geeignet ist. Der Radius wird auch intern geschätzt und mit dem vom Benutzer eingegebenen Wert des Parameters *'greedy_radius_value'* multipliziert. Dadurch ist *'greedy_radius_value'* als Faktor einsetzbar, um den geschätzten Radius bei Bedarf zu korrigieren.

Werteliste: 'auto', *'fixed'*, *'z_factor'*.

Default: 'auto'.

'greedy_radius_value': siehe *'greedy_radius_type'*.

Wertevorschläge: 0.01, *0.05*, *0.5*, *0.66*, *1.0*, *1.5*, *2.0*, *3.0*, *4.0*

'greedy_neigh_orient_tol': setzt den SNC-Parameter α in Gradeinheiten. α steuert die Oberflächenkrümmung (siehe die oben definierten SNC-Regeln).

Wertevorschläge: 10, *20*, *30*, *40*.

Default: 30.

'greedy_neigh_orient_consistent': erzwingt, dass die Normalen von zwei SNC-benachbarten Punkte eine ähnliche Orientierung haben müssen (d.h. sie zeigen nicht in Gegenrichtungen). Wenn dieser Parameter freigeschaltet ist, wird der zweite Teil der 2. SNC-Regel entkräftet, d.h. wenn $\angle(N, N') > \alpha$ dann können beide Punkte nicht als SNC-Nachbarn betrachtet werden selbst wenn $\angle(N, -N') \leq \alpha$.

Werteliste: 'true', *'false'*.

Default: 'false'.

'greedy_neigh_latitude_tol': setzt den SNC-Parameter β in Gradeinheiten. β steuert die Größe des Breitengradfensters (siehe die oben definierten SNC-Regeln).

Wertevorschläge: 10, *20*, *30*, *40*.

Default: 30.

'greedy_neigh_vertical_tol': setzt den SNC-Parameter d als Multiplikationsfaktor relativ zum Radiuswert r.

Wertevorschläge: 0.01, *0.1*, *0.2*, *0.3*.

Default: 0.1.

'greedy_hole_filling': setzt die maximale Länge der Oberflächenränder (als Anzahl von Punkten), welche zum Schließen von Lücken in Betracht gezogen werden müssen. Wenn *'greedy_hole_filling'='false'*, dann wird das Lückenschließen deaktiviert.

Wertevorschläge: 'false', *20*, *40*, *60*.

Default: 40.

'greedy_fix_flips': aktiviert/deaktiviert den Orientierungskorrektur-Schritt des Algorithmus.

Werteliste: 'true', *'false'*.

Default: 'true'.

'greedy_prefetch_neighbors': aktiviert/deaktiviert das Vorbereiten gecachter Listen von Nachbarn. Diese Listen enthalten die Indizes der nächsten k Punkte für jeden Punkt. Die Listen tragen erheblich zu der Algorithmusgeschwindigkeit bei, aber sie verursachen einen vergleichsweise hohen Speicherverbrauch (in einer Größenordnung von $(O(kn)$, wobei k ist der mit *'greedy_kNN'* gesetzten Wert und n die Anzahl der Punkte im `ObjectModel3D` ist). Für größere Punktmengen kann es vorkommen, dass der benötigte Speicher nicht allokiert werden kann. Dann liefert der Operator entsprechend eine Speicher bezogene Fehlermeldung zurück. In solchen Fällen sollen die gecachten Listen deaktiviert werden.

Werteliste: 'true', *'false'*.

Default: 'true'.

'greedy_mesh_erosion': setzt die Anzahl von Erosionsschleifen, welche im Mesh-Morphologie-Schritt des Algorithmus angewendet werden müssen.

Wertevorschläge: 0, *1*, *2*, *3*.

Default: 0.

'greedy_mesh_dilation': setzt die Anzahl von Dilatationsschleifen. Die Mesh-Dilatation wird nach der Mesh-Erosion angewendet. Wenn *'greedy_mesh_dilation'* einen höheren Wert als *'greedy_mesh_erosion'* bekommt, wird der Parameter intern auf den Wert von *'greedy_mesh_erosion'* gesetzt.

Wertevorschläge: 0, *1*, *2*, *3*.

Default: 0.

'greedy_remove_small_surfaces': steuert die Entfernung kleinerer Oberflächen von dem Endergebnis. Wenn *'greedy_remove_small_surfaces'='false'*, dann wird die Entfernung deaktiviert. Wenn ein Wert zwischen *0.0* und *1.0* gesetzt wird, dann werden alle Oberflächen entfernt, die weniger Dreiecke haben als *'greedy_remove_small_surfaces'*×`num_triangles`, wobei `num_triangles` die gemeinsame Anzahl der Dreiecke in `TriangulatedObjectModel3D` angibt. Wenn der Wert höher als *1* gesetzt ist, dann werden alle Oberflächen entfernt, die weniger Dreiecke haben als *'greedy_remove_small_surfaces'*.

Wertevorschläge: 'false', *0.01*, *0.05*, *0.1*, *10*, *100*, *1000*, *10000*.

Default: 'false'.

'greedy_timeout': der Timeout gibt die Möglichkeit den Operator nach einer bestimmten Zeitspanne in Sekunden zu unterbrechen. Das ist insbesondere dann wichtig, wenn eine maximale Zeit der Ausführung unabhängig von dem Ergebnis des Operators zu gewährleisten ist. Die zeitliche Auflösung des Timeout-Mechanismus liegt bei ungefähr 10 ms. Werte kleiner als 0 werden nicht angenommen. Durch das Setzen von *'greedy_timeout'* auf *'false'* wird der Timeout deaktiviert. Die zeitliche Genauigkeit des Timeouts hängt von vielen Faktoren ab, darunter ist die Größe des Modells, die Geschwindigkeit des Rechners und der *'timer_mode'* der über `set_system` gesetzt wurde.

Wertevorschläge: 'false', *0.1*, *0.5*, *1*, *10*, *100*.

Default: 'false'.

'greedy_suppress_timeout_error': wenn ein Timeout auftritt, liefert der Operator eine entsprechende Fehlermeldung zurück. Wenn *'greedy_suppress_timeout_error'* auf *'true'* gesetzt ist, liefert der Operator keine Fehlermeldung mehr, sondern liefert die Zwischenergebnisse der Vermaschung in

TriangulatedObjectModel3D zurück. Ob ein Timeout aufgetreten ist, sieht man im Feld *'timeout_occured'* des Ausgabeparameters Information (nur im *'verbose'*-Modus).

Werteliste: 'false' 'true'.

Default: 'false'.

'greedy_output_all_points': bestimmt ob alle Eingabepunkte zurückgeliefert werden, unabhängig davon ob sie für die ausgegebene Triangulation benutzt wurden oder nicht. Dieser Parameter wird primär aufgrund der Rückwärtskompatibilität zur Verfügung gestellt. Ist *'greedy_output_all_points'* auf *'false'* gesetzt, werden die alten Punktindizes als erweitertes Attribut im 3D-Objekt-Modell TriangulatedObjectModel3D unter dem Namen *'original_point_indices'* gespeichert. Dieses Attribut kann später von Operatoren aufgerufen werden, welche erweiterte Attribute verwenden wie beispielsweise get_object_model_3d_params.

Werteliste: 'false', 'true'.

Default: 'false'.

'information': legt fest welche Informationen in Information zurückgegeben werden. Wenn *'information'='num_triangles'* (default), dann wird nur die Anzahl der generierten Dreiecke zurückgeliefert. Wenn *'information'='verbose'* wird eine Liste von Name-Wert Paaren zurückgeliefert. Zur Zeit enthalten die Paare die folgenden Informationen:

Name	Wert	Beschreibung
'num_triangles'	<Anzahl der Dreiecke>	gibt die Anzahl der generierten dreieckigen Flächen zurück.
'specified_radius_type'	*'auto'* | *'fixed'* | *'z_factor'* | *'none'*	gibt den Radiustyp zurück, der vom Benutzer gesetzt wurde.
'specified_radius_value'	<specified radius value>	gibt den Radiuswert zurück, der vom Benutzer gesetzt wurde.
'used_radius_type'	*'fixed'* | *'z_factor'* | *'sampling'*	gibt den intern verwendeten Radiustyp zurück; wenn der Benutzer *'auto'* für *'specified_radius_type'*, setzt, dann gibt dieses Feld zurück, was für ein Radiustyp von dem Algorithmus intern ausgewählt wurde; wenn `ObjectModel3D` ein 3D-Primitiv ist, dann wird der vom Benutzer gesetzten Radiuswert intern als Abtastschritt verwendet und *'used_radius_type'* gibt 'sampling' zurück.
'used_radius_value'	<interner Radiuswert>	gibt den intern verwendeten Radiuswert zurück; wenn *'used_radius_type'='fixed'*, dann wird der SNC-Nachbarschaftsradius in Meter zurückgegeben; wenn *'used_radius_type'='z_factor'*, dann wird der Multiplikationsfaktor zurückgegeben, welcher benutzt wurde, um den SNC-Nachbarschaftsradius aus der Z-Koordinate des Zentrumspunkts zu berechnen; wenn *'used_radius_type'='sampling'*, dann wird der Abtastschritt zurückgegeben, der für die Triangulierung eines 3D-Primitivs verwendet wurde, insbesondere für Zylinder und Kugel.
'neigh_orient_tol'	<α>	gibt den Krümmungsparameter α in Grad zurück, welcher für die Triangulierung verwendet wurde.
'neigh_latitude_tol'	<β>	gibt den Breitengradtoleranzwinkel β in Grad zurück, welcher für die Triangulierung verwendet wurde.
'neigh_vertical_tol'	<d>	gibt den Abstandsparameter d als Faktor von `used_radius_value` zurück.
'fix_flips'	*'true'* | *'false'*	gibt zurück ob der Orientierungskorrektur-Schritt aktiviert wurde.
'hole_filling'	*'false'* | <max hole boundary length>	gibt *'false'* zurück, wenn das Schließen von Lücken vom Benutzer deaktiviert wurde, ansonsten den maximalen Umkreis (als Anzahl von Punkten), der in Betracht gezogenen Lücken.
'timeout'	*'false'* | <timeout>	gibt *'false'* zurück, wenn der Timeout deaktiviert wurde, oder die Länge des aktivierten Timeouts in Sekunden.
'timeout_occured'	*'yes'* | *'no'*	gibt zurück ob ein Timeout aufgetreten ist.

Werteliste: 'num_triangles', 'verbose'.
Default: 'num_triangles'.

Implizite Triangulierung

Wenn `Method`=*'implicit'*, dann wird ein impliziter Triangulierungsalgorithmus aufgerufen, der auf dem Poisson-Gleichungslöser basiert (siehe die Referenzen). Er erstellt eine wasserdichte Oberfläche, d.h. die Oberfläche ist komplett geschlossen. Die implizite Triangulierung benötigt 3D-Punktdaten mit Normalen. Darüber hinaus, müssen die Normalen streng nach Außen oder nach Innen bezüglich des Raums ausgerichtet sein, der von der zu rekonstruierenden Oberfläche umschlossen wird. Im Gegensatz zum *'greedy'*-Algorithmus baut der *'implicit'*-Algorithmus die Oberfläche **nicht** durch die Punkte des Eingabeobjekts `ObjectModel3D`. Stattdessen erstellt der Algorithmus eine Oberfläche, welche die Punkte approximiert und generiert einen neuen Satz von Punkten, die auf dieser Oberfläche liegen.

Als erstes organisiert der Algorithmus die Punkte in einer adaptiven Octree-Struktur: der Raum der kleinsten umschließenden Quader um die Punkte wird in der Mitte seiner drei Dimensionen in acht Sub-Räume, oder Voxels aufgespalten. Die Voxels, welche immer noch große Punktmenge enthalten, können weiter in acht Subvoxels aufgespalten werden. Die Voxel, die keine oder zu wenig Punkte enthalten, müssen nicht unbedingt weiter aufgespalten werden. Das Spalten entwickelt sich solange rekursiv in Regionen des Raums wo die Punktdaten dichter sind weiter, bis die am feinsten aufgespaltenen Voxels keine oder nur wenige Punkte enthalten. Die Rekursionsebene, welche von den kleinsten Voxels erreicht wird, wird als *Octree-Tiefe* bezeichnet.

Als nächstes berechnet der Algorithmus die Werte einer sogenannten *impliziten Indikatorfunktion* der Oberfläche. Die Berechnung basiert auf der Annahme, dass die 3D-Punkte des Modells `ObjectModel3D` auf der Oberfläche eines Körpers liegen und dass die Normalen des Modells bezüglich des Körpers streng nach Außen oder Innen gerichtet sind (siehe die Referenzen). Diese Annahme erklärt auch die Vorgabe, dass die Normalen unbedingt konsistent untereinander sein müssen. Die implizite Funktion hat einen Wert von 1 in Octree-Voxelecken, welche streng innerhalb des Körpers liegen und 0 - wenn sie streng außerhalb des Körpers liegen. Wegen Rauschen kann das für manche Voxelecken nicht eindeutig bestimmt werden und sie bekommen einen Wert zwischen 0 und 1.

Eine *implizite Oberfläche*, welche durch die implizite Funktion definiert wird, verläuft so, dass alle ihrer Punkte einen Indikatorwert von 0.5 haben. Die *'implicit'*-Triangulierung verwendet am Ende einen üblichen *Maching Cubes Algorithmus*, um die Schnittpunkte zwischen der impliziten Oberfläche und den Seiten der Octree-Voxels zu bestimmen. Die Schnittpunkte werden als neuer 3D-Punktsatz in `TriangulatedObjectModel3D` zurückgegeben. Als Folge davon wird die Granularität der rekonstruierten Oberfläche im Wesentlichen durch die Auflösung des Octrees bestimmt (d.h., durch seine *Tiefe*).

Mit `GenParamName` und `GenParamValue` können die folgenden zusätzlichen Parameternamen und -werte des *'implicit'*-Algorithmus eingestellt werden:

'implicit_octree_depth': setzt die Tiefe des Octrees. Die Octree-Tiefe bestimmt die Granularität der rekonstruierten Oberfläche - je höher der Tiefenwert ist, desto feiner werden Oberflächendetails rekonstruiert. Die Tiefe hat allerdings einen exponenziellen Einfluss auf den Speicherverbrauch des Verfahrens. Deswegen ist die Octree-Tiefe auf *12* begrenzt.

Restriktion: 5 $\leq$ 'implicit_octree_depth' $\leq$ 12.

Wertevorschläge: 5, 6, 8, 10, 11, 12.

Default: 6.

'implicit_solver_depth': aktiviert ein alternatives Verfahren, welches die implizite Funktion bis zu einer benutzerdefinierten Tiefe vorbereitet. Danach übernimmt das Standardverfahren die weitere Berechnungen. Dieser Algorithmus braucht weniger Speicher als der Original, dafür ist er aber etwas langsamer.

`Einschränkung`: *'implicit_solver_depth' $\leq$ 'implicit_octree_depth'.*

Wertevorschläge: 2, 4, 6, 8, 10, 11, 12.

Default: 6.

'implicit_min_num_samples': setzt die minimale Anzahl der Punkte pro Octree-Voxel. Wenn die Anzahl der Punkte in einem Voxel kleiner als dieser Wert ist, wird das Voxel nicht weiter aufgespalten. Für rauschfreie Daten kann dieser Wert niedriger gesetzt werden (z.B. zwischen 1 und 5), was eine feinere Auflösung des Octrees erlaubt. Für verrauschten Daten soll dieser Wert höher gestellt werden (z.B. 10-20, oder sogar noch höher), um mehrere verrauschten Punkte in einzelnen Voxeln zu sammeln und damit den Einfluss des Rauschens zu minimieren.

Wertevorschläge: 1, 5, 10, 15, 20, 30.

Default: 1.

'information': legt fest, welche Informationen in `Information` zurückgegeben werden. Wenn *'information'='num_triangles'* (default), dann wird nur die Anzahl der generierten Dreiecke zurückgeliefert. Wenn *'information'='verbose'* wird eine Liste von Name-Wert Paaren zurückgeliefert. Zur Zeit enthalten die Paare die folgenden Informationen:

Name	Wert	Beschreibung
'num_triangles'	<Anzahl der Dreiecke>	gibt die Anzahl der generierten dreieckigen Flächen zurück.
'num_points'	<Anzahl der Punkte>	gibt die Anzahl der neu generierten 3D-Punkte zurück.

Werteliste: 'num_triangles', 'verbose'.
Default: 'num_triangles'.

Vergleich der Triangulierungsmethode

In diesem Paragraph wird ein einfacher Vergleich beider unterstützter Triangulierungsverfahren angeboten:

Merkmal	Greedy-Triangulierung	Implicit-Triangulierung
benötigte Daten:	3D-Punkte mit 3D-Normale	3D-Punkte mit 3D-Normale, konsistent nach Außen oder Innen ausgerichtet
resultierende Oberfläche:	offen, Triangulierung der Eingabepunkte	geschlossen (wasserdicht), Approximation der Eingabepunkte
resultierende Punkte:	die Eingabepunkte bleiben	neue Punkte werden generiert
Behandlung des Rauschens:	moderate Punkt- und Normalenrauschen werden adäquat behandelt	Punkt- und Normalenrauschen werden implizit behandelt; moderate and hohe Rauschgrade werden akzeptiert
Auflösung der Triangulierung:	explizit, gesteuert durch die SNC-Parameter	implizit, gesteuert durch die Octree-Tiefe und die minimale Anzahl von Punkten pro Voxel
Geschwindigkeit:	$O(Nk \log N)$	$O(ND^3)$
Speicherverbrauch:	$O(Nk)$, mit Nachbarschaftscaches $O(N)$, ohne Nachbarschaftscaches	$O(D^3)$
		wobei: N: *Anzahl von Punkte* k: *Anzahl der betrachteten Nachbarn* D: *Tiefe des Octrees*

Je nach Anzahl von Punkten in *'ObjectModel3D'*, Rauschgrad und spezifischer Struktur der Eingabedaten, liefern beide Triangulierungsverfahren unterschiedliche Ergebnisse und werden mit unterschiedlicher Geschwindigkeit und Speicherverbrauch ausgeführt. Der Greedy-Algorithmus arbeitet schnell, beansprucht wenig Speicher und liefert eine Oberfläche mit großer Detailauflösung zurück wenn die Punktmenge der Eingabe vergleichsweise klein ist (z.B. bis zu 500.000 Punkte). Dadurch, dass der Algorithmus grundsätzlich jeden Punkt anfassen muss, kann seine Geschwindigkeit von der Anzahl der Punkte nicht abgekoppelt werden und der Algorithmus beansprucht mehr Zeit, wenn größere Punktmengen zu behandeln sind. Sollten jedoch solche trotzdem mit dem Greedy-Verfahren trianguliert werden, empfiehlt es sich, sie zuerst mit dem Operator `sample_object_model_3d` herunterzusampeln.

Im Gegensatz dazu organisiert wie bereits früher erklärt der Implicit-Algorithm die Punktdaten in Octree. Demzufolge werden die Auflösung der Triangulierung, die Geschwindigkeit und der Speicherverbrauch des Algorithmus im Grunde genommen von der Octree-Tiefe bestimmt. Der Nachteil davon ist, dass eine höhere Auflösung der Oberfläche nur auf Kosten von unverhältnismäßig höherer Geschwindigkeit und höherem Speicherverbrauch erzielt werden kann. Der Vorteil der Octree-basierte Triangulierung ist jedoch, dass große Punktmengen effektiv im

Octree organisiert werden können (insbesondere bei niedrigen Werten der Octree-Tiefe) und die Kosten der Rekonstruktion hauptsächlich von der Octree-Tiefe und weniger von der Punktmenge abhängig gemacht werden kann. Ein erheblicher Nachteil des Implicit-Algorithmus ist die Forderung nach konsistenten benachbarten Normalen, da diese selten oder nur schwierig von typischen Routinen zur Normalenberechnung zurückgeliefert werden.

Parameter

- ▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle eines 3D-Objektmodells mit 3D-Punktdaten.
- ▷ **Method** (input_control) . . . . . string ⇝ *string*
  Triangulierungsmethode.
  Defaultwert: 'greedy'
  Werteliste: `Method` ∈ {'greedy', 'implicit', 'polygon_triangulation', 'xyz_mapping'}
- ▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'information', 'implicit_octree_depth', 'implicit_solver_depth', 'implicit_min_num_samples', 'greedy_radius_type', 'greedy_radius_value', 'greedy_kNN', 'greedy_neigh_orient_tol', 'greedy_neigh_orient_consistent', 'greedy_neigh_vertical_tol', 'greedy_neigh_latitude_tol', 'greedy_hole_filling', 'greedy_fix_flips', 'greedy_mesh_erosion', 'greedy_mesh_dilation', 'greedy_remove_small_surfaces', 'greedy_prefetch_neighbors', 'greedy_timeout', 'greedy_suppress_timeout_error', 'greedy_output_all_points', 'xyz_mapping_max_area_holes', 'xyz_mapping_output_all_points', 'xyz_mapping_max_view_angle', 'xyz_mapping_max_view_dir_x', 'xyz_mapping_max_view_dir_y', 'xyz_mapping_max_view_dir_z'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
  Werte der generischen Parameter.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {6, 8, 12, 'true', 'false', 'auto', 'fixed', 'z_factor', 'verbose', 'num_triangles'}
- ▷ **TriangulatedObjectModel3D** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle des triangulierten 3D-Objektmodells.
- ▷ **Information** (output_control) . . . . . number(-array) ⇝ *integer* / string
  Zusätzliche Information über der Triangulierung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_object_model_3d`, `gen_plane_object_model_3d`, `gen_sphere_object_model_3d`, `gen_cylinder_object_model_3d`, `gen_box_object_model_3d`, `gen_sphere_object_model_3d_center`, `sample_object_model_3d`

Nachfolger

`write_object_model_3d`, `render_object_model_3d`, `project_object_model_3d`, `simplify_object_model_3d`

Literatur

M. Kazhdan, M. Bolitho, and H. Hoppe: „Poisson Surface Reconstruction.“ Symposium on Geometry Processing (June 2006).

Modul

3D Metrology

xyz_to_object_model_3d (X, Y, Z : : : ObjectModel3D)

Transformieren von 3D-Punkten aus Bildern in ein 3D-Objektmodell.

xyz_to_object_model_3d transformiert ein Bildtripel, welches die X-, Y- und Z-Koordinaten von 3D-Punkten enthält, in ein 3D-Objektmodell. Es werden nur Punkte der Schnittmenge der Domänen aller drei Bilder in das 3D-Objektmodell übernommen. Die drei Bilder müssen die selbe Größe haben. Die Größe dieser Originalbilder kann vom Modell über get_object_model_3d_params mit *'mapping_size'* abgefragt werden. Das Handle des erstellten 3D-Objektmodells wird in ObjectModel3D zurückgegeben. Das erstellte 3D-Objektmodell enthält die 3D-Koordinaten der Punkte, sowie eine Abbildung, welche für jeden der 3D-Punkte dessen ursprüngliche Bildkoordinate (Zeile und Spalte) enthält. Punkte bei denen eine der Koordinaten unendlich oder eine nicht-normalisierte Zahl ist (NaN) werden ignoriert und nicht zum 3D-Objektmodell hinzugefügt.

Parameter

- ▷ **X** (input_object) ..... singlechannelimage ⇝ *object :* real
  Bild mit den X-Koordinaten und der ROI der 3D-Punkte.
- ▷ **Y** (input_object) ..... singlechannelimage ⇝ *object :* real
  Bild mit den Y-Koordinaten der 3D-Punkte.
- ▷ **Z** (input_object) ..... singlechannelimage ⇝ *object :* real
  Bild mit den Z-Koordinaten der 3D-Punkte.
- ▷ **ObjectModel3D** (output_control) ..... object_model_3d ⇝ *handle*
  Handle des erstellten 3D-Objektmodells.

Ergebnis

xyz_to_object_model_3d liefert den Wert TRUE wenn alle Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

disparity_image_to_xyz, get_sheet_of_light_result

Alternativen

gen_object_model_3d_from_points, get_sheet_of_light_result_object_model_3d

Siehe auch

read_object_model_3d

Modul

3D Metrology

Kapitel 5

3D-Rekonstruktion

5.1 Binokulares Stereo

```
binocular_disparity ( ImageRect1, ImageRect2 : Disparity,
    Score : Method, MaskWidth, MaskHeight, TextureThresh,
    MinDisparity, MaxDisparity, NumLevels, ScoreThresh, Filter,
    SubDisparity : )
```

Berechnet die Disparität für ein rektifiziertes Bildpaar mit Hilfe von Korrelationsmethoden.

`binocular_disparity` berechnet mit Hilfe von Korrelationsmethoden pixelweise Korrespondenzen zwischen zwei rektifizierten Bildern. Im Unterschied zu `binocular_distance` werden die Ergebnisse nicht in Distanzwerte transformiert.

Der Algorithmus benötigt ein Referenzbild `ImageRect1` und ein Suchbild `ImageRect2`, welche entzerrt sein müssen, d.h. korrespondierende Epipolarlinien liegen parallel und haben identische Bildzeilenkoordinaten ($r_1 = r_2$). Falls diese Annahme nicht zutrifft, können die Bilder durch die Operatoren `calibrate_cameras`, `gen_binocular_rectification_map` und `map_image` entzerrt werden. Ausgehend von einem Pixel im Referenzbild `ImageRect1` kann dann das zugehörige Pixel in `ImageRect2` durch die Suche entlang der korrespondierenden Zeile in `ImageRect2` gefunden werden. Eine Übereinstimmung wird gefunden, indem jeweils eine lokale Nachbarschaft um das entsprechende Pixel innerhalb eines Fensters der Größe `MaskWidth` und `MaskHeight` in beiden Bildern miteinander verglichen wird. Die Pixelkorrespondenzen werden in dem einkanaligen Bild `Disparity` zurückgegeben. Der Grauwert $d(r_1, c_1)$ beschreibt dabei die Disparität zwischen dem Pixel (r1,c1) im Referenzbild `ImageRect1` und dem korrespondierendem Pixel (r2,c2) im Suchbild `ImageRect2` und erfüllt somit die Gleichung $c_2 = c_1 + d(r_1, c_1)$. Ein Qualitätsmaß für die ermittelten Disparitätswerte wird im Bild `Score` zurückgegeben, welches jeweils das beste Ergebnis der Korrelationsfunktion S für ein Referenzpixel enthält. Für die Korrelation werden die Grauwerte der originalen, nicht vorverarbeiteten Eingabebilder genutzt.

Die verwendete Korrelationsfunktion wird durch den Parameter `Method` bestimmt, welcher drei Berechnungsmethoden unterscheidet:

- *'sad'*: Summed Absolute Differences
  $S(r,c,d) = \frac{1}{N} \sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n} \mid g_1(r',c') - g_2(r',c'+d) \mid,$
  with $0 \leq S(r,c,d) \leq 255$.
- *'ssd'*: Summed Squared Differences
  $S(r,c,d) = \frac{1}{N} \sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n} (g_1(r',c') - g_2(r',c'+d))^2,$
  with $0 \leq S(r,c,d) \leq 65025$.
- *'nmc'*: Normalized Cross Correlation
  $S(r,c,d) = \frac{\sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n} (g_1(r',c') - \bar{g_1}(r,c))(g_2(r',c'+d) - \bar{g_2}(r,c))}{\sqrt{\sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n+d} (g_1(r',c') - \bar{g_1}(r,c))^2 (g_2(r',c'+d) - \bar{g_2}(r,c+d))^2}},$
  with $-1.0 \leq S(r,c,d) \leq 1.0$.

wobei
$r1, c1, r2, c2$: Zeilen- und Spaltenkoordinaten der korrespondierenden Pixel der beiden Eingabebilder,
$g1, g2$: Grauwerte der nicht vorverarbeiteten Eingabebilder,
$N = (2m+1)(2n+1)$: Größe des Korrelationsfensters,
$\bar{g}(r,c) = \frac{1}{N} \sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n} g(r',c')$: mittlerer Grauwert innerhalb eines Korrelationsfensters der Breite $2m+1$ und Höhe $2n+1$.

Die Methoden *'sad'* und *'ssd'* vergleichen direkt die Grauwerte der Pixel innerhalb des Grauwertfensters, wohingegen *'ncc'* den mittleren Grauwert und seine Varianz innerhalb des Grauwertfensters ausgleicht. Wenn zwei Bilder in ihrer Helligkeit und ihrem Kontrast unterschiedlich sind, ist daher *'ncc'* vorzuziehen. Ansonsten sind *'sad'* und *'ssd'* vorzuziehen, da sie aufgrund einfacherer interner Berechnungen schneller sind.

Es muss darauf geachtet werden, dass die Güte der Korrelation für steigende S in den Methoden *'sad'* und *'ssd'* fällt (die beste Güte ist 0), während sie in der Methode *'ncc'* steigt (die beste Güte ist 1.0).

Die Größe des Korrelationsfensters, welche durch $2m+1$ und $2n+1$ in obigen Gleichungen referenziert wird, muss ungeradzahlig sein und kann durch die Parameter `MaskWidth` und `MaskHeight` festgelegt werden. Der Suchraum wird durch den minimalen und maximalen Disparitätswert `MinDisparity` und `MaxDisparity` festgelegt. Da Grauwerte nicht außerhalb des Bildrandes definiert sind wird die Bildregion des `Disparity` und `Score` Bildes am unteren und oberen Bildrand um (`MaskHeight`-1)/2 sowie am linken und rechten Rand um (`MaskWidth`-1)/2 beschnitten. Aus dem gleichen Grund wird der maximale Suchraum am linken und rechten Bildrand begrenzt.

Da die Korrelationsverfahren in schwach texturierten Bildregionen im hohen Maße ungenau werden, kann die kleinste, innerhalb eines Korrelationsfensters erlaubte Grauwertstreuung mit Hilfe des Parameters `TextureThresh` definiert werden. Dieser Schwellenwert wird auf beide Eingabebilder `ImageRect1` und `ImageRect2` angewendet. Zusätzlich garantiert `ScoreThresh` eine Mindestgüte der verwendeten Korrelationsfunktion und legt für die Methoden *'sad'* und *'ssd'* den größten, bzw. für *'ncc'* den kleinsten zulässigen Korrelationswert fest. Wird ferner `Filter` auf *'left_right_check'* gesetzt, werden nur noch diejenigen Disparitätswerte zurückgegeben, die übereinstimmend bei einer Suche eines Referenzmusters von `ImageRect1` in `ImageRect2` als auch von `ImageRect2` in `ImageRect1` gefunden werden. Der Wert *'none'* hingegen schaltet dieses Verhalten aus.

Die Anzahl der Pyramidenstufen, welche verwendet werden um das Zeitverhalten von `binocular_disparity` zu verbessern, wird durch den Parameter `NumLevels` bestimmt. Einem grob-zu-fein Schema folgend werden die Disparitätsbilder der höheren Stufen berechnet und in rechteckige Bildteile von ähnlicher Disparität unterteilt um den Suchbereich auf der nächst niedrigeren Stufe für diesen Bildausschnitt einzuschränken. `TextureThresh` und `ScoreThresh` werden auf jeder Stufe angewendet, so dass die mit `Disparity` und `Score` zurückgegebene Bildregion als die Schnittmenge der Bildregionen auf jeder einzelnen Stufe definiert ist. Im allgemeinen lohnt sich das Verwenden von Pyramidenstrukturen um so mehr, je mehr das Disparitätsbild in homogene Bereiche unterteilt werden kann und je größer der Disparitätsbereich definiert werden muss. Auf der anderen Seite können zu hohe Pyramidenstufen durch ihre grobe Auflösung wichtige Texturinformationen verlieren, so dass falsche Disparitätswerte zurückgegeben werden können.

Der Wert *'interpolation'* für den Parameter `SubDisparity` schließlich berechnet die Disparitäten Subpixel genau. Das Verhalten wird ausgeschaltet indem der Parameter auf *'none'* gesetzt wird.

Parameter

- ▷ **`ImageRect1`** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Rektifiziertes Bild der Kamera 1.
- ▷ **`ImageRect2`** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Rektifiziertes Bild der Kamera 2.
- ▷ **`Disparity`** (output_object) ..... singlechannelimage ⇝ *object :* real
  Disparität.
- ▷ **`Score`** (output_object) ..... singlechannelimage ⇝ *object :* real
  Maß für die Zuverlässigkeit eines Disparitätswertes.
- ▷ **`Method`** (input_control) ..... string ⇝ *string*
  Korrespondenzverfahren.
  Defaultwert: 'ncc'
  Werteliste: `Method` ∈ {'sad', 'ssd', 'ncc'}

▷ **MaskWidth** (input_control) ..... integer ⇝ *integer*
Breite des Korrelationsfensters.
Defaultwert: 11
Wertevorschläge: `MaskWidth` ∈ {5, 7, 9, 11, 21}
Restriktion: 3 <= MaskWidth && odd(MaskWidth)

▷ **MaskHeight** (input_control) ..... integer ⇝ *integer*
Höhe des Korrelationsfensters.
Defaultwert: 11
Wertevorschläge: `MaskHeight` ∈ {5, 7, 9, 11, 21}
Restriktion: 3 <= MaskHeight && odd(MaskHeight)

▷ **TextureThresh** (input_control) ..... real ⇝ *real* / integer
Schwellenwert der Grauwertvarianz.
Defaultwert: 0.0
Wertevorschläge: `TextureThresh` ∈ {0.0, 10.0, 30.0}
Restriktion: 0.0 <= TextureThresh

▷ **MinDisparity** (input_control) ..... integer ⇝ *integer*
Minimale erwartete Disparität.
Defaultwert: -30
Typischer Wertebereich: -32768 ≤ `MinDisparity` ≤ 32767

▷ **MaxDisparity** (input_control) ..... integer ⇝ *integer*
Maximale erwartete Disparität.
Defaultwert: 30
Typischer Wertebereich: -32768 ≤ `MaxDisparity` ≤ 32767

▷ **NumLevels** (input_control) ..... integer ⇝ *integer*
Anzahl der Pyramidenstufen.
Defaultwert: 1
Wertevorschläge: `NumLevels` ∈ {1, 2, 3, 4}
Restriktion: 1 <= NumLevels

▷ **ScoreThresh** (input_control) ..... real ⇝ *real* / integer
Schwellenwert der Korrelation.
Defaultwert: 0.5
Wertevorschläge: `ScoreThresh` ∈ {-1.0, 0.0, 0.3, 0.5, 0.7}

▷ **Filter** (input_control) ..... string(-array) ⇝ *string*
Nachgelagerte Ergebnisfilter.
Defaultwert: 'none'
Werteliste: `Filter` ∈ {'none', 'left_right_check'}

▷ **SubDisparity** (input_control) ..... string ⇝ *string*
Sup-Pixel Interpolation der Disparitäten.
Defaultwert: 'none'
Werteliste: `SubDisparity` ∈ {'none', 'interpolation'}

Beispiel

```
* Set internal and external stereo parameters.
* Note that, typically, these values are the result of a prior
* calibration.
gen_cam_par_area_scan_division (0.01, -665, 5.2e-006, 5.2e-006, \
                                622, 517, 1280, 1024, CamParam1)
gen_cam_par_area_scan_division (0.01, -731, 5.2e-006, 5.2e-006, \
                                654, 519, 1280, 1024, CamParam2)
create_pose (0.1535,-0.0037,0.0447,0.17,319.84,359.89, \
             'Rp+T', 'gba', 'point', RelPose)

* Compute the mapping for rectified images.
gen_binocular_rectification_map (Map1, Map2, CamParam1, CamParam2, RelPose, \
  1, 'viewing_direction', 'bilinear', CamParamRect1,CamParamRect2, Cam1PoseRect1, \
  Cam2PoseRect2,RelPoseRect)
```

```
* Compute the disparities in online images.
while (1)
  grab_image_async (Image1, AcqHandle1, -1)
  map_image (Image1, Map1, ImageRect1)

  grab_image_async (Image2, AcqHandle2, -1)
  map_image (Image2, Map2, ImageRect2)

  binocular_disparity(ImageRect1, ImageRect2, Disparity, Score, 'sad', \
    11, 11, 20, -40, 20, 2, 25, 'left_right_check','interpolation')
endwhile
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `binocular_disparity` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`map_image`

Nachfolger

`threshold`, `disparity_to_distance`, `disparity_image_to_xyz`

Alternativen

`binocular_disparity_mg`, `binocular_disparity_ms`, `binocular_distance`, `binocular_distance_mg`, `binocular_distance_ms`

Siehe auch

`map_image`, `gen_binocular_rectification_map`, `binocular_calibration`

Modul

3D Metrology

binocular_disparity_mg (ImageRect1, ImageRect2 : Disparity, Score : GrayConstancy, GradientConstancy, Smoothness, InitialGuess, CalculateScore, MGParamName, MGParamValue :)

Berechnet die Disparität für ein rektifiziertes Bildpaar mit Hife von Mehrgitter-Verfahren.

`binocular_disparity_mg` berechnet die Disparität zwischen zwei rektifizierten Stereobildern `ImageRect1` und `ImageRect2` und gibt sie in `Disparity` zurück. Im Gegensatz zu `binocular_disparity` werden Variationsmethoden basierend auf Mehrgitter-Verfahren verwendet. Damit können Disparitätswerte auch für Bildbereiche bestimmt werden, die keine oder kaum Textur aufweisen. Im Gegensatz zu `binocular_distance_mg` wird das Ergebnis nicht in Abstandswerte umgerechnet.

Die Eingabebilder müssen ein Paar rektifizierter Stereobilder darstellen, d.h. korrespondierende Punkte müssen dieselbe vertikale Koordinate haben. Die beiden Bilder müssen die gleiche Höhe haben, können aber unterschiedliche Breiten besitzen. Die Laufzeit des Verfahrens ist bei sonst gleichen Parametern in etwa linear zur Fläche der beiden Bilder.

Die Disparität gibt für jeden Bildpunkt des ersten Bildes `ImageRect1` die Verschiebung an, die nötig ist um seinen korrespondierenden Punkt im zweiten Bild `ImageRect2` zu erreichen. Zwei Punkte des ersten und zweiten Bildes heißen dabei korrespondierend, wenn sie der Bildpunkt desselben Punktes aus der ursprünglichen Szene sind. Das berechnete Disparitätsfeld ist dicht und schätzt die Disparität auch für Punkte ohne korrespondierenden Punkt ab. Die Disparität wird nur für die Zeilen berechnet, die Teil der Definitionsbereiche beider Bilder sind. Genauer gesagt berechnet sich der Definitionsbereich des Disparitätsbildes aus der Schnittmenge der Höhen der kleinsten umschließenden Rechtecke der Definitionsbereiche der Eingabebilder.

Das berechnete Disparitätsfeld ist in der Regel nicht perfekt. Wenn `CalculateScore` auf *'true'* gesetzt ist, wird die Güte der Disparität für jedes Pixel geschätzt und in `Score` als Grauwertbild mit Grauwerten im Bereich 0-10 zurückgegeben, wobei 0 für die beste und 10 für die schlechteste Qualität steht. Für das Gütemaß wird zusätzlich die Disparität vom zweiten zum ersten Bild berechnet und anschließend bestimmt, inwieweit die beiden Disparitätsfelder übereinstimmen. Daher verdoppelt sich die Laufzeit des Operators beim Aktivieren dieses Schalters in etwa.

Der Operator verwendet einen Variationsansatz, bei welchem jedem möglichen Disparitätsfeld ein Energiewert zugeordnet wird. Je geringer dieser Energiewert ist, desto besser wird das Disparitätsfeld bewertet. Das Verfahren berechnet ein Disparitätsfeld mit minimaler Energie und liefert dieses zurück.

Die einem Disparitätsfeld zugeordnete Energie zerfällt in einen Datenanteil und einen Glättungsanteil. Der Datenanteil modelliert die Tatsache, dass die Bilder in Bereichen korrespondierender Punkte Abbildungen desselben Teils der Szene sind und daher dieselben Grauwerte aufweisen. Der Glättungsanteil modelliert, dass die abgebildete Szene und die zugehörige Disparität stückweise glatt ist, und führt zu einer Interpolation der Werte auch in Bereiche ohne Informationen aus dem Datenanteil, z.B. Bereiche ohne Textur.

Im Detail liegen dem Variationsansatz die folgenden Annahmen zu Grunde:

Konstanz des Grauwerts: Es wird davon ausgegangen, dass zwei korrespondierende Punkte denselben Grauwert haben, dass also $I_1(x,y) = I_2(x+u(x,y),y)$ gilt.

Konstanz des Grauwertgradienten: Es wird angenommen, dass die Bilder an den Stellen zweier korrespondierender Punkte denselben Grauwertgradienten aufweisen, dass also $\nabla I_1(x,y) = \nabla I_2(x+u(x,y),y))$ gilt. Abweichungen davon werden energetisch mittels der L_2-Norm der Differenz der beiden Gradienten bestraft. Die Verwendung des Gradienten hat im Vergleich zur Grauwertkonstanz den Vorteil, dass er unabhängig von additiven Helligkeitsänderungen zwischen den beiden Bildern ist.

Statistische Robustheit im Datenterm: Um den Einfluss von Ausreißern, also Punkten, die eine der beiden Erhaltungsannahmen verletzen, zu reduzieren, werden Abweichungen von den beiden obigen Erhaltungsannahmen mittels einer statistisch robusten Funktion, der totalen Variation, bestraft. Diese lautet $\Psi(x) = \sqrt{x+\epsilon^2}$. Die Konstante ϵ wird dabei auf 0.01 gesetzt.

Glattheit des Disparitätsfeldes: Es wird davon ausgegangen, dass das Disparitätsfeld stückweise stetig ist und sich lokal nur gering ändert. Energetisch wird dies über die L_2-Norm der Ableitung der Disparität modelliert.

Statistische Robustheit im Glättungsterm: Analog zum Datenterm wird auch im Glättungsterm die totale Variation verwendet. Aufgrund der statistischen Robustheit gegenüber Ausreißern hat dies hier den Vorteil, dass Sprünge in der Disparität, wie sie an Objektkanten auftreten, besser erhalten bleiben.

Das Energiefunktional ergibt sich durch Integration einer Linearkombination der einzelnen Terme über alle Bildpunkte des ersten Bildes. Die Koeffizienten der Linearkombination sind dabei Parameter des Operators und erlauben eine Abstimmung des Ansatzes auf konkrete Situationen. `GrayConstancy` bestimmt den Einfluss der Grauwerterhaltung, `GradientConstancy` den Einfluss der Gradientenerhaltung und `Smoothness` den Einfluss des Glättungsterms. Die beiden ersten Parameter müssen dabei an den Grauwertbereich des Bildes angepasst werden. Die vorgeschlagenen Standardparameter gelten für Bilder mit Grauwerten von 0 bis 255.

Es sei $I_1(x,y)$ das Grauwert des ersten Bildes am Punkt (x,y), $I_2(x,y)$ der Grauwert des zweiten Bildes und $u(x,y)$ der Wert der Disparität an der Stelle (x,y). Das Energiefunktional ist dann:

$$
\begin{aligned}
E = \int \Psi \quad \Big(\quad & \texttt{GrayConstancy} \cdot \underbrace{(I_2(x+u(x,y),y) - I_1(x,y))^2}_{\text{Grauwerterhaltung}} + \\
& \texttt{GradientConstancy} \cdot \underbrace{|\nabla I_2(x+u(x,y)) - \nabla I_1(x,y)|^2}_{\text{Gradientenerhaltung}} \Big) \\
+ \quad & \texttt{Smoothness} \cdot \Psi \underbrace{\left(|\nabla u(x,y)|^2\right)}_{\text{Glattheit}} \, dx\, dy
\end{aligned}
$$

Das Verfahren geht davon aus, dass das Disparitätsfeld u, welches dieses Funktional E minimiert, die obigen Annahmen am besten erfüllt, und daher eine gute Näherung der Disparität zwischen den beiden Bildern darstellt.

Um das obige Funktional zu minimieren, wird die sogenannte Euler-Lagrange-Gleichung (ELG) des Integrals gelöst. Diese ist vergleichbar mit der Ableitung eindimensionaler Funktionen, welche an Extrema 0 ist. Die ELG ist

eine nichtlineare partielle Differentialgleichung auf dem Gebiet des Integrals und muss, analog zum eindimensionalen Fall, an Stellen von Extrema den Wert 0 annehmen. Da das Funktional in der Regel keine Maxima aufweist, entsprechen Nullstellen der ELG den gesuchten Minima des Funktionals.

Zur Bestimmung der Nullstellen der ELG werden folgende Techniken verwendet:

Fixpunkt-Iteration: Zum Lösen der ELG wird eine Fixpunkt-Iteration angewendet. Dies ist ein iteratives Verfahren, welches die Lösung schrittweise annähert, und in jedem Schritt eine Verbesserung der aktuellen Lösung berechnet. Über die Anzahl der Schritte kann daher zwischen Qualität und Geschwindigkeit abgewogen werden. Für jeden Schritt wird eine lineare partielle Differentialgleichung gelöst.

Grob-zu-Fein Verfahren: Von den beiden Stereobildern wird eine Gausssche Bildpyramide erstellt. Die ELG wird dabei zunächst für gröbere Versionen der Bilder gelöst, und die Lösung als Startwert der Fixpunkt-Iteration für die nächst feinere Version verwendet. Dies hat eine Reihe von Vor- und Nachteilen:

1. Da die Fixpunkt-Iteration von der nächstgröberen Ebene der Bildpyramide einen guten Startwert bekommt, sind weniger Iterationen nötig, wodurch das Verfahren schneller wird. Die Iteration muss nur noch kleine Korrekturen an der Disparität vornehmen.

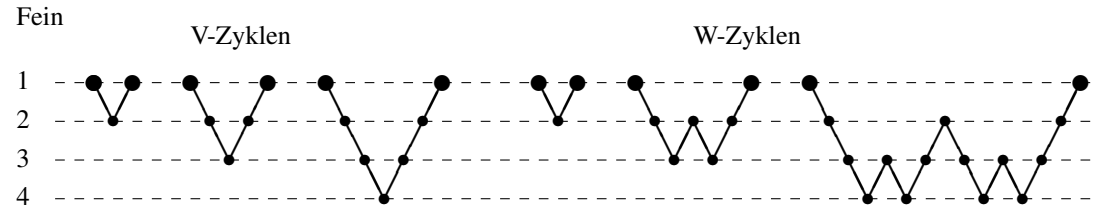
2. Große Disparitäten auf den ursprünglichen Bildern werden auf den gröberen Bildversionen kleiner und können daher einfacher bestimmt werden.

3. Die Robustheit gegen Bildstörungen wird erhöht, da diese auf den groben Bildversionen verschwinden und auf den feinen Bildversionen lediglich kleine Korrekturen berechnet werden.

4. Probleme entstehen bei kleinen Strukturen mit großem Disparitätsunterschied zur Umgebung, da diese auf den gröberen Versionen der Bilder verschwinden und daher die Disparität der Umgebung berechnet wird. Da auf den feineren Versionen der Pyramide lediglich kleine Korrekturen berechnet werden, kann dieser Fehler nicht mehr korrigiert werden.

Mehrgitterverfahren: Die bei der Fixpunkt-Iteration entstehenden linearen partiellen Differentialgleichungen werden durch Linearisierung in ein lineares Gleichungssystem umgewandelt. Für die Lösung dieses Systems werden wiederum iterative Verfahren eingesetzt. Mehrgitterverfahren gehören bei den hier auftretenden Gleichungssystemen zu den effizientesten Lösungsverfahren. Sie nutzen die Tatsache, dass klassische Löser, etwa das Gauß-Seidel-Verfahren, hochfrequente Anteile des Fehlers schnell dämpfen, niederfrequente Anteile jedoch nur langsam. Mehrgitterverfahren berechnen den Fehler daher auf einem gröberen Gitter, auf welchem diese niederfrequenten Anteile des Fehlers eine höhere Frequenz bekommen und von den Lösern schneller gedämpft werden. Diese Eigenschaft wird von Mehrgitterverfahren rekursiv genutzt. Für die Berechnung des Fehlers auf der gröberen Version des Gleichungssystems wird also eine noch gröbere Version des Systems hinzugezogen. Je nach Anzahl der Korrekturschritte ergeben sich dadurch V- oder W-Zyklen als Aufrufschemata, welche im folgenden für zwei bis vier Auflösungsstufen dargestellt sind:

Bidirektionales Mehrgitterverfahren



Hierbei werden Iterationen auf dem Originalproblem mit großen Markierungen symbolisiert, während kleine Markierungen für Iterationen auf Fehlerkorrekturproblemen stehen.

Algorithmisch lässt sich ein solcher Korrekturzyklus wie folgt beschreiben:

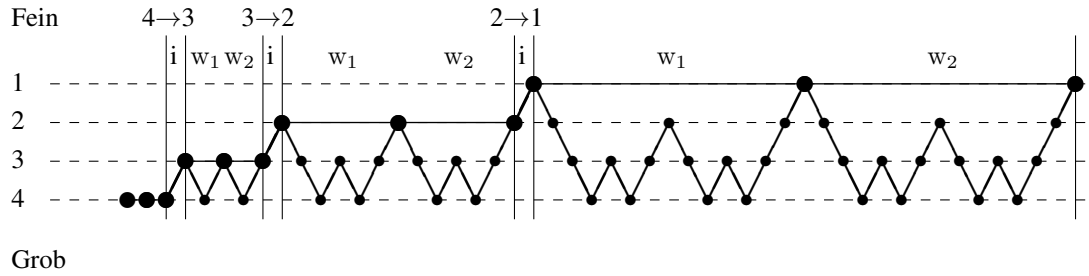
1. In einem ersten Schritt werden einige wenige Iterationen mit einem iterativen linearen oder nichtlinearen Basislöser (z.B. einer Gauß-Seidel-Variante) gemacht. Dieser Schritt nennt sich Vorglättung.

2. In einem zweiten Schritt wird der aktuelle Fehler berechnet, um die bisherige Lösung (die Lösung nach Schritt 1) zu korrigieren. Aus Effizienzgründen findet diese Fehlerberechnung auf einer gröberen Auflösungsstufe statt. Dieser Schritt, der mehrfach wiederholt werden kann, nennt sich Grobgitterkorrekturschritt.

3. In einem letzten Schritt werden erneut einige wenige Iterationen mit dem gleichen iterativen linearen oder nichtlinearen Basislöser wie in Schritt 1 gemacht. Dieser Schritt nennt sich Nachglättung.

Weiterhin kann der Startwert der Iteration mittels eines hierarchischen Verfahrens bestimmt werden. Hierbei wird die Lösung vom gröbsten Gitter ausgehend immer weiter verbessert. Die skalierte Lösung auf einem gröberen Gitter dient dabei als Startwert für das nächst feinere Gitter. Auf jeder Gitterstufe wird das lineare Gleichungssystem durch einen der oben beschriebenen V- oder W-Zyklen gelöst. Das entsprechende Mehrgitterverfahren wird volles Mehrgitterverfahren genannt und kann wie folgt visualisiert werden:

Volles Mehrgitterverfahren



Dieses Beispiel stellt ein volles Mehrgitterverfahren mit zwei W-Korrekturzyklen pro Auflösungsstufe der hierarchischen Initialisierung dar. Die Interpolationsschritte der Lösung von einer Stufe zur nächsten sind mit i bezeichnet, die beiden W-Korrekturzyklen mit w_1 und w_2. Während Iterationen auf dem Originalproblem mit großen Markierungen symbolisiert werden, stehen kleine Markierungen für Iterationen auf Fehlerkorrekturproblemen.

Je nach gewähltem Mehrgitterverfahren stehen eine Reihe von Parametern zu Verfügung, welche im Folgenden genauer beschrieben sind.

Der Parameter `InitialGuess` gibt den Initialisierungswert für die Fixpunkt-Iteration an, also den ersten Schätzwert. In der Regel ist hier 0 ausreichend. Läuft das Verfahren jedoch in lokale Minima, so kann hier ein anderer Wert übergeben werden.

Mittels der Parameter `MGParamName` und `MGParamValue` wird das Lösungsverfahren, also das Grob-zu-Fein-Verfahren, die Fixpunkt-Iteration und das Mehrgitterverfahren gesteuert. In der Regel reicht es aus, eine der vordefinierten und aufeinander abgestimmten Parametergruppen zu verwenden. Hierfür wird `MGParamName` = *'default_parameters'* und `MGParamValue` = *'very_accurate'*, *'accurate'*, *'fast_accurate'* oder *'fast'* gesetzt.

Sollen die Parameter einzeln gesteuert werden, so müssen in `MGParamName` und `MGParamValue` zwei gleichlange Tupel übergeben werden. Die Einträge in `MGParamName` bestimmen dabei jeweils den zu setzenden Parameter, die Einträge in `MGParamValue` den zu setzenden Wert. Die Parameter werden in der Reihenfolge ausgewertet, in welcher sie übergeben werden. Es kann also auch zunächst eine Gruppe von Standardparametern gewählt werden (s.o.) und danach gezielt einzelne Parameter verändert werden. Im Folgenden werden die möglichen zu setzenden Parameter beschrieben.

`MGParamName` = *'mg_solver'* wählt das Lösungsverfahren für die linearen Gleichungssysteme aus. Mögliche Werte für `MGParamValue` sind *'multigrid'* für ein einfaches Mehrgitterverfahren, *'full_multigrid'* für ein volles Mehrgitterverfahren sowie *'gauss_seidel'* für ein Gauß-Seidel-Verfahren. Die Mehrgitterverfahren haben den Vorteil einer schnelleren Konvergenz, benötigen jedoch einen zusätzlichen Aufwand für das Vergröbern der linearen Gleichungssysteme.

`MGParamName` = *'mg_cycle_type'* wählt den Rekursionstyp des Mehrgitterverfahrens aus. Mögliche Werte für `MGParamValue` sind *'v'* für einen V-Zyklus, *'w'* für einen W-Zyklus und *'none'* für keine Rekursion.

`MGParamName` = *'mg_pre_relax'* setzt für Mehrgitterverfahren die Anzahl der Gauß-Seidel-Iterationen beim Vorglätten, bzw. beim Gauß-Seidel-Verfahren die Anzahl an Iterationen beim Lösen des Gleichungssystems. Der Wert muss eine Zahl größer 0 sein.

`MGParamName` = *'mg_post_relax'* setzt für Mehrgitterverfahren die Anzahl der Gauß-Seidel-Iterationen beim Nachglätten. Der Wert muss eine Zahl größer 0 sein.

Beide gerade beschriebene Parameter gehen asymptotisch gesehen linear in die Laufzeit des Verfahrens ein. Da viele andere Berechnungen jedoch unabhängig von der Anzahl an Gauß-Seidel-Iterationen nur einmal nötig sind, führen kleine Änderungen der beiden Parameter nur zu kleinen Änderungen in der Laufzeit.

`MGParamName` = *'initial_level'* legt die gröbste zu verwendende Stufe der Bildpyramide beim Grob-zu-Fein-Verfahren fest. Dies kann entweder durch Angabe eines positiven Wertes erfolgen, welcher die gröbste Ebene angibt. Ebene 0 ist dabei die feinste Ebene mit den ursprünglichen Bildern. Alternativ kann ein negativer Wert angegeben werden, welcher die gröbste zu verwendende Stufe relativ zur gröbsten verfügbaren Stufe angibt. Die

gröbste verfügbare Stufe ist diejenige, auf welcher beide Bilder in beide Koordinatenrichtungen eine Größe von mindestens 4 Pixeln aufweisen. Wie unten angegeben, ist der Standardwert von *'initial_level'* *-2*. Dies erlaubt die Bestimmung der korrekten Disparität auch für Bilder, die sehr hohe Disparitäten aufweisen. In Einzelfällen, z.B. bei sich wiederholenden Texturen, kann dies allerdings dazu führen, dass zu große Disparitäten für Teile des Bildes berechnet werden. In diesem Fall sollte *'initial_level'* auf einen niedrigeren Wert gesetzt werden.

Bei Standardparametern werden die Bilder pro Pyramidenstufe um den Faktor 0.6 skaliert. Liegt eine Schätzung d der maximalen Disparität vor, so sollte die erste Ebene s so gewählt werden, dass 0.6^{-s} größer als d ist.

`MGParamName` = *'iterations'* legt die Anzahl an Fixpunkt-Iterationen pro Ebene fest. Die genaue Anzahl an Iterationen bestimmt sich durch $steps = \min(10, \text{iterations} + \text{level}^2)$, wobei level die aktuelle Ebene in der Bildpyramide bezeichnet. Wird dieser Parameter gleich 0 gesetzt, so wird auf der Bildpyramidenebene 0 der ursprünglichen Bilder keine Iteration durchgeführt und als Ergebnis das skalierte Resultat der Ebene 1 zurückgeliefert. Dies eignet sich für besonders schnelle Verfahren. Die Anzahl an Iterationen geht annähernd linear in die Laufzeit des Verfahrens ein.

`MGParamName` = *'pyramid_factor'* legt den Skalierungsfaktor für die Bildpyramide des Grob-zu-Fein-Verfahrens fest. Die Höhe und Breite des nächst kleineren Bildes wird dabei um den angegebenen Faktor skaliert. Der Wert muss zwischen 0.1 und 0.9 liegen.

Die vordefinierten Parametergruppen für `MGParamName` = *'default_parameters'* enthalten folgende Werte:

'default_parameters' = *'very_accurate'*: *'mg_solver'* = *'full_multigrid'*, *'mg_cycle_type'* = *'w'*, *'mg_pre_relax'* = 5, *'mg_post_relax'* = 5, *'initial_level'* = -2, *'iterations'* = 5, *'pyramid_factor'* = 0.6.

'default_parameters' = *'accurate'*: *'mg_solver'* = *'full_multigrid'*, *'mg_cycle_type'* = *'w'*, *'mg_pre_relax'* = 2, *'mg_post_relax'* = 2, *'initial_level'* = -2, *'iterations'* = 2, *'pyramid_factor'* = 0.6.

'default_parameters' = *'fast_accurate'*: *'mg_solver'* = *'full_multigrid'*, *'mg_cycle_type'* = *'v'*, *'mg_pre_relax'* = 1, *'mg_post_relax'* = 1, *'initial_level'* = -2, *'iterations'* = 1, *'pyramid_factor'* = 0.6. Dies sind auch die Standardwerte wenn kein Parametersatz ausgewählt wird.

'default_parameters' = *'fast'*: *'mg_solver'* = *'full_multigrid'*, *'mg_cycle_type'* = *'v'*, *'mg_pre_relax'* = 1, *'mg_post_relax'* = 1, *'initial_level'* = -2, *'iterations'* = 0, *'pyramid_factor'* = 0.6.

Schwächen des Operators: Große Sprünge in der Disparität, welche großen Sprüngen im Abstand der beobachteten Objekte entsprechen, werden vergleichsweise stark geglättet. Dies führt insbesondere bei dünnen Objekten mit großem Abstand zum Hintergrund zu Problemen.

Am linken und rechten Rand des Bildes kann es zu Störungen kommen, da Bereiche der Szene dort nur in jeweils einem der beiden Bilder sichtbar sind.

Weiterhin sollten allgemeine Probleme bei Stereoanwendungen vermieden werden. Darunter fallen sich horizontal wiederholende Muster, schlecht texturierte Bereiche sowie Reflexionen.

Parameter

- ▷ **`ImageRect1`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Rektifiziertes Bild von Kamera 1.
- ▷ **`ImageRect2`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Rektifiziertes Bild von Kamera 2.
- ▷ **`Disparity`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Disparität.
- ▷ **`Score`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Güte der berechneten Disparität wenn `CalculateScore` auf *'true'* gesetzt ist.
- ▷ **`GrayConstancy`** (input_control) ..... real ⇝ *real*
  Gewicht der Grauwerterhaltung im Datenterm.
  Defaultwert: 1.0
  Wertevorschläge: `GrayConstancy` ∈ {0.0, 1.0, 2.0, 10.0}
  Restriktion: GrayConstancy >= 0.0
- ▷ **`GradientConstancy`** (input_control) ..... real ⇝ *real*
  Gewicht der Gradientenerhaltung im Datenterm.
  Defaultwert: 30.0
  Wertevorschläge: `GradientConstancy` ∈ {0.0, 1.0, 5.0, 10.0, 30.0, 50.0, 70.0}
  Restriktion: GradientConstancy >= 0.0

▷ **Smoothness** (input_control) . . . . . real ⇝ *real*
Gewicht des Glattheitsterms bezüglich des Datenterms.
Defaultwert: 5.0
Wertevorschläge: Smoothness ∈ {1.0, 3.0, 5.0, 10.0}
Restriktion: Smoothness > 0.0

▷ **InitialGuess** (input_control) . . . . . real ⇝ *real*
Startwert für die Disparität.
Defaultwert: 0.0
Wertevorschläge: InitialGuess ∈ {-30.0, -20.0, -10.0, 0.0, 10.0, 20.0, 30.0}

▷ **CalculateScore** (input_control) . . . . . string ⇝ *string*
Soll die Qualitätsschätzung in Score zurückgegeben werden?
Defaultwert: 'false'
Wertevorschläge: CalculateScore ∈ {'true', 'false'}

▷ **MGParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Parametername(n) für das Mehrgitterverfahren.
Defaultwert: 'default_parameters'
Werteliste: MGParamName ∈ {'default_parameters', 'mg_solver', 'mg_cycle_type', 'mg_pre_relax', 'mg_post_relax', 'initial_level', 'pyramid_factor', 'iterations'}

▷ **MGParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / real / integer
Parameterwert(e) für das Mehrgitterverfahren.
Defaultwert: 'fast_accurate'
Wertevorschläge: MGParamValue ∈ {'very_accurate', 'accurate', 'fast_accurate', 'fast', 'v', 'w', 'none', 'gauss_seidel', 'multigrid', 'full_multigrid', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, -1, -2, -3, -4, -5}

Beispiel

```
read_image (BaseballL, 'stereo/epipolar/baseball_l')
read_image (BaseballR, 'stereo/epipolar/baseball_r')
binocular_disparity_mg (BaseballL, BaseballR, Disparity, Score, \
                        0.25, 30, 5, 0, 'true', \
                        'default_parameters','fast_accurate')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert binocular_disparity_mg den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

map_image

Nachfolger

threshold, disparity_to_distance, disparity_image_to_xyz

Alternativen

binocular_disparity, binocular_disparity_ms, binocular_distance, binocular_distance_mg, binocular_distance_ms

Siehe auch

map_image, gen_binocular_rectification_map, binocular_calibration

Modul

3D Metrology

binocular_disparity_ms (ImageRect1, ImageRect2 : Disparity, Score : MinDisparity, MaxDisparity, SurfaceSmoothing, EdgeSmoothing, GenParamName, GenParamValue :)

Berechnet die Disparität für ein rektifiziertes Stereobildpaar mit Hilfe von Multi-Scanline-Optimierung.

binocular_disparity_ms berechnet die Disparität zwischen zwei rektifizierten Stereobildern ImageRect1 und ImageRect2 mit Hilfe von Multi-Scanline-Optimierung. Das resultierende Disparitätsbild wird in Disparity zurückgegeben. Im Gegensatz zu binocular_distance_ms wird das Ergebnis nicht in Abstandswerte umgerechnet.

Für diese Aufgabe können die drei Operatoren binocular_disparity, binocular_disparity_mg und binocular_disparity_ms verwendet werden. binocular_disparity liefert robuste Ergebnisse in gut texturierten Bereichen, doch scheitert auf schwachen Texturen. binocular_disparity_mg interpoliert schwach texturierte Bereiche, doch glättet Disparitätssprünge. binocular_disparity_ms bewahrt Disparitätssprünge und interpoliert teilweise.

Die Eingabeparameter ImageRect1 und ImageRect2 müssen ein Paar rektifizierter Stereobilder darstellen, d.h. korrespondierende Punkte müssen in derselben Zeile liegen. Falls diese Annahme nicht zutrifft, können die Bilder mit den Operatoren calibrate_cameras, gen_binocular_rectification_map und map_image rektifiziert werden.

Die Breiten von ImageRect1 und ImageRect2 können sich unterscheiden, die Höhe muss jedoch übereinstimmen. Für ein Pixel in ImageRect1 wird das homologe Pixel in ImageRect2 ausgewählt, indem die entsprechende Zeile in ImageRect2 abgesucht wird und beide Pixel basierend auf einem Ähnlichkeitsmaß gematcht werden. Die Disparität ist die Anzahl der Pixel, welche ein Pixel in ImageRect1 verschoben werden muss, um das homologe Pixel in ImageRect2 zu erreichen. Der Suchbereich wird durch den minimalen und maximalen Disparitätswert MinDisparity und MaxDisparity festgelegt. Falls der minimale und maximale Disparitätswert ein leeres Tupel sind, wird der Suchbereich automatisch aus den Eingabebildern ImageRect1 und ImageRect2 berechnet.

Um die Laufzeit zu verbessern kann ein ein grob-zu-fein Schema verwendet werden. Dieses funktioniert ähnlich zu dem grob-zu-fein Schema welches in binocular_disparity beschrieben wird. Das grob-zu-feine Schema wird verwendet, wenn GenParamName auf *'method'* und GenParamValue auf *'fast'* gesetzt werden. Die Standardmethode ist *'accurate'*. Besonders für große Bilder und einen großen Suchbereich der Disparitätswerte erfordert die *'fast'* Methode deutlich weniger Speicherplatz und ist deutlich schneller als die *'accurate'* Methode. Das grob-zu-fein Schema hat den weiteren Vorteil, dass es automatisch die Reichweite von MinDisparity und MaxDisparity beim Durchlaufen der Pyramide schätzt. Als Konsequenz müssen weder MinDisparity noch MaxDisparity gesetzt werden. Allerdings sind die Disparitätsbilder für die *'fast'*-Methode weniger genau als für den *'accurate'* Standardansatz . Insbesondere bei starken Disparitätssprüngen ist die *'fast'* Methode ungenauer. Die Laufzeit kann noch weiter reduziert werden indem die *'method'* auf *'very_fast'* gesetzt wird. Dieser Ansatz geht jedoch von zahlreichen Annahmen aus, die zu einer Glättung der Disparitäten bei Diskontinuitäten führen können. Standardmäßig wird die Anzahl der Pyramidenstufen des grob-zu-fein Schemas automatisch geschätzt. Es ist jedoch möglich die Anzahl der Stufen explizit durch das Setzen von GenParamName auf *'num_levels'* und GenParamValue auf die Zahl der Ebenen zu setzen.

Das Ähnlichkeitsmaß kann festgelegt werden, indem GenParamName auf *'similarity_measure'* und GenParamValue auf den gewünschten Wert gesetzt wird. In den beiden Fällen *'census_dense'* (default) und *'census_sparse'* basiert das Ähnlichkeitsmaß auf der Census-Transformation. Ein transformiertes Bild enthält für jedes Pixel Information über die Topologie der Intensitäten innerhalb eines Supportfensters. *'census_dense'* verwendet ein Fenster von 9 x 7 Pixeln. *'census_sparse'* verwendet ein Fenster von 15 x 15 Pixeln, von welchen jedoch nur ein Teil ausgewertet wird. Während *'census_dense'* besser für feine Strukturen geeignet ist, liefert *'census_sparse'* robustere Ergebnisse in schwach texturierten Bereichen.

Um aus dem Ähnlichkeitsmaß die Disparitäten zu berechnen, werden die Zwischenergebnisse mit Hilfe einer Multi-Scanline-Methode optimiert. Die Optimierung steigert die Robustheit in schwach texturierten Bereichen ohne Disparitätssprünge zu glätten. Die Optimierung wird durch die beiden Parameter SurfaceSmoothing und EdgeSmoothing gesteuert. SurfaceSmoothing steuert die Glattheit innerhalb von Oberflächen. Hohe Werte unterdrücken Disparitätsdifferenzen von einem Pixel. EdgeSmoothing steuert die Häufigkeit und die Form von Kanten. Niedrige Werte lassen viele Kanten zu, hohe Werte führen zu weniger und runderen Kanten. Für beide Parameter liegt ein sinnvoller Wertebereich üblicherweise zwischen 0 und 100. Sind beide Parameter auf null gesetzt, wird keine Optimierung durchgeführt.

Die resultierende Disparität wird in dem einkanaligen Bild Disparity zurückgegeben. Ein Qualitätsmaß für

jeden Disparitätswert wird in `Score` ausgegeben. Es enthält das beste (niedrigste) Ergebnis des optimierten Ähnlichkeitsmaßes eines Referenzpixels.

Schließlich gibt es zwei optionale Nachbearbeitungsschritte. Wird `GenParamName` auf *'consistency_check'* und `GenParamValue` auf *'true'* (default) gesetzt, werden nur noch diejenigen Disparitätswerte zurückgegeben, die übereinstimmend bei einer Suche eines Referenzmusters von `ImageRect1` in `ImageRect2` als auch von `ImageRect2` in `ImageRect1` gefunden werden. Der Wert *'false'* schaltet dieses Verhalten aus.

Wird `GenParamName` auf *'sub_disparity'* und `GenParamValue` auf *'true'* (default) gesetzt, werden kontinuierliche Werte für die bisher ganzzahligen Disparitätswerte geschätzt. Der Wert *'false'* schaltet dieses Verhalten aus.

Die Laufzeit des Operators verhält sich in etwa linear zur Bildbreite, zur Bildhöhe und zum Disparitätsbereich. Folglich sollte der Disparitätsbereich für große Bilder so klein wie möglich gewählt werden. Die Laufzeit der grob-zu-fein Methode (wird verwendet indem `GenParamName` auf *'method'* und `GenParamValue` auf *'fast'* oder *'very_fast'* gesetzt werden), verhält sich linear zur Bildbreite und zur Bildhöhe. Für kleine Bilder und kleine Disparitätsbereiche kann die Laufzeit der grob-zu-fein Methode größer sein als die der *'accurate'* Methode.

Parameter

- ▷ **`ImageRect1`** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Rektifiziertes Bild von Kamera 1.
- ▷ **`ImageRect2`** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Rektifiziertes Bild von Kamera 2.
- ▷ **`Disparity`** (output_object) ..... singlechannelimage ⇝ *object :* real
  Disparität.
- ▷ **`Score`** (output_object) ..... singlechannelimage ⇝ *object :* real
  Güte der berechneten Disparität.
- ▷ **`MinDisparity`** (input_control) ..... integer ⇝ *integer*
  Minimale erwartete Disparität.
  Defaultwert: -30
  Typischer Wertebereich: -32768 ≤ `MinDisparity` ≤ 32767
- ▷ **`MaxDisparity`** (input_control) ..... integer ⇝ *integer*
  Maximale erwartete Disparität.
  Defaultwert: 30
  Typischer Wertebereich: -32768 ≤ `MaxDisparity` ≤ 32767
  Restriktion: MinDisparity <= MaxDisparity
- ▷ **`SurfaceSmoothing`** (input_control) ..... integer ⇝ *integer*
  Glättung von Oberflächen.
  Defaultwert: 50
  Wertevorschläge: `SurfaceSmoothing` ∈ {20, 50, 100}
  Restriktion: SurfaceSmoothing >= 0
- ▷ **`EdgeSmoothing`** (input_control) ..... integer ⇝ *integer*
  Glättung von Kanten.
  Defaultwert: 50
  Wertevorschläge: `EdgeSmoothing` ∈ {20, 50, 100}
  Restriktion: EdgeSmoothing >= 0
- ▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
  Parametername(n) für das Multi-Scanline-Verfahren.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'method', 'similarity_measure', 'consistency_check', 'sub_disparity'}
- ▷ **`GenParamValue`** (input_control) ..... attribute.value(-array) ⇝ *string*
  Parameterwert(e) für das Multi-Scanline-Verfahren.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {'accurate', 'fast', 'very_fast', 'census_dense', 'census_sparse', 'true', 'false'}

Beispiel

```
read_image (BaseballL, 'stereo/epipolar/baseball_l')
read_image (BaseballR, 'stereo/epipolar/baseball_r')
```

```
binocular_disparity_ms (BaseballL, BaseballR, Disparity, Score, \
                        -40, -10, 50, 50, [], [])
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `binocular_disparity_ms` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`map_image`

Nachfolger

`threshold`, `disparity_to_distance`, `disparity_image_to_xyz`

Alternativen

`binocular_disparity`, `binocular_disparity_mg`, `binocular_distance`, `binocular_distance_mg`, `binocular_distance_ms`

Siehe auch

`map_image`, `gen_binocular_rectification_map`, `binocular_calibration`

Modul

3D Metrology

binocular_distance (ImageRect1, ImageRect2 : Distance, Score : CamParamRect1, CamParamRect2, RelPoseRect, Method, MaskWidth, MaskHeight, TextureThresh, MinDisparity, MaxDisparity, NumLevels, ScoreThresh, Filter, SubDistance :)

Berechnet das Distanzbild für ein rektifiziertes Bildpaar mit Hilfe von Korrelationsmethoden.

`binocular_distance` berechnet das Distanzbild für ein rektifiziertes Bildpaar mit Hilfe von Korrelationsmethoden. Der Operator berechnet zunächst die Disparitäten zwischen den zwei rektifizierten Bildern `ImageRect1` und `ImageRect2` ähnlich wie in `binocular_disparity`. Die resultierenden Disparitätswerte werden dann wie in `disparity_to_distance` in Distanzwerte der korrespondierenden 3D Weltpunkte zu dem rektifizierten Stereo-Kamerasystem transformiert. Die Distanzen werden in dem einkanaligen Bild `Distance` zurückgegeben, in dem jeder Grauwert den Abstand des jeweiligen 3D Weltpunktes zu dem Stereo-Kamerasystem repräsentiert.

Der Algorithmus benötigt ein Referenzbild `ImageRect1` und ein Suchbild `ImageRect2`, welche entzerrt sein müssen, d.h. korrespondierende Epipolarlinien liegen parallel und haben identische Bildzeilenkoordinaten ($r_1 = r_2$). Falls diese Annahme nicht zutrifft, können die Bilder durch die Operatoren `calibrate_cameras`, `gen_binocular_rectification_map` und `map_image` entzerrt werden. Ausgehend von einem Pixel im Referenzbild `ImageRect1` kann dann das zugehörige Pixel in `ImageRect2` durch die Suche entlang der korrespondierenden Zeile in `ImageRect2` gefunden werden. Eine Übereinstimmung wird gefunden, indem jeweils eine lokale Nachbarschaft um das entsprechende Pixel innerhalb eines Fensters der Größe `MaskWidth` und `MaskHeight` in beiden Bildern miteinander verglichen wird. Für jedes definierte Referenzpixel wird die Pixelkorrespondenz in einen Distanzwert des 3D Punktes transformiert, der durch den Schnittpunkt der Sichtstrahlen durch beide korrespondierenden Pixel definiert ist.

Für diese Transformation werden die entzerrten internen Parameter für die Kamera 1 in CamParamRect1 und für die Kamera 2 in CamParamRect2 als auch die relativen externen Kameraparameter RelPoseRect benötigt. Letzteres ist eine Pose der Form ${}^{ccsR1}\mathbf{P}_{ccsR2}$, welche die relative Lage der beiden Kamerasysteme zueinander beschreibt. Genauer gesagt definiert sie eine Punkttransformation von dem entzerrten Kamerakoordinatensystem 2 (*ccsR2*) in das entzerrte Kamerakoordinatensystem 1 (*ccsR1*). Siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". Die Werte für diese Parameter erhält man beispielsweise von den Operatoren calibrate_cameras und gen_binocular_rectification_map. Ein Qualitätsmaß für jeden Distanzwert wird in Score zurückgegeben, welches jeweils das beste Ergebnis der Korrelationsfunktion S für einen Referenzpixel enthält. Für die Korrelation werden die Grauwerte der originalen, nicht vorverarbeiteten Eingabebilder genutzt.

Die verwendete Korrelationsfunktion wird durch den Parameter Method bestimmt, welcher drei Arten unterscheidet:

- *'sad'*: Summed Absolute Differences
  $S(r,c,d) = \frac{1}{N} \sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n} \mid g_1(r',c') - g_2(r',c'+d) \mid,$
  with $0 \leq S(r,c,d) \leq 255$.
- *'ssd'*: Summed Squared Differences
  $S(r,c,d) = \frac{1}{N} \sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n} (g_1(r',c') - g_2(r',c'+d))^2,$
  with $0 \leq S(r,c,d) \leq 65025$.
- *'nnc'*: Normalized Cross Correlation
  $S(r,c,d) = \frac{\sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n} (g_1(r',c') - \bar{g_1}(r,c))(g_2(r',c'+d) - \bar{g_2}(r,c))}{\sqrt{\sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n+d} (g_1(r',c') - \bar{g_1}(r,c))^2 (g_2(r',c'+d) - \bar{g_2}(r,c+d))^2}},$
  with $-1.0 \leq S(r,c,d) \leq 1.0$.

wobei

$r1, c1, r2, c2$: Zeilen- und Spaltenkoordinaten der korrespondierenden Pixel der beiden Eingabebilder,

$g1, g2$: Grauwerte der nicht vorverarbeiteten Eingabebilder,

$N = (2m+1)(2n+1)$: Größe des Korrelationsfensters,

$\bar{g}(r,c) = \frac{1}{N} \sum_{r'=r-m}^{r+m} \sum_{c'=c-n}^{c+n} g(r',c')$: mittlerer Grauwert innerhalb eines Korrelationsfensters der Breite $2m+1$ und Höhe $2n+1$.

Die Methoden *'sad'* und *'ssd'* vergleichen direkt die Grauwerte der Pixel innerhalb des Grauwertfensters, wohingegen *'ncc'* den mittleren Grauwert und seine Varianz innerhalb des Grauwertfensters ausgleicht. Wenn zwei Bilder in ihrer Helligkeit und ihrem Kontrast unterschiedlich sind, ist daher *'ncc'* vorzuziehen. Ansonsten sind *'sad'* und *'ssd'* vorzuziehen, da sie aufgrund einfacherer interner Berechnungen schneller sind.

Es muss darauf geachtet werden, dass die Güte der Korrelation für steigende S in den Methoden *'sad'* und *'ssd'* fällt (die beste Güte ist 0), während sie in der Methode *'ncc'* steigt (die beste Güte ist 1.0).

Die Größe des Korrelationsfensters, welche durch $2m+1$ und $2n+1$ in obigen Gleichungen referenziert wird, muss ungeradzahlig sein und kann durch die Parameter MaskWidth und MaskHeight festgelegt werden. Der Suchraum wird durch den minimalen und maximalen Disparitätswert MinDisparity und MaxDisparity festgelegt. Da Grauwerte nicht außerhalb des Bildrandes definiert sind wird die Bildregion des Distance und Score Bildes am unteren und oberen Bildrand um (MaskHeight-1)/2 sowie am linken und rechten Rand um (MaskWidth-1)/2 beschnitten. Aus dem gleichen Grund wird der maximale Suchraum am linken und rechten Bildrand begrenzt.

Da die Korrelationsverfahren in schwach texturierten Bildregionen im hohen Maße ungenau werden, kann die kleinste, innerhalb eines Korrelationsfensters erlaubte Grauwertstreuung mit Hilfe des Parameters TextureThresh definiert werden. Dieser Schwellenwert wird auf beiden Eingabebildern ImageRect1 und ImageRect2 angewendet. Zusätzlich garantiert ScoreThresh eine Mindestgüte der verwendeten Korrelationsfunktion und legt für die Methoden *'sad'* und *'ssd'* den größten, bzw. für *'ncc'* den kleinsten zulässigen Korrelationswert fest. Wird ferner Filter auf *'left_right_check'* gesetzt, werden nur noch diejenigen Disparitätswerte zurückgegeben, die übereinstimmend bei einer Suche eines Referenzmusters von ImageRect1 in ImageRect2

als auch von ImageRect2 in ImageRect1 gefunden werden. Der Wert *'none'* hingegen schaltet dieses Verhalten aus.

Die Anzahl der Pyramidenstufen, welche verwendet werden um das Zeitverhalten von binocular_distance zu verbessern, wird durch den Parameter NumLevels bestimmt. Einem grob-zu-fein Schema folgend werden die Disparitätsbilder der höheren Stufen berechnet und in rechteckige Bildteile von ähnlicher Disparität unterteilt um den Suchbereich auf der nächst niedrigeren Stufe für diesen Bildausschnitt einzuschränken. TextureThresh und ScoreThresh werden auf jeder Stufe angewendet, so dass die mit Distance und Score zurückgegebene Bildregion als die Schnittmenge der Bildregionen auf jeder einzelnen Stufe definiert ist. Im Allgemeinen lohnt sich das Verwenden von Pyramidenstrukturen um so mehr, je mehr das Distanzbild in homogene Bereiche unterteilt werden kann und je größer der Disparitätsbereich definiert werden muss. Auf der anderen Seite können zu hohe Pyramidenstufen durch ihre grobe Auflösung wichtige Texturinformationen verlieren, so dass falsche Distanzwerte zurückgegeben werden können.

Der Wert *'interpolation'* für den Parameter SubDistance schließlich erhöht die Genauigkeit und Unterteilung der Distanzwerte. Das Verhalten wird ausgeschaltet indem der Parameter auf *'none'* gesetzt wird.

Achtung

Bei der Verwendung von telezentrischen Kameras bezieht sich die Distanz Distance nicht auf die Entfernung eines Punktes zum Projektionszentrum einer Kamera. Die Entfernung wird zu der Ebene gemessen, die durch die y-Achse beider Kameras und der Basis (siehe gen_binocular_rectification_map) definiert ist.

Für Stereosysteme gemischten Typs (d.h., falls das ursprüngliche Stereosystem aus einer perspektivischen und einer telezentrischen Kamera bestand; siehe gen_binocular_rectification_map), ist die rektifizierende Ebene der zwei Kameras in einer Lage relativ zum Objekt, die zu sehr unintuitiven Distanzen führen würde. Daher unterstützt binocular_distance Stereosysteme gemischten Typs nicht. Für Stereosysteme gemischten Typs sollte stattdessen reconstruct_surface_stereo verwendet werden. Dort kann das Referenzkoordinatensystem beliebig gewählt werden. Alternativ können eventuell binocular_disparity und disparity_image_to_xyz verwendet werden.

Auch Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **ImageRect1** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Rektifiziertes Bild von Kamera 1.
- ▷ **ImageRect2** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Rektifiziertes Bild von Kamera 2.
- ▷ **Distance** (output_object) ..... singlechannelimage ⇝ *object :* real
  Distanzbild.
- ▷ **Score** (output_object) ..... singlechannelimage ⇝ *object :* real
  Maß für die Zuverlässigkeit eines Disparitätwertes.
- ▷ **CamParamRect1** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter der rektifizierten Kamera 1.
- ▷ **CamParamRect2** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter der rektifizierten Kamera 2.
- ▷ **RelPoseRect** (input_control) ..... pose ⇝ *real* / integer
  Punkttransformation von entzerrter Kamera 2 zu entzerrter Kamera 1.
  Parameteranzahl: 7
- ▷ **Method** (input_control) ..... string ⇝ *string*
  Korrespondenzverfahren.
  Defaultwert: 'ncc'
  Werteliste: Method ∈ {'sad', 'ssd', 'ncc'}
- ▷ **MaskWidth** (input_control) ..... integer ⇝ *integer*
  Breite des Korrelationsfensters.
  Defaultwert: 11
  Wertevorschläge: MaskWidth ∈ {5, 7, 9, 11, 21}
  Restriktion: 3 <= MaskWidth && odd(MaskWidth)

▷ **MaskHeight** (input_control) ..... integer ⇝ *integer*
Höhe des Korrelationsfensters.
Defaultwert: 11
Wertevorschläge: `MaskHeight` ∈ {5, 7, 9, 11, 21}
Restriktion: 3 <= MaskHeight && odd(MaskHeight)

▷ **TextureThresh** (input_control) ..... real ⇝ *real* / integer
Schwellenwert der Grauwertvarianz.
Defaultwert: 0.0
Wertevorschläge: `TextureThresh` ∈ {0.0, 2.0, 5.0, 10.0}
Restriktion: 0.0 <= TextureThresh

▷ **MinDisparity** (input_control) ..... integer ⇝ *integer*
Minimale erwartete Disparität.
Defaultwert: 0
Typischer Wertebereich: -32768 ≤ `MinDisparity` ≤ 32767

▷ **MaxDisparity** (input_control) ..... integer ⇝ *integer*
Maximale erwartete Disparität.
Defaultwert: 30
Typischer Wertebereich: -32768 ≤ `MaxDisparity` ≤ 32767

▷ **NumLevels** (input_control) ..... integer ⇝ *integer*
Anzahl der Pyramidenstufen.
Defaultwert: 1
Wertevorschläge: `NumLevels` ∈ {1, 2, 3, 4}
Restriktion: 1 <= NumLevels

▷ **ScoreThresh** (input_control) ..... real ⇝ *real* / integer
Schwellenwert der Korrelationsfunktion.
Defaultwert: 0.0
Wertevorschläge: `ScoreThresh` ∈ {0.0, 2.0, 5.0, 10.0}

▷ **Filter** (input_control) ..... string(-array) ⇝ *string*
Verwendeter Filter.
Defaultwert: 'none'
Werteliste: `Filter` ∈ {'none', 'left_right_check'}

▷ **SubDistance** (input_control) ..... string(-array) ⇝ *string*
Distanz Interpolation.
Defaultwert: 'none'
Werteliste: `SubDistance` ∈ {'none', 'interpolation'}

Beispiel

```
* Set internal and external stereo parameters.
* Note that, typically, these values are the result of a prior
* calibration.
gen_cam_par_area_scan_division (0.01, -665, 5.2e-006, 5.2e-006, \
                                622, 517, 1280, 1024, CamParam1)
gen_cam_par_area_scan_division (0.01, -731, 5.2e-006, 5.2e-006, \
                                654, 519, 1280, 1024, CamParam2)
create_pose (0.1535,-0.0037,0.0447,0.17,319.84,359.89, \
             'Rp+T', 'gba', 'point', RelPose)
* Compute the mapping for rectified images.
gen_binocular_rectification_map (Map1, Map2, CamParam1, CamParam2, \
                                 RelPose, 1, 'viewing_direction', 'bilinear', \
                                 CamParamRect1, CamParamRect2, \
                                 Cam1PoseRect1, Cam2PoseRect2, RelPoseRect)
* Compute the distance values in online images.
while (1)
  grab_image_async (Image1, AcqHandle1, -1)
  map_image (Image1, Map1, ImageRect1)

  grab_image_async (Image2, AcqHandle2, -1)
  map_image (Image2, Map2, ImageRect2)
```

```
  binocular_distance (ImageRect1, ImageRect2, Distance, Score, \
                      CamParamRect1, CamParamRect2, RelPoseRect, 'sad', \
                      11, 11, 20, -40, 20, 2, 25, \
                      'left_right_check', 'interpolation')
endwhile
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert binocular_disparity den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

map_image

Nachfolger

threshold

Alternativen

binocular_distance_mg, binocular_distance_ms, binocular_disparity, binocular_disparity_mg, binocular_disparity_ms

Siehe auch

map_image, gen_binocular_rectification_map, binocular_calibration, distance_to_disparity, disparity_to_distance, disparity_image_to_xyz

Modul

3D Metrology

binocular_distance_mg (ImageRect1, ImageRect2 : Distance, Score : CamParamRect1, CamParamRect2, RelPoseRect, GrayConstancy, GradientConstancy, Smoothness, InitialGuess, CalculateScore, MGParamName, MGParamValue :)

Berechnet das Distanzbild für ein rektifiziertes Bildpaar mit Hilfe von Mehrgitter-Verfahren.

binocular_distance_mg berechnet das Distanzbild für ein rektifiziertes Bildpaar mit Hilfe von Mehrgitter-Verfahren. Der Operator berechnet zunächst die Disparitäten zwischen den zwei rektifizierten Bildern ImageRect1 und ImageRect2 ähnlich wie in binocular_disparity_mg. Die resultierenden Disparitätswerte werden dann wie in disparity_to_distance in Distanzwerte der korrespondierenden 3D Weltpunkte zu dem rektifizierten Stereo-Kamerasystem transformiert. Die Distanzen werden in dem einkanaligen Bild Distance zurückgegeben, in dem jeder Grauwert den Abstand des jeweiligen 3D Weltpunktes zu dem Stereo-Kamerasystem repräsentiert. Anders als in binocular_distance verwendet dieser Operator Variationsmethoden basierend auf Mehrgitter-Verfahren. Damit können auch Distanzwerte für Bildbereiche bestimmt werden, die keine oder kaum Textur aufweisen.

Die Eingabeparameter ImageRect1 und ImageRect2 müssen ein Paar rektifizierte Stereobilder darstellen, d.h. korrespondierende Punkte müssen dieselbe Zeilen-Koordinate haben. Falls diese Annahme nicht zutrifft, können die Bilder mit den Operatoren calibrate_cameras, gen_binocular_rectification_map und map_image rektifiziert werden.

Für die Transformation der Disparität in den Abstand werden die internen Parameter für die rektifizierte Kamera 1 in CamParamRect1 und für die rektifizierte Kamera 2 in CamParamRect2 sowie die relativen Lage der Kameras in RelPoseRect benötigt. Die relative Lage definiert eine Punkttransformation von dem entzerrten Kamerasystem 2 in das entzerrte Kamerasystem 1. Die Werte für diese Parameter erhält man beispielsweise von den Operatoren calibrate_cameras und gen_binocular_rectification_map.

Die Beschreibung des Verfahrens sowie der weiteren Parameter findet sich in der Dokumentation von binocular_disparity_mg.

Achtung

Bei der Verwendung von telezentrischen Kameras bezieht sich die Distanz `Distance` nicht auf die Entfernung eines Punktes zum Projektionszentrum einer Kamera. Die Entfernung wird zu der Ebene gemessen, die durch die y-Achse beider Kameras und der Basis (siehe `gen_binocular_rectification_map`) definiert ist.

Für Stereosysteme gemischten Typs (d.h., falls das ursprüngliche Stereosystem aus einer perspektivischen und einer telezentrischen Kamera bestand; siehe `gen_binocular_rectification_map`), ist die rektifizierende Ebene der zwei Kameras in einer Lage relativ zum Objekt, die zu sehr unintuitiven Distanzen führen würde. Daher unterstützt `binocular_distance_mg` Stereosysteme gemischten Typs nicht. Für Stereosysteme gemischten Typs sollte stattdessen `reconstruct_surface_stereo` verwendet werden. Dort kann das Referenzkoordinatensystem beliebig gewählt werden. Alternativ können eventuell `binocular_disparity_mg` und `disparity_image_to_xyz` verwendet werden.

Auch Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **`ImageRect1`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Rektifiziertes Bild von Kamera 1.
- ▷ **`ImageRect2`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Rektifiziertes Bild von Kamera 2.
- ▷ **`Distance`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Abstandsbild.
- ▷ **`Score`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Güte der berechneten Disparität wenn `CalculateScore` auf *'true'* gesetzt ist.
- ▷ **`CamParamRect1`** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter der rektifizierten Kamera 1.
- ▷ **`CamParamRect2`** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter der rektifizierten Kamera 2.
- ▷ **`RelPoseRect`** (input_control) ..... pose ⇝ *real* / integer
  Punkttransformation von entzerrter Kamera 2 zu entzerrter Kamera 1.
  Parameteranzahl: 7
- ▷ **`GrayConstancy`** (input_control) ..... real ⇝ *real*
  Gewicht der Grauwerterhaltung im Datenterm.
  Defaultwert: 1.0
  Wertevorschläge: `GrayConstancy` ∈ {0.0, 1.0, 2.0, 10.0}
  Restriktion: GrayConstancy >= 0.0
- ▷ **`GradientConstancy`** (input_control) ..... real ⇝ *real*
  Gewicht der Gradientenerhaltung im Datenterm.
  Defaultwert: 30.0
  Wertevorschläge: `GradientConstancy` ∈ {0.0, 1.0, 5.0, 10.0, 30.0, 50.0, 70.0}
  Restriktion: GradientConstancy >= 0.0
- ▷ **`Smoothness`** (input_control) ..... real ⇝ *real*
  Gewicht des Glattheitsterms bezüglich des Datenterms.
  Defaultwert: 5.0
  Wertevorschläge: `Smoothness` ∈ {1.0, 3.0, 5.0, 10.0}
  Restriktion: Smoothness > 0.0
- ▷ **`InitialGuess`** (input_control) ..... real ⇝ *real*
  Startwert für die Disparität.
  Defaultwert: 0.0
  Wertevorschläge: `InitialGuess` ∈ {-30.0, -20.0, -10.0, 0.0, 10.0, 20.0, 30.0}
- ▷ **`CalculateScore`** (input_control) ..... string ⇝ *string*
  Soll die Qualitätsschätzung in `Score` zurückgegeben werden?.
  Defaultwert: 'false'
  Wertevorschläge: `CalculateScore` ∈ {'true', 'false'}
- ▷ **`MGParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
  Parametername(n) für das Mehrgitterverfahren.
  Defaultwert: 'default_parameters'
  Werteliste: `MGParamName` ∈ {'default_parameters', 'mg_solver', 'mg_cycle_type', 'mg_pre_relax', 'mg_post_relax', 'initial_level', 'pyramid_factor', 'iterations'}

▷ **MGParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / real / integer
Parameterwert(e) für das Mehrgitterverfahren.
Defaultwert: 'fast_accurate'
Wertevorschläge: MGParamValue ∈ {'very_accurate', 'accurate', 'fast_accurate', 'fast', 'v', 'w', 'none', 'gauss_seidel', 'multigrid', 'full_multigrid', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, -1, -2, -3, -4, -5}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert binocular_distance_mg den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

map_image

Nachfolger

threshold

Alternativen

binocular_distance, binocular_distance_ms, binocular_disparity, binocular_disparity_mg, binocular_disparity_ms

Siehe auch

map_image, gen_binocular_rectification_map, binocular_calibration, disparity_to_distance, distance_to_disparity, disparity_image_to_xyz

Modul

3D Metrology

binocular_distance_ms (ImageRect1, ImageRect2 : Distance, Score : CamParamRect1, CamParamRect2, RelPoseRect, MinDisparity, MaxDisparity, SurfaceSmoothing, EdgeSmoothing, GenParamName, GenParamValue :)

Berechnet das Distanzbild für ein rektifiziertes Stereobildpaar mit Hilfe von Multi-Scanline-Optimierung.

binocular_distance_ms berechnet das Distanzbild für ein rektifiziertes Stereobildpaar mit Hilfe von Multi-Scanline-Optimierung. Der Operator berechnet zunächst die Disparitäten zwischen den zwei rektifizierten Bildern ImageRect1 und ImageRect2, ähnlich wie binocular_disparity_ms. Die resultierenden Disparitätswerte werden dann wie in disparity_to_distance in Distanzwerte der korrespondierenden 3D Weltpunkte zu dem rektifizierten Stereo-Kamerasystem transformiert. Die Distanzen werden in dem einkanaligen Bild Distance zurückgegeben, in dem jeder Grauwert den Abstand des jeweiligen 3D Weltpunktes zu dem Stereo-Kamerasystem repräsentiert.

Die Eingabeparameter ImageRect1 und ImageRect2 müssen ein Paar rektifizierter Stereobilder darstellen, d.h. korrespondierende Punkte müssen in derselben Zeile liegen. Falls diese Annahme nicht zutrifft, können die Bilder mit den Operatoren calibrate_cameras, gen_binocular_rectification_map und map_image rektifiziert werden.

Für die Transformation der Disparität in den Abstand werden die internen Parameter für die rektifizierte Kamera 1 in CamParamRect1 und für die rektifizierte Kamera 2 in CamParamRect2 sowie die relative Lage der Kameras in RelPoseRect benötigt. Die relative Lage definiert eine Punkttransformation von dem entzerrten Kamerasystem 2 in das entzerrte Kamerasystem 1. Die Werte für diese Parameter erhält man beispielsweise von den Operatoren calibrate_cameras und gen_binocular_rectification_map.

Die Beschreibung der weiteren Parameter findet sich in der Dokumentation von binocular_disparity_ms.

Achtung

Bei der Verwendung von telezentrischen Kameras bezieht sich die Distanz Distance nicht auf die Entfernung eines Punktes zum Projektionszentrum einer Kamera. Die Entfernung wird zu der Ebene gemessen, die durch die y-Achse beider Kameras und der Basis (siehe gen_binocular_rectification_map) definiert ist.

Für Stereosysteme gemischten Typs (d.h., falls das ursprüngliche Stereosystem aus einer perspektivischen und einer telezentrischen Kamera bestand; siehe gen_binocular_rectification_map), ist die rektifizierende Ebene der zwei Kameras in einer Lage relativ zum Objekt, die zu sehr unintuitiven Distanzen führen würde. Daher unterstützt binocular_distance_ms Stereosysteme gemischten Typs nicht. Für Stereosysteme gemischten Typs sollte stattdessen reconstruct_surface_stereo verwendet werden. Dort kann das Referenzkoordinatensystem beliebig gewählt werden. Alternativ können eventuell binocular_disparity_ms und disparity_image_to_xyz verwendet werden.

Auch Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **ImageRect1** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
  Rektifiziertes Bild von Kamera 1.
- ▷ **ImageRect2** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
  Rektifiziertes Bild von Kamera 2.
- ▷ **Distance** (output_object) . . . . . singlechannelimage ⇝ *object :* real
  Abstandsbild.
- ▷ **Score** (output_object) . . . . . singlechannelimage ⇝ *object :* real
  Güte der berechneten Disparität.
- ▷ **CamParamRect1** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter der rektifizierten Kamera 1.
- ▷ **CamParamRect2** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter der rektifizierten Kamera 2.
- ▷ **RelPoseRect** (input_control) . . . . . pose ⇝ *real* / integer
  Punkttransformation von entzerrter Kamera 2 zu entzerrter Kamera 1.
  Parameteranzahl: 7
- ▷ **MinDisparity** (input_control) . . . . . integer ⇝ *integer*
  Minimale erwartete Disparität.
  Defaultwert: -30
  Typischer Wertebereich: -32768 ≤ MinDisparity ≤ 32767
- ▷ **MaxDisparity** (input_control) . . . . . integer ⇝ *integer*
  Maximale erwartete Disparität.
  Defaultwert: 30
  Typischer Wertebereich: -32768 ≤ MaxDisparity ≤ 32767
  Restriktion: MinDisparity <= MaxDisparity
- ▷ **SurfaceSmoothing** (input_control) . . . . . integer ⇝ *integer*
  Glättung von Oberflächen.
  Defaultwert: 50
  Wertevorschläge: SurfaceSmoothing ∈ {20, 50, 100}
  Restriktion: SurfaceSmoothing >= 0
- ▷ **EdgeSmoothing** (input_control) . . . . . integer ⇝ *integer*
  Glättung von Kanten.
  Defaultwert: 50
  Wertevorschläge: EdgeSmoothing ∈ {20, 50, 100}
  Restriktion: EdgeSmoothing >= 0
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Parametername(n) für das Multi-Scanline-Verfahren.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'similarity_measure', 'disparity_offset', 'num_levels', 'consistency_check', 'sub_disparity'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer
Parameterwert(e) für das Multi-Scanline-Verfahren.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'census_dense', 'census_sparse', 'true', 'false'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert binocular_distance_ms den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

map_image

Nachfolger

threshold

Alternativen

binocular_distance, binocular_distance_mg, binocular_disparity, binocular_disparity_mg, binocular_disparity_ms

Siehe auch

map_image, gen_binocular_rectification_map, binocular_calibration, disparity_to_distance, distance_to_disparity, disparity_image_to_xyz

Modul

3D Metrology

disparity_image_to_xyz (Disparity : X, Y, Z : CamParamRect1, CamParamRect2, RelPoseRect :)

Berechnet aus einem Disparitätsbild eines rektifizierten Bildpaars die zugehörigen 3D-Punkte.

Aus dem Disparitätsbild Disparity eines rektifizierten binokularen Stereosystems berechnet disparity_image_to_xyz die korrespondierenden 3D-Punkte. Ihre Koordinaten bezüglich der rektifizierten Kamera 1 werden als Grauwerte in den Bildern X, Y und Z zurückgegeben, d.h. die Pixel an der Position (Row,Colum) in X, Y und Z enthalten die X-, Y- und Z-Koordinate des Pixels (Row,Colum) im Disparitätsbild.

Das rektifizierte binokulare Kamerasystem wird durch die internen Kameraparameter CamParamRect1 der rektifizierten Kamera 1 und CamParamRect2 der rektifizierten Kamera 2 und die externen Parameter RelPoseRect bestimmt. Letzteres ist eine Pose, die in der Form ${}^{ccsR1}\mathbf{P}_{ccsR2}$ erwartet wird. Das heißt, sie beschreibt die Lage des rektifizierten Kamerasystems 2 (*ccR2*) bezüglich des rektifizierten Kamerasystems 1 (*ccR1*). Siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". Die Werte für diese Kameraparameter können von den Operatoren calibrate_cameras und gen_binocular_rectification_map erhalten werden.

Achtung

Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **Disparity** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* real
  Disparitätsbild.
- ▷ **X** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* real
  X-Koordinaten der Punkte im rektifizierten Kamerasystem 1.
- ▷ **Y** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* real
  Y-Koordinaten der Punkte im rektifizierten Kamerasystem 1.
- ▷ **Z** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* real
  Z-Koordinaten der Punkte im rektifizierten Kamerasystem 1.
- ▷ **CamParamRect1** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter der rektifizierten Kamera 1.
- ▷ **CamParamRect2** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter der rektifizierten Kamera 2.
- ▷ **RelPoseRect** (input_control) . . . . . pose ⇝ *real* / integer
  Lage der zweiten rektifizierten Kamera bzgl. der ersten rektifizierten Kamera.
  Parameteranzahl: 7

Beispiel

```
disparity_image_to_xyz (ImageDisparity, ImgX, ImgY, ImgZ, RectCamParL, \
                        RectCamParR, RectLPosRectR)
get_region_points (ImageDisparity, Rows, Columns)
get_grayval (ImgX, Rows, Columns, XValues)
get_grayval (ImgY, Rows, Columns, YValues)
get_grayval (ImgZ, Rows, Columns, ZValues)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `disparity_image_to_xyz` den Wert TRUE. Das Verhalten bei leerer Eingabe (kein Eingabebild vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`binocular_disparity`

Nachfolger

`threshold`, `write_image`

Alternativen

`disparity_to_point_3d`, `binocular_distance`

Siehe auch

`binocular_calibration`, `gen_binocular_rectification_map`, `intersect_lines_of_sight`

Modul

3D Metrology

disparity_to_distance (: : CamParamRect1, CamParamRect2, RelPoseRect, Disparity : Distance)

Berechnet den Abstand für Disparitäten in einem gleichgerichteten binokularen Kamerapaar.

disparity_to_distance transformiert einen Disparitätswert in einen Abstand eines Objektpunktes zum binokularen Stereosystem. Die Kameras dieses Systems müssen gleichgerichtet sein und werden durch die entzerrten internen Parameter CamParamRect1 der Kamera 1 und CamParamRect2 der Kamera 2 und die externen Parameter RelPoseRect beschrieben, welche die relative Lage der beiden Kamerasysteme zueinander beschreibt und eine Punkttransformation vom entzerrten Kamerasystem 2 zum entzerrten Kamerasystem 1 festlegt. Die Werte für diese Parameter kann man von den Operatoren calibrate_cameras und gen_binocular_rectification_map erhalten. Der Disparitätswert Disparity wird durch die Spaltendifferenz der Bildkoordinaten zweier korrespondierender Punkte auf einer Epipolarlinie definiert nach der Beziehung $d = c_2 - c_1$ (siehe auch binocular_disparity). Dieser Wert bestimmt eine Menge von 3D Objektpunkten mit gleichem Abstand zu einer zur entzerrten Bildebene parallelen Ebene. Der Abstand zu der Ebene $z = 0$, welche parallel zur entzerrten Bildebene ist und beide optischen Zentren enthält, wird in Distance zurückgegeben.

Achtung

Bei der Verwendung von telezentrischen Kameras bezieht sich die Distanz Distance nicht auf die Entfernung eines Punktes zur Bildebene einer Kamera. Die Entfernung wird zu der Ebene gemessen, die durch die y-Achse beider Kameras und der Basis (siehe gen_binocular_rectification_map) definiert ist.

Für Stereosysteme gemischten Typs (d.h., falls das ursprüngliche Stereosystem aus einer perspektivischen und einer telezentrischen Kamera bestand; siehe gen_binocular_rectification_map), ist die rektifizierende Ebene der zwei Kameras in einer Lage relativ zum Objekt, die zu sehr unintuitiven Distanzen führen würde. Daher unterstützt disparity_to_distance Stereosysteme gemischten Typs nicht. Für Stereosysteme gemischten Typs sollte stattdessen disparity_to_point_3d verwendet werden.

Auch Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **CamParamRect1** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Parameter der rektifizierten Kamera 1.
- ▷ **CamParamRect2** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Parameter der rektifizierten Kamera 2.
- ▷ **RelPoseRect** (input_control) . . . . . pose ⇝ *real* / integer
  Punkttransformation von entzerrter Kamera 2 zur entzerrten Kamera 1.
  Parameteranzahl: 7
- ▷ **Disparity** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Disparität zwischen den Punkt-Abbildungen.
- ▷ **Distance** (output_control) . . . . . real(-array) ⇝ *real*
  Abstand eines Weltpunktes zur rektifizierten Kamera 1.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert disparity_to_distance den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

binocular_calibration, gen_binocular_rectification_map, map_image, binocular_disparity

Alternativen

binocular_distance

Siehe auch

distance_to_disparity, disparity_to_point_3d

Modul

3D Metrology

disparity_to_point_3d (: : CamParamRect1, CamParamRect2, RelPoseRect, Row1, Col1, Disparity : X, Y, Z)

Berechnet aus Bildpunkten und ihren Disparitäten in einem gleichgerichteten Kamerapaar die zugehörigen Weltpunkte.

Ausgehend von einem Bildpunkt (Row1,Col1) in dem rektifizierten Kamerasystem 1 und seiner Disparität in einem rektifizierten binokularen Stereosystem, berechnet disparity_to_point_3d den korrespondierenden 3D Objektpunkt. Dabei spezifiziert der Disparitätswert Disparity die Spaltendifferenz der Bildkoordinaten zweier korrespondierender Merkmale auf einer Epipolarlinie entsprechend der Gleichung $d = c_2 - c_1$. Das rektifizierte binokulare Kamerasystem wird durch seine internen Kameraparameter CamParamRect1 der Kamera 1 und CamParamRect2 der Kamera 2 und die externen Parameter RelPoseRect bestimmt, welche die Lage des rektifizierte Kamerasystems 2 bezüglich des rektifizierten Kamerasystems 1 beschreiben. Die Werte für diese Kameraparameter können von den Operatoren calibrate_cameras und gen_binocular_rectification_map erhalten werden. Der 3D-Punkt wird in den kartesischen Koordinaten (X,Y,Z) des rektifizierten Kamerasystems 1 zurückgegeben.

Achtung

Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **CamParamRect1** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter der Kamera 1.
- ▷ **CamParamRect2** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter der Kamera 2.
- ▷ **RelPoseRect** (input_control) ..... pose ⇝ *real* / integer
  Lage der zweiten Kamera bzgl. der ersten.
  Parameteranzahl: 7
- ▷ **Row1** (input_control) ..... number(-array) ⇝ *real* / integer
  Zeilenkoordinate eines Punktes im rektifizierten Kamerasystem 1.
- ▷ **Col1** (input_control) ..... number(-array) ⇝ *real* / integer
  Spaltenkoordinate eines Punktes im rektifizierten Kamerasystem 1.
- ▷ **Disparity** (input_control) ..... number(-array) ⇝ *real* / integer
  Disparität zwischen den Punkt-Abbildungen.
- ▷ **X** (output_control) ..... real(-array) ⇝ *real*
  X Koordinate des Weltpunktes.
- ▷ **Y** (output_control) ..... real(-array) ⇝ *real*
  Y Koordinate des Weltpunktes.
- ▷ **Z** (output_control) ..... real(-array) ⇝ *real*
  Z Koordinate des Weltpunktes.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert disparity_to_point_3d den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

binocular_calibration, gen_binocular_rectification_map

Alternativen

disparity_image_to_xyz

Siehe auch

binocular_disparity, binocular_distance, intersect_lines_of_sight

Modul

3D Metrology

distance_to_disparity (: : CamParamRect1, CamParamRect2, RelPoseRect, Distance : Disparity)

Berechnet aus Abständen die erwartete Disparität der Bildpunkte in einem gleichgerichteten Kamerasystem.

distance_to_disparity transformiert den Abstand eines 3D Punktes zum binokularen Stereosystem in einen Disparitätswert. Die Kameras dieses Systems müssen gleichgerichtet sein und werden durch die entzerrten internen Parameter CamParamRect1 der Kamera 1 und CamParamRect2 der Kamera 2 und die externen Parameter RelPoseRect beschrieben, welche die relative Lage der beiden Kamerasysteme zueinander festlegt, indem es eine Punkttransformation vom entzerrten Kamerasystem 1 in das entzerrte Kamerasystem 2 definiert. Die Werte für diese Parameter kann man von dem Operator calibrate_cameras und gen_binocular_rectification_map erhalten. Der Distanzwert wird in Distance übergeben und der entsprechende Disparitätswert in Disparity zurückgegeben. Die Disparität definiert sich dabei durch die Differenz der Spalten der Bildkoordinaten zweier korrespondierender Punkte auf einer epipolaren Linie entsprechend der Gleichung $d = c_2 - c_1$.

Achtung

Bei der Verwendung von telezentrischen Kameras bezieht sich die Distanz Distance nicht auf die Entfernung eines Punktes zur Bildebene einer Kamera. Die Entfernung wird zu der Ebene gemessen, die durch die y-Achse beider Kameras und der Basis (siehe gen_binocular_rectification_map) definiert ist.

Für Stereosysteme gemischten Typs (d.h., falls das ursprüngliche Stereosystem aus einer perspektivischen und einer telezentrischen Kamera bestand; siehe gen_binocular_rectification_map), ist die rektifizierende Ebene der zwei Kameras in einer Lage relativ zum Objekt, die zu sehr unintuitiven Distanzen führen würde. Daher unterstützt distance_to_disparity Stereosysteme gemischten Typs nicht.

Auch Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **CamParamRect1** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter der Kamera 1.
- ▷ **CamParamRect2** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter der Kamera 2.
- ▷ **RelPoseRect** (input_control) . . . . . pose ⇝ *real* / integer
  Punkttransformation von entzerrter Kamera 2 zur entzerrten Kamera 1.
  Parameteranzahl: 7
- ▷ **Distance** (input_control) . . . . . real(-array) ⇝ *real*
  Abstand eines Weltpunktes zu Kamera 1.
- ▷ **Disparity** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Disparität zwischen den Punkt-Abbildungen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert distance_to_disparity den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

binocular_calibration, gen_binocular_rectification_map

Nachfolger

binocular_disparity

Modul

3D Metrology

essential_to_fundamental_matrix (: : EMatrix, CovEMat, CamMat1, CamMat2 : FMatrix, CovFMat)

Berechnung der Fundamental-Matrix aus der Essential-Matrix.

Die Fundamental-Matrix beschreibt die Epipolarbedingung in Bildkoordinaten (C,R), während die Essential-Matrix das Gegenstück für 3D Richtungsvektoren (X,Y,1) ist:

$$\begin{pmatrix} C_2 \\ R_2 \\ 1 \end{pmatrix}^T \cdot \texttt{FMatrix} \cdot \begin{pmatrix} C_1 \\ R_1 \\ 1 \end{pmatrix} = 0 \quad \text{and} \quad \begin{pmatrix} X_2 \\ Y_2 \\ 1 \end{pmatrix}^T \cdot \texttt{EMatrix} \cdot \begin{pmatrix} X_1 \\ Y_1 \\ 1 \end{pmatrix} = 0 \quad .$$

Bildkoordinaten resultieren aus 3D Richtungsvektoren durch die Multiplikation mit der Kameramatrix $CamMat$:

$$\begin{pmatrix} col \\ row \\ 1 \end{pmatrix} = CamMat \cdot \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \quad .$$

Die Fundamental-Matrix FMatrix kann daher aus der Essential-Matrix EMatrix und den Kameramatrizen CamMat1, CamMat2 mittels folgender Formel berechnet werden:

$$\texttt{FMatrix} = \texttt{CamMat2}^{-T} \cdot \texttt{EMatrix} \cdot \texttt{CamMat1}^{-1} \quad .$$

Die Transformation geht einher mit einer Fortpflanzung der Kovarianzmatrizen von CovEMat zu CovFMat. Sollte CovEMat leer sein, so wird CovFMat ebenfalls leer sein.

Der Konvertierungsoperator essential_to_fundamental_matrix wird besonders zur Visualisierung der Epipolarlinienstruktur mittels der Fundamental-Matrix verwendet, was die zu Grunde liegende Stereogeometrie verdeutlicht.

Parameter

- ▷ **EMatrix** (input_control) . . . . . hom_mat2d ⇝ *real* / integer
  Essential-Matrix.
- ▷ **CovEMat** (input_control) . . . . . number-array ⇝ *real* / integer
  9×9 Kovarianzmatrix der Essential-Matrix.
  Defaultwert: []
- ▷ **CamMat1** (input_control) . . . . . hom_mat2d ⇝ *real* / integer
  Kameramatrix der 1. Kamera.
- ▷ **CamMat2** (input_control) . . . . . hom_mat2d ⇝ *real* / integer
  Kameramatrix der 2. Kamera.
- ▷ **FMatrix** (output_control) . . . . . hom_mat2d ⇝ *real*
  Berechnete Fundamental-Matrix.
- ▷ **CovFMat** (output_control) . . . . . real-array ⇝ *real*
  9×9 Kovarianzmatrix der Fundamental-Matrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

vector_to_essential_matrix

Alternativen

rel_pose_to_fundamental_matrix

Modul

3D Metrology

gen_binocular_proj_rectification (: Map1, Map2 : FMatrix, CovFMat, Width1, Height1, Width2, Height2, SubSampling, Mapping : CovFMatRect, H1, H2)

Projektive Rektifizierung eines schwach kalibrierten Stereo-Bildpaares.

Eine binokulare Stereokonfiguration wird schwach kalibriert genannt, wenn die Fundamental-Matrix, welche die projektive Beziehung der beiden Bilder beschreibt, bekannt ist. Rektifizierung ist das Finden eines geeigneten Satzes von Transformationen, welcher die beiden Bilder derart transformiert, dass alle korrespondierenden Epipolargeraden identisch sind und parallel zur horizontalen Achse verlaufen. Rektifizierte Bilder erscheinen vergleichbar einer Stereoanordnung, bei der die linke und rechte Bildebene identisch sind und die relative Lage der Kamerazentren nur aus einer horizontalen Translation besteht. Eine Rektifizierung ist nur möglich, wenn beide Epipole außerhalb der Bildbereiche liegen.

Typischerweise wird die Fundamental-Matrix `FMatrix` vorher durch den Operator `match_fundamental_matrix_ransac` berechnet. Ausgehend von der Fundamental-Matrix werden die beiden Homographien (d.h. projektive Transformationen) `H1` und `H2` berechnet, welche die Rektifizierung der beiden Bilder beschreiben. Da die projektive Rektifizierung ein unterbestimmtes Problem darstellt, werden zusätzliche Bedingungen aufgestellt: Es werden diejenigen Transformation gewählt, welche die projektive Verzerrung minimal halten. Für diese Berechnung sind die Dimensionen der Bildbereiche `Width1`,`Height1`, `Width2` und `Height2` notwendig. Nach der Rektifizierung hat die Fundamental-Matrix immer folgende kanonische Form:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \quad .$$

Ist die Kovarianzmatrix `CovFMat` der Fundamental-Matrix `FMatrix` bekannt, so wird auch die Kovarianzmatrix `CovFMatRect` der obigen rektifizierten Fundamental-Matrix berechnet. Im Rahmen eines nachfolgenden Stereo-Matchings kann diese dazu dienen den Matching Prozess zu verbessern, da die Kovarianzmatrix den Bereich angibt, wo ein korrespondierender Match zu finden ist.

Ähnlich des verwandten Operators `gen_binocular_rectification_map` beschreiben die Ausgabebilder `Map1` und `Map2` die Transformationen, welche das Stereobildpaar rektifizieren. Der Parameter `Mapping` gibt an, ob für die Abbildung bilineare Interpolation (*'bilinear_map'*) verwendet werden soll, oder ob einfach der nächste Nachbar (*'nn_map'*) zu wählen ist. Die Größe bzw. Auflösung der Abbildungskarten ist über `SubSampling` einzustellen. Ein Wert von zum Beispiel 2 halbiert die Bildgröße. Wenn nur die Homographien und nicht die Abbildungskarten gewünscht sind, ist die Option `Mapping` auf *'no_map'* zu setzen. Unter dem Aspekt der Rechengeschwindigkeit ist diese Wahl zu bevorzugen, wenn für eine spezifische Stereokonfiguration nur ein Bildpaar rektifiziert werden soll. Wenn dagegen die Stereokonfiguration fix ist, so sollten nur einmal die Abbildungskarten berechnet werden; alle weiteren Bildpaare werden dann mit Hilfe des Operators `map_image` rektifiziert, was zum geringsten Rechenaufwand zur online Rektifizierung führt.

Bei der Verwendung der Abbildungskarten haben die transformierten Bilder dieselbe Größe wie die Karten. Jedes Pixel in der Karte enthält die Information, wie das neue Pixel an dieser Position generiert wird. Die Karten `Map1`, `Map2` sind einkanalige Bilder, wenn `Mapping` auf *'nn_map'* gesetzt ist, und bestehen aus fünf Kanälen, wenn *'bilinear_map'* gewünscht wird. Im ersten Kanal, welcher vom Datentyp int4 ist, wird die lineare Koordinate des im Originalbild referenzierten Pixels gespeichert. Mit `Mapping` gleich *'nn_map'* ist das referenzierte Pixel der nächste Nachbar des zurück transformierten Kartenpixels. Im Falle der bilinearen Interpolation ist das Referenzpixel die nächste obere, linke Ecke des zurück transformierten Pixels. Die folgende Abbildung zeigt die Ordnung der Pixel im Originalbild, welche den zurück transformierten Koordinaten am nächsten sind, wobei das Referenzpixel die Nummer zwei trägt:

2	3
4	5

.

Die Kanäle 2 bis 5, welche vom Datentyp uint2 sind, enthalten die Gewichte der entsprechenden Pixel für die bilineare Interpolation.

Für ein rektifiziertes Bildpaar kann die Disparität zwischen Bildpunkten oder Regionen mittels des Operators `binocular_disparity` berechnet werden. Im Gegensatz zum Stereo bei voll kalibrierten Kameras mit der Rektifizierung mittels `gen_binocular_rectification_map` kann bei schwach kalibrierten Kameras keine

metrische Tiefeninformation abgeleitet werden. Die Disparität gibt dann lediglich eine qualitative Tiefenordnung der betrachteten Szene wieder.

Parameter

- ▷ **Map1** (output_object) ..... image(-array) ⇝ *object :* int4 / uint2
  Bild, das die Rektifizierung des 1. Bildes codiert.
- ▷ **Map2** (output_object) ..... image(-array) ⇝ *object :* int4 / uint2
  Bild, das die Rektifizierung des 2. Bildes codiert.
- ▷ **FMatrix** (input_control) ..... hom_mat2d ⇝ *real* / integer
  Fundamental-Matrix.
- ▷ **CovFMat** (input_control) ..... number-array ⇝ *real* / integer
  9×9 Kovarianzmatrix der Fundamental-Matrix.
  Defaultwert: []
- ▷ **Width1** (input_control) ..... integer ⇝ *integer*
  Breite des 1. Bildes.
  Defaultwert: 512
  Wertevorschläge: `Width1` ∈ {128, 256, 512, 1024}
  Restriktion: Width1 > 0
- ▷ **Height1** (input_control) ..... integer ⇝ *integer*
  Höhe des 1. Bildes.
  Defaultwert: 512
  Wertevorschläge: `Height1` ∈ {128, 256, 512, 1024}
  Restriktion: Height1 > 0
- ▷ **Width2** (input_control) ..... integer ⇝ *integer*
  Breite des 2. Bildes.
  Defaultwert: 512
  Wertevorschläge: `Width2` ∈ {128, 256, 512, 1024}
  Restriktion: Width2 > 0
- ▷ **Height2** (input_control) ..... integer ⇝ *integer*
  Höhe des 2. Bildes.
  Defaultwert: 512
  Wertevorschläge: `Height2` ∈ {128, 256, 512, 1024}
  Restriktion: Height2 > 0
- ▷ **SubSampling** (input_control) ..... number ⇝ *integer* / real
  Unterabtastfaktor.
  Defaultwert: 1
  Werteliste: `SubSampling` ∈ {1, 2, 3, 1.5}
- ▷ **Mapping** (input_control) ..... string ⇝ *string*
  Typ des Mappings.
  Defaultwert: 'no_map'
  Werteliste: `Mapping` ∈ {'no_map', 'nn_map', 'bilinear_map'}
- ▷ **CovFMatRect** (output_control) ..... number-array ⇝ *real*
  9×9 Kovarianzmatrix der rektifizierten Fundamental-Matrix.
- ▷ **H1** (output_control) ..... hom_mat2d ⇝ *real*
  Projektive Transformation des 1. Bildes.
- ▷ **H2** (output_control) ..... hom_mat2d ⇝ *real*
  Projektive Transformation des 2. Bildes.

Beispiel

```
* Rectify an image pair using a map.
get_image_size (Image1, Width1, Height1)
get_image_size (Image2, Width2, Height2)
points_harris (Image1, 3, 1, 0.2, 10000, Row1, Col1)
points_harris (Image2, 3, 1, 0.2, 10000, Row2, Col2)
match_fundamental_matrix_ransac (Image1, Image2, Row1, Col1, Row2, Col2, \
                                 'ncc', 21, 0, 200, 20, 50, 0, 0.9, \
                                 'gold_standard', 0.3, 1, FMatrix, \
```

```
                                  CovFMat, Error, Points1, Points2)
gen_binocular_proj_rectification (Map1, Map2, FMatrix, [], Width1, \
                                  Height1, Width2, Height2, 1, \
                                  'bilinear_map', CovFMatRect, H1, H2)
map_image (Image1, Map1, Image1Rect)
map_image (Image2, Map2, Image2Rect)

* Rectify an image pair without using a map.
get_image_size (Image1, Width1, Height1)
get_image_size (Image2, Width2, Height2)
points_harris (Image1, 3, 1, 0.2, 10000, Row1, Col1)
points_harris (Image2, 3, 1, 0.2, 10000, Row2, Col2)
match_fundamental_matrix_ransac (Image1, Image2, Row1, Col1, Row2, Col2, \
                                 'ncc', 21, 0, 200, 20, 50, 0, 0.9, \
                                 'gold_standard', 0.3, 1, FMatrix, \
                                 CovFMat, Error, Points1, Points2)
gen_binocular_proj_rectification (Map1, Map2, FMatrix, [], Width1, \
                                  Height1, Width2, Height2, 1, \
                                  'no_map', CovFMatRect, H1, H2)
* Determine the maximum extent of the two rectified images.
projective_trans_point_2d (H1, [0,0,Height1,Height1], \
                           [0,Width1,0,Width1], [1,1,1,1], R1, C1, W1)
R1 := int(floor(R1/W1))
C1 := int(floor(C1/W1))
projective_trans_point_2d (H2, [0,0,Height2,Height2], \
                           [0,Width2,0,Width2], [1,1,1,1], R2, C2, W2)
R2 := int(floor(R2/W2))
C2 := int(floor(C2/W2))
WidthRect := max([C1,C2])
HeightRect := max([R1,R2])
projective_trans_image_size (Image1, Image1Rect, H1, 'bilinear', \
                             WidthRect, HeightRect, 'false')
projective_trans_image_size (Image2, Image2Rect, H2, 'bilinear', \
                             WidthRect, HeightRect, 'false')
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

match_fundamental_matrix_ransac, vector_to_fundamental_matrix

Nachfolger

map_image, projective_trans_image, binocular_disparity

Alternativen

gen_binocular_rectification_map

Literatur

J. Gluckmann and S.K. Nayar: „Rectifying transformations that minimize resampling effects“; IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2001, vol I, pages 111-117.

Modul

3D Metrology

gen_binocular_rectification_map (: Map1, Map2 : CamParam1, CamParam2, RelPose, SubSampling, Method, MapType : CamParamRect1, CamParamRect2, CamPoseRect1, CamPoseRect2, RelPoseRect)

Berechnet die Abbildung der beiden Bildebenen eines binokularen Stereosystems in eine gemeinsame Bildebene.

Stereobildpaare werden rektifiziert, indem eine Transformationsvorschrift für jede Bildebene bestimmt wird, so dass Paare von korrespondierenden Epipolarlinien kollinear sind und parallel zur X-Achse des Bildkoordinatensystems verlaufen. Diese Bildgeometrie ist eine Voraussetzung für eine effiziente Berechnung von Disparitäten oder Distanzen mit Operatoren wie z.B. binocular_disparity oder binocular_distance. Man kann sich die rektifizierten Bilder vorstellen, als würden sie von einem neuen Stereoaufbau aufgenommen, welchen man durch eine Rotation (und Translation bei telezentrischen Flächen- und Zeilenkameras) der ursprünglichen Kameras erhalten würde. Die Projektionszentren (bzw. bei telezentrischen Kameras die Richtung der optischen Achse) werden beibehalten. Bei perspektivischen Kameras werden zusätzlich die Bildebenen in eine gemeinsame Ebene transformiert. Letzteres ist gleichbedeutend mit der Gleichsetzung der Bildweiten und der Gleichrichtung der optischen Achsen. Für ein Stereosystem gemischten Typs (d.h. eine perspektivische und eine telezentrische Kamera) werden die Bildebenen auch, wie unten beschrieben, in eine gemeinsame Ebene transformiert.

Um die Abbildungsvorschrift für rektifizierte Bilder zu erhalten verlangt gen_binocular_rectification_map die internen Kameraparameter CamParam1 von Kamera 1 und CamParam2 von Kamera 2. Ebenso wird die relative 3D-Lage RelPose, ${}^{ccs1}\mathbf{P}_{ccs2}$, benötigt. Diese 3D-Lage beschreibt die Transformation eines Punktes vom Kamerakoordinatensystem 2 (*ccs2*) nach Kamerakoordinatensystem 1 (*ccs 1*), siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". Die Werte für diese Parameter kann man beispielsweise vom Operator calibrate_cameras erhalten.

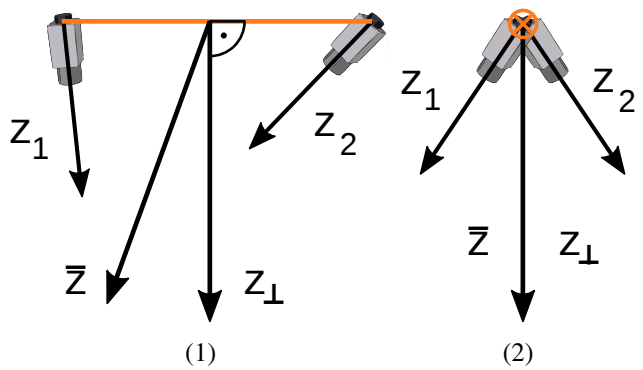
Die für die Rektifizierung modifizierten internen Kameraparameter werden in CamParamRect1 und CamParamRect2 ausgegeben. Die Rotation (und bei telezentrischen Kameras die Translation) des entzerrten Kamerasystems bezüglich des ursprünglichen Systems wird durch eine Punkttransformation vom entzerrten System ins originale System in CamPoseRect1 und CamPoseRect2 zurückgegeben. Diese 3D-Lagen sind in der Form ${}^{ccsX}\mathbf{P}_{ccsRX}$. Dabei steht *ccsX* für das Kamerakoordinatensystem von Kamera X und *ccsRX* für das entzerrtes Kamerakoordinatensystem von Kamera X. Schließlich gibt RelPoseRect die relative 3D-Lage der entzerrten Koordinatensysteme, ${}^{ccsR1}\mathbf{P}_{ccsR2}$, zurück. Diese 3D-Lage beschreibt eine Punkttransformation vom entzerrten Kamerakoordinatensystem 2 ins entzerrte Kamerakoordinatensystem 1.

Rektifizierungsmethode

Bei perspektivischen Flächenkameras ist dies lediglich eine Verschiebung in der x Richtung. Im Allgemeinen werden die Abbildungsvorschriften so festgelegt, dass sich das entzerrte Kamerasystem 1 links von dem entzerrten Kamerasystem 2 befindet. Dies bedeutet, dass das optische Zentrum der entzerrten Kamera 2 eine positive X-Koordinate im entzerrten Koordinatensystem von Kamera 1 besitzt.

Die Projektion auf eine gemeinsame Ebene hat mehrere Freiheitsgrade, welche implizit durch die Auswahl einer bestimmten Methode in Method bestimmt werden können:

- *'viewing_direction'* benutzt die Basisgerade als x-Achse der gemeinsamen Bildebene. Der Mittelwert der Blickrichtungen (z-Achsen) der beiden Kameras wird benutzt, um die x-z-Ebene des rektifizierten Systems aufzuspannen. Die resultierende rektifizierte z-Achse ist die Orientierung der gemeinsamen Bildebene, welche somit genau in dieser Ebene liegt und senkrecht zur Basisgeraden steht. In vielen Fällen weicht die resultierende rektifizierte z-Achse nicht weit vom Mittelwert der z-Achsen der alten Kameras ab. Die neue Bildweite wird so bestimmt, dass die alten Bildhauptpunkte von der neuen Bildebene gleichen Abstand haben. Die verschiedenen z-Achsenrichtungen sind im unteren Schema dargestellt.



Darstellung der verschiedenen z-Achsenrichtungen bei Verwendung von *'viewing_direction'*. (1): Blickrichtung der Basisgeraden (in orange) zugewandt. (2): Blickrichtung entlang der Basisgeraden (in die Seite hinein zeigend, in orange).

- *'geometric'* legt die Orientierung der gemeinsamen Bildebene durch das Kreuzprodukt der Richtungen der Basisgeraden und der Schnittgeraden der ursprünglichen Bildebenen fest. Die neue Bildweite wird so bestimmt, dass die alten Bildhauptpunkte von der neuen Bildebene gleichen Abstand haben.

Bei telezentrischen Flächen- und Zeilenkameras wird der Parameter `Method` ignoriert. Die relative Lage beider Kameras ist in einem solchen Aufbau nicht eindeutig bestimmt, da die Kameras entlang ihrer optischen Achse verschoben die gleichen Aufnahmen liefern. Um trotzdem eine absolute Distanzmessung zur Kamera zu ermöglichen, wird intern angenommen, dass die Kameras einen festen Abstand und Orientierung zueinander haben. Hierfür werden sie entlang ihrer optischen Achse derart verschoben, dass ihr Abstand ein Meter beträgt und die Gerade auf der beide Kameras liegen (Basis) die optischen Achsen beider Kameras im gleichen Winkel schneiden (Die Basis und die optischen Achsen bilden ein gleichschenkliges Dreieck). Die optischen Achsen bleiben dabei erhalten. Die relative Lage der rektifizierten Kameras `RelPoseRect` kann sich von der relativen Lage der ursprünglichen Kameras `RelPose` dadurch stark unterscheiden.

Für ein Stereosystem gemischten Typs (d.h. eine perspektivische und eine telezentrische Kamera) wird der Parameter `Method` ignoriert. Die gemeinsame rektifizierende Bildebene ist eindeutig aus der Geometrie der perspektivischen Kamera und der relativen Lage der beiden Kameras bestimmt. Die Normale der rektifizierenden Bildebene ist der Vektor, der vom Projektionszentrum der perspektivischen Kamera zum Punkt auf der optischen Achse der telezentrischen Kamera zeigt, der den kürzesten Abstand zum Projektionszentrum der perspektivischen Kamera besitzt. Dies ist gleichzeitig die z-Achse der rektifizierten perspektivischen Kamera. Die geometrische Basis des gemischten Stereosystems ist eine Gerade, die durch das Projektionszentrum der perspektivischen Kamera verläuft und dieselbe Richtung wie die z-Achse der telezentrischen Kamera besitzt, d.h. die Basis ist parallel zur Blickrichtung der telezentrischen Kamera. Die x-Achse der rektifizierten perspektivischen Kamera ist durch die Basis gegeben und die y-Achse wird so konstruiert, dass sich eine rechtshändiges Koordinatensystem ergibt. Um die telezentrische Kamera zu rektifizieren, muss ihre optische Achse auf die Basis verschoben werden und die Bildebene muss um $90°$ oder $-90°$ verkippt werden. Um dies zu erreichen, muss ein spezieller Typ von objektseitig telezentrischer Kamera, der diese spezielle Geometrie darstellen kann (gekennzeichnet durch den negativen Abstand der verkippten Bildebene `ImagePlaneDist`), für die rektifizierte telezentrische Kamera verwendet werden. Die Repräsentation dieses speziellen Kameratyps sollte als Blackbox betrachtet werden, da sie ausschließlich zum Zwecke der Rektifizierung in HALCON verwendet wird (und daher auch nicht in `camera_calibration` beschrieben wird). Die rektifizierte telezentrische Kamera hat dieselbe Orientierung wie die ursprüngliche telezentrische Kamera. Ihr Referenzpunkt ist aber auf einen Punkt auf der Basis verschoben worden.

Abbildungsvorschrift für die Bilder (Rectification Maps)

Die Abbildungsvorschriften für die Bilder von Kamera 1 und Kamera 2 werden in `Map1` und `Map2` in Form von Bildmatritzen zurückgegeben. Der Parameter `MapType` gibt an, in welchem Typ die Abbildungsvorschrift gespeichert werden sollen. Wurde *'nearest_neighbor'* gewählt, so bestehen beide Abbildungsvorschriften aus einem Bild mit einem Kanal, in dem für jedes Pixel des Ergebnisbildes die linearen Koordinaten desjenigen Pixels des Eingabebildes stehen, welches den transformierten Koordinaten am nächsten liegt. Wurde die bilineare Interpolation (*'bilinear'*) gewählt, so bestehen beide Abbildungsvorschriften aus einem Bild mit fünf Kanälen. Der erste Kanal enthält für jedes Pixel des Ergebnisbildes die linearen Koordinaten desjenigen Pixels des Eingabebildes, welches links oberhalb von den transformierten Koordinaten liegt. Die übrigen vier Kanäle enthalten die Gewichte der einzelnen vier Nachbarpixel der transformierten Koordinaten für die bilineare Interpolation in folgender Reihenfolge:

2	3
4	5

Der zweite Kanal z.B. enthält die Gewichte der Pixel, die links oberhalb der transformierten Koordinaten liegen. Wurde *'coord_map_sub_pix'* gewählt, bestehen beide Abbildungsvorschriften aus einem Vektorfeld, welches für jedes Pixel des Ergebnisbildes die subpixelgenauen Koordinaten im Eingabebild enthält.

Die Größe und Auflösung der Abbildungen und transformierten Bilder kann mit dem Parameter `SubSampling` bestimmt werden, welcher einen Abtastfaktor auf die Originalbilder beschreibt.

Wenn die erzeugte Abbildung in einem anderen Programm wiederverwendet werden soll, kann man sie mit dem Operator `write_image` als Mehrkanalbild abspeichern (Format: *'tiff'*).

Achtung

Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **Map1** (output_object) . . . . . (multichannel-)image ⇝ *object :* int4 / uint2 / vector_field
  Abbildungsvorschrift für Bilder von Kamera 1.
- ▷ **Map2** (output_object) . . . . . (multichannel-)image ⇝ *object :* int4 / uint2 / vector_field
  Abbildungsvorschrift für Bilder von Kamera 2.
- ▷ **CamParam1** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Parameter der Kamera 1.
- ▷ **CamParam2** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Parameter der Kamera 2.
- ▷ **RelPose** (input_control) . . . . . pose ⇝ *real* / integer
  Punkttransformation von Kamera 2 zu Kamera 1.
  Parameteranzahl: 7
- ▷ **SubSampling** (input_control) . . . . . real ⇝ *real*
  Abtastfaktor.
  Defaultwert: 1.0
  Wertevorschläge: `SubSampling` ∈ {0.5, 0.66, 1.0, 1.5, 2.0, 3.0, 4.0}
- ▷ **Method** (input_control) . . . . . string ⇝ *string*
  Art der Entzerrung.
  Defaultwert: 'viewing_direction'
  Werteliste: `Method` ∈ {'viewing_direction', 'geometric'}
- ▷ **MapType** (input_control) . . . . . string ⇝ *string*
  Art der Abbildungsvorschrift.
  Defaultwert: 'bilinear'
  Werteliste: `MapType` ∈ {'nearest_neighbor', 'bilinear', 'coord_map_sub_pix'}
- ▷ **CamParamRect1** (output_control) . . . . . campar ⇝ *real* / integer / string
  Interne Parameter der entzerrten Kamera 1.
- ▷ **CamParamRect2** (output_control) . . . . . campar ⇝ *real* / integer / string
  Interne Parameter der entzerrten Kamera 2.
- ▷ **CamPoseRect1** (output_control) . . . . . pose ⇝ *real* / integer
  Punkttransformation von der entzerrten Kamera 1 zur ursprünglichen Kamera 1.
  Parameteranzahl: 7
- ▷ **CamPoseRect2** (output_control) . . . . . pose ⇝ *real* / integer
  Punkttransformation von der entzerrten Kamera 2 zur ursprünglichen Kamera 2.
  Parameteranzahl: 7
- ▷ **RelPoseRect** (output_control) . . . . . pose ⇝ *real* / integer
  Punkttransformation von der entzerrten Kamera 2 zur entzerrten Kamera 1.
  Parameteranzahl: 7

Beispiel

```
* Set internal and external stereo parameters.
* Note that, typically, these values are the result of a prior
* calibration.
gen_cam_par_area_scan_division (0.01, -665, 5.2e-006, 5.2e-006, \
                                622, 517, 1280, 1024, CamParam1)
gen_cam_par_area_scan_division (0.01, -731, 5.2e-006, 5.2e-006, \
                                654, 519, 1280, 1024, CamParam2)
create_pose (0.1535,-0.0037,0.0447,0.17,319.84,359.89, \
             'Rp+T', 'gba', 'point', RelPose)

* Compute the mapping for rectified images.
gen_binocular_rectification_map (Map1, Map2, CamParam1, CamParam2, \
                                 RelPose, 1,'viewing_direction', 'bilinear',\
                                 CamParamRect1, CamParamRect2, \
```

```
                                CamPoseRect1, CamPoseRect2, \
                                RelPoseRect)

* Compute the disparities in online images.
while (1)
  grab_image_async (Image1, AcqHandle1, -1)
  map_image (Image1, Map1, ImageMapped1)

  grab_image_async (Image2, AcqHandle2, -1)
  map_image (Image2, Map2, ImageMapped2)

  binocular_disparity(ImageMapped1, ImageMapped2, Disparity, Score, \
                      'sad', 11, 11, 20, -40, 20, 2, 25, \
                      'left_right_check', 'interpolation')
endwhile
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_binocular_rectification_map` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`binocular_calibration`

Nachfolger

`map_image`

Alternativen

`gen_image_to_world_plane_map`

Siehe auch

`map_image`, `binocular_disparity`, `binocular_distance`, `binocular_disparity_mg`, `binocular_distance_mg`, `binocular_disparity_ms`, `binocular_distance_ms`, `gen_image_to_world_plane_map`, `contour_to_world_plane_xld`, `image_points_to_world_plane`

Modul

3D Metrology

intersect_lines_of_sight (: : CamParam1, CamParam2, RelPose, Row1, Col1, Row2, Col2 : X, Y, Z, Dist)

Berechnet einen Weltpunkt aus dem Schnitt von zwei Sichtlinien in einem binokularen Kamerasystem.

Ausgehend von zwei Sichtstrahlen von unterschiedlichen Kameras, welche durch die beiden Bildpunkte (`Row1`,`Col1`) in Kamera 1 und (`Row1`,`Col1`) in Kamera 2 festgelegt werden, errechnet `intersect_lines_of_sight` den 3D Schnittpunkt dieser Strahlen. Das binokulare Kamerasystem wird zum einen durch seine internen Parameter `CamParam1` der projektiven Kamera 1 und `CamParam2` der projektiven Kamera 2 definiert, und zum anderen durch seine externen Parameter `RelPose`. Letzteres ist eine Pose, die in der Form ${}^{ccs1}\mathbf{P}_{ccs2}$ erwartet wird. Sie beschreibt die Lage des Kamerakoordinatensystem 2 (*ccs2*) bezüglich des Kamerakoordinatensystems 1 (*ccs1*). Siehe auch Transformationen / Posen und „`Solution Guide III-C - 3D Vision`". Diese Kameraparameter erhält man beispielsweise vom Operator `calibrate_cameras`, wenn sich die Koordinaten der Bildpunkte (`Row1`,`Col1`) und (`Row2`,`Col2`) auf das entsprechende ursprüngliche Bildkoordinatensystem beziehen. Im Falle von entzerrten Bildkoordinaten (wie man sie von rektifizierten Bildern erhält) müssen die entsprechend entzerrten Kameraparameter übergeben werden, wie sie beispielsweise der Operator `gen_binocular_rectification_map` liefert. Der 'Schnittpunkt' schließlich wird als der Punkt mit

dem kleinsten Abstand zu beiden Sichtstrahlen definiert. Dieser Punkt wird in karthesischen Koordinaten (X,Y,Z) des Kamerasystems 1 zurückgegeben, seinen Abstand zu den Sichtstrahlen liefert der Parameter Dist.

Achtung

Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt.

Parameter

- ▷ **CamParam1** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Parameter der projektiven Kamera 1.
- ▷ **CamParam2** (input_control) ..... campar ⇝ *real* / integer / string
  Interne parameter der projektiven Kamera 2.
- ▷ **RelPose** (input_control) ..... pose ⇝ *real* / integer
  Punkttransformation von Kamera 2 zu Kamera 1.
  Parameteranzahl: 7
- ▷ **Row1** (input_control) ..... number(-array) ⇝ *real* / integer
  Zeilenkoordinate eines Punktes im Bild 1.
- ▷ **Col1** (input_control) ..... number(-array) ⇝ *real* / integer
  Spaltenkoordinate eines Punktes im Bild 1.
- ▷ **Row2** (input_control) ..... number(-array) ⇝ *real* / integer
  Zeilenkoordinate des korrespondierenden Punktes im Bild 2.
- ▷ **Col2** (input_control) ..... number(-array) ⇝ *real* / integer
  Spaltenkoordinate des korrespondierenden Punktes im Bild 2.
- ▷ **X** (output_control) ..... real(-array) ⇝ *real*
  X-Koordinate des Weltpunktes.
- ▷ **Y** (output_control) ..... real(-array) ⇝ *real*
  Y-Koordinate des Weltpunktes.
- ▷ **Z** (output_control) ..... real(-array) ⇝ *real*
  Z-Koordinate des Weltpunktes.
- ▷ **Dist** (output_control) ..... real(-array) ⇝ *real*
  Abstand des 3D Punktes zu den Sehstrahlen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert distance_to_disparity den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

binocular_calibration

Siehe auch

disparity_to_point_3d

Modul

3D Metrology

```
match_essential_matrix_ransac ( Image1, Image2 : : Rows1, Cols1,
    Rows2, Cols2, CamMat1, CamMat2, GrayMatchMethod, MaskSize,
    RowMove, ColMove, RowTolerance, ColTolerance, Rotation,
    MatchThreshold, EstimationMethod, DistanceThreshold,
    RandSeed : EMatrix, CovEMat, Error, Points1, Points2 )
```

Automatische Bestimmung der Essential-Matrix für ein Stereo-Bildpaar durch Zuordnung von Bildpunkten.

Ausgehend von einer Menge charakteristischer Punkte (Rows1,Cols1) und (Rows2,Cols2) in den beiden Eingabebildern Image1, Image2 und bekannten Kameramatrizen CamMat1, CamMat2 bestimmt

match_essential_matrix_ransac automatisch die Geometrie der Stereokonfiguration und findet die Korrespondenzen zwischen den Bildpunkten. Die Stereogeometrie ist durch die Essential-Matrix EMatrix gegeben, und alle Korrespondenzen müssen die Epipolargleichung erfüllen.

Der Operator match_essential_matrix_ransac ist für ein lineares Kameramodell konzipiert. Die internen Kameraparameter werden durch die Argumente CamMat1, CamMat2 übergeben, welche 3×3 obere Dreiecksmatrizen sind und eine affine Transformation beschreiben. Die Beziehung zwischen einem 3D Richtungsvektor (X,Y,1) und dessen (projektiven) 2D Bildkoordinaten (col,row,1) ist:

$$\begin{pmatrix} col \\ row \\ 1 \end{pmatrix} = CamMat \cdot \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \quad \text{mit} \quad CamMat = \begin{pmatrix} f/s_x & s & c_x \\ 0 & f/s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad .$$

Zu beachten ist hier die Reihenfolge (Spalte/Zeile), welche mit der (x,y) Notation im Kamerakoordinatensystem konform gehen muss. Die Brennweite ist mit f bezeichnet, s_x, s_y sind Skalierungsfaktoren, s beschreibt eine Scherung und (c_x, c_y) gibt den Bildhauptpunkt an. Hauptsächlich sind dies die Elemente aus den Kameraparametern, wie sie z.B. in calibrate_cameras verwendet werden. Alternativ kann eine Kameramatrix auch durch andere Elemente beschrieben werden, siehe z.B. stationary_camera_self_calibration. Multipliziert man die (projektiven) Bildkoordinaten mit der invertierten Kameramatrix, so erhält man die 3D Richtungsvektoren. Für bekannte Kameramatrizen schreibt sich die Epipolargleichung wie folgt:

$$\begin{pmatrix} X_2 \\ Y_2 \\ 1 \end{pmatrix}^T \cdot EMatrix \cdot \begin{pmatrix} X_1 \\ Y_1 \\ 1 \end{pmatrix} \quad = \quad 0 \quad .$$

Das Matchingverfahren beruht auf charakteristischen Punkten, welche mit Punktoperatoren, wie z.B. points_foerstner oder points_harris, extrahiert wurden. Die Bestimmung der Essential-Matrix und der Korrespondenzen erfolgt in 2 Schritten: Zuerst werden die Grauwertkorrelationen von Umgebungen der Eingabepunkte im ersten und zweiten Bild bestimmt und anhand dieser ein initiales Matching zwischen den Punkten ermittelt. Dann wird das RANSAC-Verfahren angewendet, um diejenige Essential-Matrix zu finden, welche die Anzahl der korrespondierenden Punktpaare unter Erfüllung der Epipolarbedingung maximiert.

Die Größe der Grauwertfenster beträgt MaskSize $\times$ MaskSize. Es können drei Metriken für die Korrelation gewählt werden. Hat GrayMatchMethod den Wert *'ssd'*, so wird das Quadrat der Grauwertdifferenzen verwendet, *'sad'* entspricht dem Betrag der Grauwertdifferenzen und *'ncc'* ist die normierte Kreuzkorrelation (siehe auch binocular_disparity). Diese Metrik wird über alle Punktpaare minimiert (*'ssd'*, *'sad'*) bzw. maximiert (*'ncc'*), ein so gefundenes Matching wird aber nur akzeptiert, falls der Wert der Metrik unter dem Wert von MatchThreshold (*'ssd'*, *'sad'*) bzw. über demselben (*'ncc'*) liegt.

Zur Geschwindigkeitssteigerung kann der Suchbereich für das Matching eingeschränkt werden. Nur Punkte innerhalb eines $2 \cdot$RowTolerance$\times 2 \cdot$ColTolerance Punkte großen Fensters werden betrachtet. Die Verschiebung des Mittelpunkts dieses Fensters im zweiten Bild gegenüber der Position des aktuellen Punktes im ersten Bild wird durch die Parameter RowMove und ColMove bestimmt.

Falls die Kameras nicht nur verschoben sind, sondern die zweite Kamera gegenüber der ersten um die optische Achse gedreht worden ist, kann im Parameter Rotation eine Schätzung des Drehwinkels bzw. ein Winkelintervall im Bogenmaß übergeben werden. Eine gute Schätzung des Winkels erhöht die Qualität des Grauwertwertmatchings. Falls sich die tatsächliche Rotation zu stark von der angegebenen Schätzung unterscheidet, schlägt das Matching typischerweise fehl. In diesem Fall sollte ein Winkelintervall angegeben werden. Je größer das angegebene Winkelintervall, desto langsamer läuft der Operator, denn für alle relevanten (endlich vielen) Winkel innerhalb des Intervalls wird das Verfahren komplett durchlaufen.

Ist das initiale Punktmatching bestimmt, wird es anschließend durch einen randomisierten Auswahlalgorithmus (RANSAC) zur Bestimmung der Essential-Matrix EMatrix benutzt. Dabei wird versucht die Matrix so zu wählen, dass sie bezüglich der Schranke DistanceThreshold zu möglichst vielen Punktpaaren konsistent ist.

Der Parameter EstimationMethod gibt an, ob die Kameras sich in einer besonderen relativen Orientierung zueinander befinden und bestimmt auch das Berechnungsverfahren. Für *'normalized_dlt'* und *'gold_standard'* kann die relative Lage der Kameras zueinander beliebig sein. Für *'trans_normalized_dlt'* und *'trans_gold_standard'* ist die relative Lage eine reine Translation. Für eine eindeutige Korrespondenzfindung ist die minimale Anzahl an notwendigen Punktkorrespondenzen im allgemeinen Fall sechs und im speziellen Fall der reinen Translation drei.

Wird *'normalized_dlt'* oder *'trans_normalized_dlt'* gewählt, so ist das Berechnungsverfahren ein lineares Verfahren. Wird *'gold_standard'* oder *'trans_gold_standard'* gewählt, so ist das Berechnungsverfahren im statistischen

Sinne optimal, und die Kovarianz der Essential-Matrix `CovEMat` wird zusätzlich berechnet. Es ist zu beachten, dass sich je nach gewähltem Verfahren nicht nur die berechnete Essential-Matrix unterscheidet, sondern auch die zugeordneten Punkte verschieden sein können.

Der Wert `Error` gibt die Qualität des Matchings an und ist der mittlere euklidische Abstand der Punkte zu ihren korrespondierenden Epipolarlinien gemessen in Pixeln.

Punktpaare, welche die Konsistenzbedingungen erfüllen, werden als Korrespondenzen akzeptiert. `Points1` enthält die Indizes der zugeordneten Eingabepunkte im ersten Bild, `Points2` die Indizes der dazu korrespondierenden Punkte im zweiten Bild.

Bei dem Operator `match_essential_matrix_ransac` ist folgender Spezialfall zu beachten: Liegen alle zugeordneten Raumpunkte in einer einzigen Ebene und zusätzlich alle näher zu einer der beiden Kameras, so gibt es insgesamt zwei Lösungen. Das heißt, dass in diesem Fall das Problem der Berechnung der Essential-Matrix nicht eindeutig lösbar ist. Es werden daher auch beide Lösungen ausgegeben. Das bedeutet, dass die Ausgabeparameter `EMatrix`, `CovEMat` und `Error` von doppelter Länge sind, wobei die Werte der zweiten Lösung an die Werte der ersten Lösung hinten angehängt sind.

Der Parameter `RandSeed` kann benutzt werden, um das randomisierte Verhalten des RANSAC-Verfahrens zu kontrollieren und somit reproduzierbare Ergebnisse zu erhalten. Wird `RandSeed` auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit `RandSeed` initialisiert wird. Ist `RandSeed` = *0*, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Parameter

- ▷ **`Image1`** (input_object) . . . singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 1.
- ▷ **`Image2`** (input_object) . . . singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 2.
- ▷ **`Rows1`** (input_control) . . . number-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 1.
  Restriktion: length(Rows1) >= 6 || length(Rows1) >= 3
- ▷ **`Cols1`** (input_control) . . . number-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 1.
  Restriktion: length(Cols1) == length(Rows1)
- ▷ **`Rows2`** (input_control) . . . number-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 2.
  Restriktion: length(Rows2) >= 6 || length(Rows2) >= 3
- ▷ **`Cols2`** (input_control) . . . number-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 2.
  Restriktion: length(Cols2) == length(Rows2)
- ▷ **`CamMat1`** (input_control) . . . hom_mat2d ⇝ *real* / integer
  Kameramatrix der 1. Kamera.
- ▷ **`CamMat2`** (input_control) . . . hom_mat2d ⇝ *real* / integer
  Kameramatrix der 2. Kamera.
- ▷ **`GrayMatchMethod`** (input_control) . . . string ⇝ *string*
  Metrik für den Vergleich der Grauwerte.
  Defaultwert: 'ssd'
  Werteliste: `GrayMatchMethod` ∈ {'ssd', 'sad', 'ncc'}
- ▷ **`MaskSize`** (input_control) . . . integer ⇝ *integer*
  Größe der Grauwertmaske.
  Defaultwert: 10
  Typischer Wertebereich: 3 ≤ `MaskSize` ≤ 15
  Restriktion: MaskSize >= 1
- ▷ **`RowMove`** (input_control) . . . integer ⇝ *integer*
  Mittlere Zeilenverschiebung.
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ `RowMove` ≤ 200

▷ **ColMove** (input_control) . . . . . integer ⇝ *integer*
Mittlere Spaltenverschiebung.
Defaultwert: 0
Typischer Wertebereich: $0 \leq$ ColMove ≤ 200

▷ **RowTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Höhe des Suchfensters für das Punktmatching.
Defaultwert: 200
Typischer Wertebereich: $50 \leq$ RowTolerance ≤ 200
Restriktion: RowTolerance >= 1

▷ **ColTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Breite des Suchfensters für das Punktmatching.
Defaultwert: 200
Typischer Wertebereich: $50 \leq$ ColTolerance ≤ 200
Restriktion: ColTolerance >= 1

▷ **Rotation** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Drehwinkelbereich.
Defaultwert: 0.0
Wertevorschläge: Rotation ∈ {0.0, 0.1, -0.1, 0.7854, 1.571, 3.142}

▷ **MatchThreshold** (input_control) . . . . . number ⇝ *integer* / real
Schwellenwert für Grauwertkorrespondenzen.
Defaultwert: 10
Wertevorschläge: MatchThreshold ∈ {10, 20, 50, 100, 0.9, 0.7}

▷ **EstimationMethod** (input_control) . . . . . string ⇝ *string*
Algorithmus zur Berechnung der Essential-Matrix und zur Auswahl spezieller relativer Orientierungen.
Defaultwert: 'normalized_dlt'
Werteliste: EstimationMethod ∈ {'normalized_dlt', 'gold_standard', 'trans_normalized_dlt', 'trans_gold_standard'}

▷ **DistanceThreshold** (input_control) . . . . . number ⇝ *real* / integer
Maximale Abweichung eines Punktes von seiner Epipolarlinie.
Defaultwert: 1
Typischer Wertebereich: $0.5 \leq$ DistanceThreshold ≤ 5
Restriktion: DistanceThreshold > 0

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Startwert für den Zufallszahlengenerator.
Defaultwert: 0

▷ **EMatrix** (output_control) . . . . . hom_mat2d ⇝ *real*
Berechnete Essential-Matrix.

▷ **CovEMat** (output_control) . . . . . real-array ⇝ *real*
9×9 Kovarianzmatrix der Essential-Matrix.

▷ **Error** (output_control) . . . . . real(-array) ⇝ *real*
Mittlerer, quadratischer Epipolar Abstand.

▷ **Points1** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 1.

▷ **Points2** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 2.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

points_foerstner, points_harris

Nachfolger

vector_to_essential_matrix

Siehe auch

match_fundamental_matrix_ransac, match_rel_pose_ransac, stationary_camera_self_calibration

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

3D Metrology

match_fundamental_matrix_distortion_ransac (Image1, Image2 : : Rows1, Cols1, Rows2, Cols2, GrayMatchMethod, MaskSize, RowMove, ColMove, RowTolerance, ColTolerance, Rotation, MatchThreshold, EstimationMethod, DistanceThreshold, RandSeed : FMatrix, Kappa, Error, Points1, Points2)

Automatische Bestimmung der Fundamental-Matrix und des radialen Verzeichnungskoeffizienten für ein Stereo-Bildpaar durch Zuordnung von Bildpunkten.

Ausgehend von einer Menge charakteristischer Punkte (Rows1, Cols1) und (Rows2, Cols2) in den beiden Eingabebildern Image1 und Image2, welche dieselbe Größe besitzen müssen, bestimmt match_fundamental_matrix_distortion_ransac automatisch die Korrespondenz der Punkte, die Fundamental-Matrix FMatrix und den radialen Verzeichnungskoeffizienten Kappa (κ), welche die Epipolargleichung

$$\begin{pmatrix} c_2 \\ r_2 \\ 1 \end{pmatrix}^T \cdot \texttt{FMatrix} \cdot \begin{pmatrix} c_1 \\ r_1 \\ 1 \end{pmatrix} = 0 \ .$$

am besten erfüllen. Hierbei sind (r_1, c_1) und (r_2, c_2) Bildpunkte, die durch Entzerrung der Eingabebildpunkte mit dem Divisionsmodell entstehen (siehe Kalibrierung):

$$r = \frac{\tilde{r}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)} \qquad c = \frac{\tilde{c}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)}$$

Hierbei bezeichnen $(\tilde{r}_1, \tilde{c}_1) = (\texttt{Rows1} - 0.5(h-1), \texttt{Cols1} - 0.5(w-1))$ und $(\tilde{r}_2, \tilde{c}_2) = (\texttt{Rows2} - 0.5(h-1), \texttt{Cols2} - 0.5(w-1))$ die verzerrten Bildpunkte relativ zum Bildmittelpunkt und w und h die Breite und Höhe der Eingabebilder. match_fundamental_matrix_distortion_ransac nimmt also an, dass der Hauptpunkt, d.h. das Zentrum der radialen Verzeichnungen, im Bildmittelpunkt liegt.

Das zurückgelieferte Kappa kann dazu verwendet werden, Kameraparameter zu konstruieren, die zur Entzerrung von Bildern oder Bildpunkten verwendet werden können (siehe change_radial_distortion_cam_par, change_radial_distortion_image und change_radial_distortion_points):

$$\texttt{CamPar} = [0.0, \texttt{Kappa}, 1.0, 1.0, 0.5(w-1), 0.5(h-1), w, h]$$

Zu beachten bei den obigen Formeln ist die Position von Spalte bzw. Zeile in den Punktkoordinaten. Da die Fundamental-Matrix die projektive Beziehung zweier Stereobilder im 3D Raum wiedergibt, muss die X/Y Notation mit dem Kamerakoordinatensystem konform sein. (X,Y) Koordinaten entsprechen daher einem (Column,Row) Paar.

Das Matchingverfahren beruht auf charakteristischen Punkten, welche mit Punktoperatoren, wie z.B. points_foerstner oder points_harris, extrahiert wurden. Die Bestimmung der Korrespondenzen erfolgt in 2 Schritten: Zuerst werden die Grauwertkorrelationen der Umgebungen der Eingabepunkte im ersten und zweiten Bild bestimmt und anhand dieser ein initiales Matching zwischen den Punkten ermittelt. Dann wird das RANSAC-Verfahren angewendet, um die Fundamental-Matrix und den radialen Verzeichnungskoeffizienten zu

finden, welche die Anzahl der korrespondierenden Punktpaare unter Erfüllung der Epipolarbedingung maximieren.

Die Größe der Grauwertfenster, die für das Matching verwendet werden, beträgt `MaskSize` × `MaskSize`. Es können drei Metriken für die Korrelation gewählt werden. Hat `GrayMatchMethod` den Wert *'ssd'*, so wird das Quadrat der Grauwertdifferenzen verwendet, *'sad'* entspricht dem Betrag der Grauwertdifferenzen und *'ncc'* ist die normierte Kreuzkorrelation (siehe auch `binocular_disparity`). Diese Metrik wird über alle Punktpaare minimiert (*'ssd'*, *'sad'*) bzw. maximiert (*'ncc'*), eine so gefundene Korrespondenz wird aber nur akzeptiert, falls der Wert der Metrik unter dem Wert von `MatchThreshold` (*'ssd'*, *'sad'*) bzw. über demselben (*'ncc'*) liegt.

Zur Geschwindigkeitssteigerung kann der Suchbereich für die Match-Kandidaten auf ein Rechteck, das durch Größe und Verschiebung spezifiziert wird, eingeschränkt werden. Nur Punkte innerhalb eines $2 \cdot$ `RowTolerance` $\times 2 \cdot$ `ColTolerance` Punkte großen Fensters werden betrachtet. Die Verschiebung des Mittelpunkts dieses Fensters im zweiten Bild gegenüber der Position des aktuellen Punktes im ersten Bild wird durch die Parameter `RowMove` und `ColMove` bestimmt.

Falls die Kameras nicht nur verschoben sind, sondern die zweite Kamera gegenüber der ersten um die optische Achse gedreht worden ist, kann im Parameter `Rotation` eine Schätzung des Drehwinkels bzw. ein Winkelintervall im Bogenmaß übergeben werden. Eine gute Schätzung des Winkels erhöht die Qualität des Grauwertwertmatchings. Falls sich die tatsächliche Rotation zu stark von der angegebenen Schätzung unterscheidet, schlägt das Matching typischerweise fehl. In diesem Fall sollte ein Winkelintervall angegeben werden. Je größer das angegebene Winkelintervall, desto langsamer läuft der Operator, denn für alle relevanten (automatisch bestimmten) Winkel innerhalb des Intervalls wird das Verfahren komplett durchlaufen.

Ist das initiale Punktmatching bestimmt, wird es anschließend durch einen randomisierten Auswahlalgorithmus (RANSAC) zur Bestimmung der Fundamental-Matrix `FMatrix` und des radialen Verzeichnungskoeffizienten `Kappa` benutzt. Dabei wird versucht, diese Parameter so zu wählen, dass sie bezüglich der Schranke `DistanceThreshold` zu möglichst vielen Punktpaaren konsistent ist.

Der Parameter `EstimationMethod` gibt an, ob die Kameras sich in einer besonderen relativen Orientierung zueinander befinden, und bestimmt auch das Berechnungsverfahren. Für *'linear'* und *'gold_standard'* kann die relative Lage der Kameras zueinander beliebig sein. *'trans_linear'* oder *'trans_gold_standard'* ist zu wählen, wenn die relative Lage der beiden Kameras eine reine Translation ist und die linke und rechte Kamera identisch sind. Für eine eindeutige Korrespondenzfindung ist die minimale Anzahl an notwendigen Punktkorrespondenzen im allgemeinen Fall neun und im speziellen Fall der reinen Translation vier.

Wird `EstimationMethod` auf *'linear'* oder *'trans_linear'* gesetzt, so ist das Berechnungsverfahren ein lineares Verfahren. Dieses Verfahren ist sehr schnell. Für den Fall der reinen Translation (`EstimationMethod` = *'trans_linear'*) liefert das lineare Verfahren genaue Ergebnisse für geringes bis mittleres Rauschen der Punktkoordinaten und für die meisten Verzeichnungen (außer sehr kleinen Verzeichnungen). Für eine beliebige Lage der Kameras (`EstimationMethod` = *'linear'*) liefert das lineare Verfahren nur für sehr geringes Rauschen der Punktkoordinaten und hinreichend große Verzeichnungen genaue Ergebnisse. Für `EstimationMethod` gleich *'gold_standard'* bzw. *'trans_gold_standard'* wird eine mathematisch optimale, dafür aber langsamere, Optimierung durchgeführt, die den geometrischen Reprojektionsfehler von rekonstruierten projektiven 3D Punkten minimiert. Für eine beliebige Lage der Kameras sollte im allgemeinen `EstimationMethod` = *'gold_standard'* gewählt werden.

Der Wert `Error` gibt die Qualität des Matchings an und ist der mittlere symmetrische euklidische Abstand der Punkte zu ihren korrespondierenden Epipolarlinien, gemessen in Pixeln.

Punktpaare, welche die Konsistenzbedingungen erfüllen, werden als Korrespondenzen akzeptiert. `Points1` enthält die Indizes der zugeordneten Eingabepunkte im ersten Bild, `Points2` die Indizes der dazu korrespondierenden Punkte im zweiten Bild.

Der Parameter `RandSeed` kann benutzt werden, um das randomisierte Verhalten des RANSAC-Verfahrens zu kontrollieren und somit reproduzierbare Ergebnisse zu erhalten. Wird `RandSeed` auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit `RandSeed` initialisiert wird. Ist `RandSeed` = *0*, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Parameter

▷ **Image1** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Eingabebild 1.

▷ **Image2** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Eingabebild 2.

▷ **Rows1** (input_control) . . . . . point.y-array ⇝ *real* / integer
Eingabepunkte in Bild 1 (Zeilenkoordinate).
Restriktion: length(Rows1) >= 9 || length(Rows1) >= 4

▷ **Cols1** (input_control) . . . . . point.x-array ⇝ *real* / integer
Eingabepunkte in Bild 1 (Spaltenkoordinate).
Restriktion: length(Cols1) == length(Rows1)

▷ **Rows2** (input_control) . . . . . point.y-array ⇝ *real* / integer
Eingabepunkte in Bild 2 (Zeilenkoordinate).
Restriktion: length(Rows2) >= 9 || length(Rows2) >= 4

▷ **Cols2** (input_control) . . . . . point.x-array ⇝ *real* / integer
Eingabepunkte in Bild 2 (Spaltenkoordinate).
Restriktion: length(Cols2) == length(Rows2)

▷ **GrayMatchMethod** (input_control) . . . . . string ⇝ *string*
Metrik für den Vergleich der Grauwerte.
Defaultwert: 'ncc'
Werteliste: `GrayMatchMethod` ∈ {'ncc', 'ssd', 'sad'}

▷ **MaskSize** (input_control) . . . . . integer ⇝ *integer*
Größe der Grauwertmasken.
Defaultwert: 10
Typischer Wertebereich: 3 ≤ `MaskSize` ≤ 15
Restriktion: MaskSize >= 1

▷ **RowMove** (input_control) . . . . . integer ⇝ *integer*
Mittlere Zeilenverschiebung.
Defaultwert: 0

▷ **ColMove** (input_control) . . . . . integer ⇝ *integer*
Mittlere Spaltenverschiebung.
Defaultwert: 0

▷ **RowTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Höhe des Suchfensters für das Punktmatching.
Defaultwert: 200
Restriktion: RowTolerance >= 1

▷ **ColTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Breite des Suchfensters für das Punktmatching.
Defaultwert: 200
Restriktion: ColTolerance >= 1

▷ **Rotation** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Schätzung für die Rotation des zweiten Bildes relativ zum ersten Bild.
Defaultwert: 0.0
Wertevorschläge: `Rotation` ∈ {0.0, 0.1, -0.1, 0.7854, 1.571, 3.142}

▷ **MatchThreshold** (input_control) . . . . . number ⇝ *integer* / real
Schwellenwert für Grauwertkorrespondenzen.
Defaultwert: 0.7
Wertevorschläge: `MatchThreshold` ∈ {0.9, 0.7, 0.5, 10, 20, 50, 100}

▷ **EstimationMethod** (input_control) . . . . . string ⇝ *string*
Algorithmus zur Berechnung der Fundamental-Matrix und zur Auswahl spezieller relativer Orientierungen.
Defaultwert: 'gold_standard'
Werteliste: `EstimationMethod` ∈ {'linear', 'gold_standard', 'trans_linear', 'trans_gold_standard'}

▷ **DistanceThreshold** (input_control) . . . . . number ⇝ *real* / integer
Maximale Abweichung eines Punktes von seiner Epipolarlinie.
Defaultwert: 1
Restriktion: DistanceThreshold > 0

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Startwert für den Zufallszahlengenerator.
Defaultwert: 0

▷ **FMatrix** (output_control) . . . . . hom_mat2d ⇝ *real*
Berechnete Fundamental-Matrix.

▷ **Kappa** (output_control) ..... real ⇝ *real*
Berechneter Verzeichnungskoeffizient.

▷ **Error** (output_control) ..... real ⇝ *real*
Mittlerer quadratischer Epipolarabstand.

▷ **Points1** (output_control) ..... integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 1.

▷ **Points2** (output_control) ..... integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 2.

Beispiel

```
points_foerstner (Image1, 1, 2, 3, 200, 0.1, 'gauss', 'true', \
                  Rows1, Cols1, _, _, _, _, _, _, _, _)
points_foerstner (Image2, 1, 2, 3, 200, 0.1, 'gauss', 'true', \
                  Rows2, Cols2, _, _, _, _, _, _, _, _)
match_fundamental_matrix_distortion_ransac (Image1, Image2, \
                                            Rows1, Cols1, Rows2, \
                                            Cols2, 'ncc', 10, 0, 0, \
                                            100, 200, 0, 0.5, \
                                            'trans_gold_standard', \
                                            1, 42, FMatrix, Kappa, \
                                            Error, Points1, Points2)
get_image_size (Image1, Width, Height)
CamParDist := ['area_scan_division',0.0,Kappa,1.0,1.0,\
               0.5*(Width-1),0.5*Height-1,Width,Height]
change_radial_distortion_cam_par ('fixed', CamParDist, 0, CamPar)
change_radial_distortion_image (Image1, Image1, Image1Rect, \
                                CamParDist, CamPar)
change_radial_distortion_image (Image2, Image2, Image2Rect, \
                                CamParDist, CamPar)
gen_binocular_proj_rectification (Map1, Map2, FMatrix, [], Width, \
                                  Height, Width, Height, 1, \
                                  'bilinear_map', _, H1, H2)
map_image (Image1Rect, Map1, Image1Mapped)
map_image (Image2Rect, Map2, Image2Mapped)
binocular_disparity_mg (Image1Mapped, Image2Mapped, Disparity, \
                        Score, 1, 30, 8, 0, 'false', \
                        'default_parameters', 'fast_accurate')
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

points_foerstner, points_harris

Nachfolger

vector_to_fundamental_matrix_distortion, change_radial_distortion_cam_par, change_radial_distortion_image, change_radial_distortion_points, gen_binocular_proj_rectification

Siehe auch

match_fundamental_matrix_ransac, match_essential_matrix_ransac, match_rel_pose_ransac, proj_match_points_ransac, calibrate_cameras

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

3D Metrology

match_fundamental_matrix_ransac (Image1, Image2 : : Rows1, Cols1, Rows2, Cols2, GrayMatchMethod, MaskSize, RowMove, ColMove, RowTolerance, ColTolerance, Rotation, MatchThreshold, EstimationMethod, DistanceThreshold, RandSeed : FMatrix, CovFMat, Error, Points1, Points2)

Automatische Bestimmung der Fundamental-Matrix für ein Stereo-Bildpaar durch Zuordnung von Bildpunkten.

Ausgehend von einer Menge charakteristischer Punkte (Rows1,Cols1) und (Rows2,Cols2) in den beiden Eingabebildern Image1 und Image2 bestimmt match_fundamental_matrix_ransac automatisch die Korrespondenz der Punkte und die Fundamental-Matrix FMatrix, welche die Epipolargleichung erfüllen:

$$\begin{pmatrix} \texttt{Cols2} \\ \texttt{Rows2} \\ 1 \end{pmatrix}^T \cdot \texttt{FMatrix} \cdot \begin{pmatrix} \texttt{Cols1} \\ \texttt{Rows1} \\ 1 \end{pmatrix} = 0 \quad .$$

Zu beachten ist hier die Position von Spalte bzw. Zeile in den Punktkoordinaten. Da die Fundamental-Matrix die projektive Beziehung zweier Stereobilder im 3D Raum wiedergibt, muss die X/Y Notation mit dem Kamerakoordinatensystem konform sein. (X,Y) Koordinaten entsprechen daher einem (Column,Row) Paar.

Das Matchingverfahren beruht auf charakteristischen Punkten, welche mit Punktoperatoren, wie z.B. points_foerstner oder points_harris, extrahiert wurden. Die Bestimmung der Fundamental-Matrix und der Korrespondenzen erfolgt in 2 Schritten: Zuerst werden die Grauwertkorrelationen von Umgebungen der Eingabepunkte im ersten und zweiten Bild bestimmt und anhand dieser ein initiales Matching zwischen den Punkten ermittelt. Dann wird das RANSAC-Verfahren angewendet, um diejenige Fundamental-Matrix zu finden, welche die Anzahl der korrespondierenden Punktpaare unter Erfüllung der Epipolarbedingung maximiert.

Die Größe der Grauwertfenster beträgt MaskSize $\times$ MaskSize. Es können drei Metriken für die Korrelation gewählt werden. Hat GrayMatchMethod den Wert *'ssd'*, so wird das Quadrat der Grauwertdifferenzen verwendet, *'sad'* entspricht dem Betrag der Grauwertdifferenzen und *'ncc'* ist die normierte Kreuzkorrelation (siehe auch binocular_disparity). Diese Metrik wird über alle Punktpaare minimiert (*'ssd'*, *'sad'*) bzw. maximiert (*'ncc'*), ein so gefundenes Matching wird aber nur akzeptiert, falls der Wert der Metrik unter dem Wert von MatchThreshold (*'ssd'*, *'sad'*) bzw. über demselben (*'ncc'*) liegt.

Zur Geschwindigkeitssteigerung kann der Suchbereich für das Matching eingeschränkt werden. Nur Punkte innerhalb eines $2 \cdot$RowTolerance$\times 2 \cdot$ColTolerance Punkte großen Fensters werden betrachtet. Die Verschiebung des Mittelpunkts dieses Fensters im zweiten Bild gegenüber der Position des aktuellen Punktes im ersten Bild wird durch die Parameter RowMove und ColMove bestimmt.

Falls die Kameras nicht nur verschoben sind, sondern die zweite Kamera gegenüber der ersten um die optische Achse gedreht worden ist, kann im Parameter Rotation eine Schätzung des Drehwinkels bzw. ein Winkelintervall im Bogenmaß übergeben werden. Eine gute Schätzung des Winkels erhöht die Qualität des Grauwertwertmatchings. Falls sich die tatsächliche Rotation zu stark von der angegebenen Schätzung unterscheidet, schlägt das Matching typischerweise fehl. In diesem Fall sollte ein Winkelintervall angegeben werden. Je größer das angegebene Winkelintervall, desto langsamer läuft der Operator, denn für alle relevanten (endlich vielen) Winkel innerhalb des Intervalls wird das Verfahren komplett durchlaufen.

Ist das initiale Punktmatching bestimmt, wird es anschließend durch einen randomisierten Auswahlalgorithmus (RANSAC) zur Bestimmung der Fundamental-Matrix FMatrix benutzt. Dabei wird versucht die Matrix so zu wählen, dass sie bezüglich der Schranke DistanceThreshold zu möglichst vielen Punktpaaren konsistent ist.

Der Parameter EstimationMethod gibt an, ob die Kameras sich in einer besonderen relativen Orientierung zueinander befinden, und bestimmt auch das Berechnungsverfahren. Für *'normalized_dlt'* und *'gold_standard'* kann die relative Lage der Kameras zueinander beliebig sein. *'trans_normalized_dlt'* oder *'trans_gold_standard'* ist zu wählen, wenn die relative Lage der beiden Kameras eine reine Translation ist, und linke und rechte Kamera identisch sind. Für eine eindeutige Korrespondenzfindung ist die minimale Anzahl an notwendigen Punktkorrespondenzen im allgemeinen Fall acht und im speziellen Fall der reinen Translation drei.

Wird *'normalized_dlt'* oder *'trans_normalized_dlt'* gewählt, so ist das Berechnungsverfahren ein lineares Verfahren. Wird *'gold_standard'* oder *'trans_gold_standard'* gewählt, so ist das Berechnungsverfahren im statistischen Sinne optimal, und die Kovarianz der Fundamental-Matrix `CovFMat` wird zusätzlich berechnet.

Der Wert `Error` gibt die Qualität des Matchings an und ist der mittlere euklidische Abstand der Punkte zu ihren korrespondierenden Epipolarlinien gemessen in Pixeln.

Punktpaare, welche die Konsistenzbedingungen erfüllen, werden als Korrespondenzen akzeptiert. `Points1` enthält die Indizes der zugeordneten Eingabepunkte im ersten Bild, `Points2` die Indizes der dazu korrespondierenden Punkte im zweiten Bild.

Der Parameter `RandSeed` kann benutzt werden, um das randomisierte Verhalten des RANSAC-Verfahrens zu kontrollieren und somit reproduzierbare Ergebnisse zu erhalten. Wird `RandSeed` auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit `RandSeed` initialisiert wird. Ist `RandSeed` = *0*, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Parameter

- ▷ **`Image1`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 1.
- ▷ **`Image2`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 2.
- ▷ **`Rows1`** (input_control) ..... number-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 1.
  Restriktion: length(Rows1) >= 8 || length(Rows1) >= 3
- ▷ **`Cols1`** (input_control) ..... number-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 1.
  Restriktion: length(Cols1) == length(Rows1)
- ▷ **`Rows2`** (input_control) ..... number-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 2.
  Restriktion: length(Rows2) >= 8 || length(Rows2) >= 3
- ▷ **`Cols2`** (input_control) ..... number-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 2.
  Restriktion: length(Cols2) == length(Rows2)
- ▷ **`GrayMatchMethod`** (input_control) ..... string ⇝ *string*
  Metrik für den Vergleich der Grauwerte.
  Defaultwert: 'ssd'
  Werteliste: `GrayMatchMethod` ∈ {'ssd', 'sad', 'ncc'}
- ▷ **`MaskSize`** (input_control) ..... integer ⇝ *integer*
  Größe der Grauwertmaske.
  Defaultwert: 10
  Typischer Wertebereich: 3 ≤ `MaskSize` ≤ 15
  Restriktion: MaskSize >= 1
- ▷ **`RowMove`** (input_control) ..... integer ⇝ *integer*
  Mittlere Zeilenverschiebung.
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ `RowMove` ≤ 200
- ▷ **`ColMove`** (input_control) ..... integer ⇝ *integer*
  Mittlere Spaltenverschiebung.
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ `ColMove` ≤ 200
- ▷ **`RowTolerance`** (input_control) ..... integer ⇝ *integer*
  Halbe Höhe des Suchfensters für das Punktmatching.
  Defaultwert: 200
  Typischer Wertebereich: 50 ≤ `RowTolerance` ≤ 200
  Restriktion: RowTolerance >= 1
- ▷ **`ColTolerance`** (input_control) ..... integer ⇝ *integer*
  Halbe Breite des Suchfensters für das Punktmatching.
  Defaultwert: 200
  Typischer Wertebereich: 50 ≤ `ColTolerance` ≤ 200
  Restriktion: ColTolerance >= 1

▷ **Rotation** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Drehwinkelbereich.
Defaultwert: 0.0
Wertevorschläge: `Rotation` ∈ {0.0, 0.1, -0.1, 0.7854, 1.571, 3.142}

▷ **MatchThreshold** (input_control) . . . . . number ⇝ *integer* / real
Schwellenwert für Grauwertkorrespondenzen.
Defaultwert: 10
Wertevorschläge: `MatchThreshold` ∈ {10, 20, 50, 100, 0.9, 0.7}

▷ **EstimationMethod** (input_control) . . . . . string ⇝ *string*
Algorithmus zur Berechnung der Fundamental-Matrix und zur Auswahl spezieller relativer Orientierungen.
Defaultwert: 'normalized_dlt'
Werteliste: `EstimationMethod` ∈ {'normalized_dlt', 'gold_standard', 'trans_normalized_dlt', 'trans_gold_standard'}

▷ **DistanceThreshold** (input_control) . . . . . number ⇝ *real* / integer
Maximale Abweichung eines Punktes von seiner Epipolarlinie.
Defaultwert: 1
Typischer Wertebereich: 0.5 ≤ `DistanceThreshold` ≤ 5
Restriktion: DistanceThreshold > 0

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Startwert für den Zufallszahlengenerator.
Defaultwert: 0

▷ **FMatrix** (output_control) . . . . . hom_mat2d ⇝ *real*
Berechnete Fundamental-Matrix.

▷ **CovFMat** (output_control) . . . . . real-array ⇝ *real*
9×9 Kovarianzmatrix der Fundamental-Matrix.

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlerer Epipolar-Abstands-Fehler.

▷ **Points1** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 1.

▷ **Points2** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 2.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`points_foerstner`, `points_harris`

Nachfolger

`vector_to_fundamental_matrix`, `gen_binocular_proj_rectification`

Siehe auch

`match_essential_matrix_ransac`, `match_rel_pose_ransac`, `proj_match_points_ransac`

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

3D Metrology

```
match_rel_pose_ransac ( Image1, Image2 : : Rows1, Cols1, Rows2,
    Cols2, CamPar1, CamPar2, GrayMatchMethod, MaskSize, RowMove,
    ColMove, RowTolerance, ColTolerance, Rotation, MatchThreshold,
    EstimationMethod, DistanceThreshold, RandSeed : RelPose,
    CovRelPose, Error, Points1, Points2 )
```

Automatische Bestimmung der relativen Orientierung zweier Kameras durch Zuordnung von Bildpunkten.

Ausgehend von einer Menge charakteristischer Punkte (Rows1, Cols1) und (Rows2, Cols2) in den beiden Eingabebildern Image1, Image2 und bekannten Kameramaparametern CamPar1, CamPar2 bestimmt match_rel_pose_ransac automatisch die Geometrie der Stereokonfiguration und findet die Korrespondenzen zwischen den Bildpunkten. Die Stereogeometrie ist durch die relative Orientierung RelPose, auch relative Lage genannt, der Kameras zueinander gegeben, und alle Korrespondenzen müssen die Epipolargleichung erfüllen. RelPose gibt die Orientierung der Kamera 1 relative zur Kamera 2 an (Siehe create_pose für weitere Information über die Beschreibung von Orientierungen.). Dies steht in Übereinstimmung mit der expliziten Kalibrierung einer Stereokonfiguration mit dem Operator calibrate_cameras. Sei R, t die Rotation beziehungsweise Translation der relativen Orientierung. Die Essential-Matrix E ist dann durch $E = ([t]_\times R)^T$ definiert, wobei $[t]_\times$ eine 3×3 schiefsymmetrische Matrix ist, welche das Kreuzprodukt mit dem Vektor t beschreibt. Die gesuchte Lage kann mittels der Epipolar-Gleichungen bestimmt werden:

$$\begin{pmatrix} X_2 \\ Y_2 \\ 1 \end{pmatrix}^T \cdot ([t]_\times R)^T \cdot \begin{pmatrix} X_1 \\ Y_1 \\ 1 \end{pmatrix} = 0 \quad \text{mit} \quad [t]_\times = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix} \quad .$$

Beachte, dass die Essential-Matrix eine homogene Größe ist, d.h. sie ist bis auf einen Skalierungsfaktor bestimmt. Somit kann der Translationsvektor der relativen Orientierung auch nur bis auf eine Skalierung berechnet werden. Der Operator wird diesen Vektor auf die Länge eins normiert berechnen. Als Konsequenz ergibt sich, dass eine anschließende Rekonstruktion der betrachteten Szene, z.B. mittels des Operators vector_to_rel_pose, auch nur bis auf einen globalen Skalierungsfaktor durchgeführt werden kann.

Der Operator match_rel_pose_ransac ist für ein nichtlineares Kameramodell konzipiert und beschreibt somit auch radiale Linsenverzeichnungen. Dies steht im Gegensatz zum Operator match_essential_matrix_ransac, welcher nur Geraden erhaltene, d.h. lineare, Kameras beschreibt. Kameraparameter werden mit den Argumenten CamPar1 und CamPar2 übergeben. Die 3D Richtungsvektoren $(X_1, Y_1, 1)$ und $(X_2, Y_2, 1)$ werden aus den Punktkoordinaten (Rows1,Cols1) und (Rows2,Cols2) durch Invertierung der Kameraprojektion berechnet (siehe Kalibrierung).

Das Matchingverfahren beruht auf charakteristischen Punkten, welche mit Punktoperatoren wie z.B. mit points_foerstner oder points_harris extrahiert wurden. Die Bestimmung der relativen Lage und der Korrespondenzen erfolgt in 2 Schritten: Zuerst werden die Grauwertkorrelationen von Umgebungen der Eingabepunkte im ersten und zweiten Bild bestimmt und anhand dieser ein initiales Matching zwischen den Punkten ermittelt. Dann wird das RANSAC-Verfahren angewendet, um diejenige relativen Lage zu finden, welche die Anzahl der korrespondierenden Punktpaare unter Erfüllung der Epipolarbedingung maximiert.

Die Größe der Grauwertfenster beträgt MaskSize × MaskSize. Es können drei Metriken für die Korrelation gewählt werden. Hat GrayMatchMethod den Wert *'ssd'*, so wird das Quadrat der Grauwertdifferenzen verwendet, *'sad'* entspricht dem Betrag der Grauwertdifferenzen und *'ncc'* ist die normierte Kreuzkorrelation (siehe auch binocular_disparity). Diese Metrik wird über alle Punktpaare minimiert (*'ssd'*, *'sad'*) bzw. maximiert (*'ncc'*), ein so gefundenes Matching wird aber nur akzeptiert, falls der Wert der Metrik unter dem Wert von MatchThreshold (*'ssd'*, *'sad'*) bzw. über demselben (*'ncc'*) liegt.

Zur Geschwindigkeitssteigerung kann der Suchbereich für das Matching eingeschränkt werden. Nur Punkte innerhalb eines 2·RowTolerance×2·ColTolerance Punkte großen Fensters werden betrachtet. Die Verschiebung des Mittelpunkts dieses Fensters im zweiten Bild gegenüber der Position des aktuellen Punktes im ersten Bild wird durch die Parameter RowMove und ColMove bestimmt.

Falls die Kameras nicht nur verschoben sind, sondern die zweite Kamera gegenüber der ersten um die optische Achse gedreht worden ist, kann im Parameter Rotation eine Schätzung des Drehwinkels bzw. ein Winkelintervall im Bogenmaß übergeben werden. Eine gute Schätzung des Winkels erhöht die Qualität des Grauwertwertmatchings. Falls sich die tatsächliche Rotation zu stark von der angegebenen Schätzung unterscheidet, schlägt das Matching typischerweise fehl. In diesem Fall sollte ein Winkelintervall angegeben werden. Je größer das angegebene Winkelintervall, desto langsamer läuft der Operator, denn für alle relevanten (endlich vielen) Winkel innerhalb des Intervalls wird das Verfahren komplett durchlaufen.

Ist das initiale Punktmatching bestimmt, wird es anschließend durch einen randomisierten Auswahlalgorithmus (RANSAC) zur Bestimmung der relativen Lage `RelPose` benutzt. Dabei wird diejenige relative Lage bestimmt, welche mit einer maximalen Anzahl an Korrespondenzen konsistent ist. Damit ein Punktpaar als Korrespondenz akzeptiert wird, darf der Abstand der Punkte von der korrespondierenden Epipolarlinie nicht größer als `DistanceThreshold` werden.

Der Parameter `EstimationMethod` gibt an, ob die Kameras sich in einer besonderen relativen Orientierung zueinander befinden, und bestimmt auch das Berechnungsverfahren. Für *'normalized_dlt'* und *'gold_standard'* kann die relative Lage der Kameras zueinander beliebig sein. Für *'trans_normalized_dlt'* und *'trans_gold_standard'* ist die relative Lage eine reine Translation. Für eine eindeutige Korrespondenzfindung ist die minimale Anzahl an notwendigen Punktkorrespondenzen im allgemeinen Fall sechs und im speziellen Fall der reinen Translation drei.

Wird *'normalized_dlt'* oder *'trans_normalized_dlt'* gewählt, so ist das Berechnungsverfahren ein lineares Verfahren. Wird *'gold_standard'* oder *'trans_gold_standard'* gewählt, so ist das Berechnungsverfahren im statistischen Sinne optimal, und die Kovarianz der relativen Lage `CovRelPose` wird zusätzlich berechnet. Es ist zu beachten, dass sich je nach gewähltem Verfahren nicht nur die berechnete Lage unterscheidet, sondern auch die zugeordneten Punkte verschieden sein können.

Der Wert `Error` gibt die Qualität des Matchings an und ist der mittlere euklidische Abstand der Punkte zu ihren korrespondierenden Epipolarlinien gemessen in Pixeln.

Punktpaare, welche die Konsistenzbedingungen erfüllen, werden als Korrespondenzen akzeptiert. `Points1` enthält die Indizes der zugeordneten Eingabepunkte im ersten Bild, `Points2` die Indizes der dazu korrespondierenden Punkte im zweiten Bild.

Bei dem Operator `match_rel_pose_ransac` ist folgender Spezialfall zu beachten: Liegen alle zugeordneten Raumpunkte in einer einzigen Ebene und zusätzlich alle näher zu einer der beiden Kameras, so gibt es insgesamt zwei Lösungen. Das heißt, dass in diesem Fall das Problem der Berechnung der relativen Lage nicht eindeutig lösbar ist. Es werden daher auch beide Lösungen ausgegeben. Das bedeutet, dass die Ausgabeparameter `RelPose`, `CovRelPose` und `Error` von doppelter Länge sind, wobei die Werte der zweiten Lösung an die Werte der ersten Lösung hinten angehängt sind.

Der Parameter `RandSeed` kann benutzt werden, um das randomisierte Verhalten des RANSAC-Verfahrens zu kontrollieren und somit reproduzierbare Ergebnisse zu erhalten. Wird `RandSeed` auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit `RandSeed` initialisiert wird. Ist `RandSeed` $= 0$, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Parameter

- ▷ **`Image1`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 1.
- ▷ **`Image2`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 2.
- ▷ **`Rows1`** (input_control) . . . . . number-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 1.
  Restriktion: length(Rows1) >= 6 || length(Rows1) >= 3
- ▷ **`Cols1`** (input_control) . . . . . number-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 1.
  Restriktion: length(Cols1) == length(Rows1)
- ▷ **`Rows2`** (input_control) . . . . . number-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 2.
  Restriktion: length(Rows2) >= 6 || length(Rows2) >= 3
- ▷ **`Cols2`** (input_control) . . . . . number-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 2.
  Restriktion: length(Cols2) == length(Rows2)
- ▷ **`CamPar1`** (input_control) . . . . . campar ⇝ *real* / integer / string
  Kameraparameter der 1. Kamera.
- ▷ **`CamPar2`** (input_control) . . . . . campar ⇝ *real* / integer / string
  Kameraparameter der 2. Kamera.
- ▷ **`GrayMatchMethod`** (input_control) . . . . . string ⇝ *string*
  Metrik für den Vergleich der Grauwerte.
  Defaultwert: 'ssd'
  Werteliste: `GrayMatchMethod` ∈ {'ssd', 'sad', 'ncc'}

▷ **MaskSize** (input_control) . . . . . integer ⇝ *integer*
Größe der Grauwertmaske.
Defaultwert: 10
Typischer Wertebereich: $3 \leq$ `MaskSize` ≤ 15
Restriktion: MaskSize >= 1

▷ **RowMove** (input_control) . . . . . integer ⇝ *integer*
Mittlere Zeilenverschiebung.
Defaultwert: 0
Typischer Wertebereich: $0 \leq$ `RowMove` ≤ 200

▷ **ColMove** (input_control) . . . . . integer ⇝ *integer*
Mittlere Spaltenverschiebung.
Defaultwert: 0
Typischer Wertebereich: $0 \leq$ `ColMove` ≤ 200

▷ **RowTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Höhe des Suchfensters für das Punktmatching.
Defaultwert: 200
Typischer Wertebereich: $50 \leq$ `RowTolerance` ≤ 200
Restriktion: RowTolerance >= 1

▷ **ColTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Breite des Suchfensters für das Punktmatching.
Defaultwert: 200
Typischer Wertebereich: $50 \leq$ `ColTolerance` ≤ 200
Restriktion: ColTolerance >= 1

▷ **Rotation** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Drehwinkelbereich.
Defaultwert: 0.0
Wertevorschläge: `Rotation` $\in$ {0.0, 0.1, -0.1, 0.7854, 1.571, 3.142}

▷ **MatchThreshold** (input_control) . . . . . number ⇝ *integer* / real
Schwellenwert für Grauwertkorrespondenzen.
Defaultwert: 10
Wertevorschläge: `MatchThreshold` $\in$ {10, 20, 50, 100, 0.9, 0.7}

▷ **EstimationMethod** (input_control) . . . . . string ⇝ *string*
Algorithmus zur Berechnung der relativen Lage und zur Auswahl spezieller relativer Lagen.
Defaultwert: 'normalized_dlt'
Werteliste: `EstimationMethod` $\in$ {'normalized_dlt', 'gold_standard', 'trans_normalized_dlt', 'trans_gold_standard'}

▷ **DistanceThreshold** (input_control) . . . . . number ⇝ *real* / integer
Maximale Abweichung eines Punktes von seiner Epipolarlinie.
Defaultwert: 1
Typischer Wertebereich: $0.5 \leq$ `DistanceThreshold` ≤ 5
Restriktion: DistanceThreshold > 0

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Startwert für den Zufallszahlengenerator.
Defaultwert: 0

▷ **RelPose** (output_control) . . . . . pose ⇝ *real* / integer
Berechnete relative Orientierung/Lage der Kameras zueinander (3D-Lage).

▷ **CovRelPose** (output_control) . . . . . real-array ⇝ *real*
6×6 Kovarianzmatrix der relativen Orientierung.

▷ **Error** (output_control) . . . . . real(-array) ⇝ *real*
Mittlerer, quadratischer Epipolar Abstand.

▷ **Points1** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 1.

▷ **Points2** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 2.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

points_foerstner, points_harris

Nachfolger

vector_to_rel_pose, gen_binocular_rectification_map

Siehe auch

binocular_calibration, match_fundamental_matrix_ransac, match_essential_matrix_ransac, create_pose

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

3D Metrology

reconst3d_from_fundamental_matrix (: : Rows1, Cols1, Rows2, Cols2, CovRR1, CovRC1, CovCC1, CovRR2, CovRC2, CovCC2, FMatrix, CovFMat : X, Y, Z, W, CovXYZW)

Berechnung der projektiven 3D Rekonstruktion von Punkten für eine gegebene Fundamental-Matrix.

Ein Bildpaar wird schwach kalibriert genannt, wenn die Fundamental-Matrix bekannt ist. Sei eine solche Fundamental-Matrix FMatrix und eine Menge von korrespondierenden Punkten (Rows1,Cols1) und (Rows2,Cols2) gegeben, so bestimmt der Operator reconst3d_from_fundamental_matrix die Raumpunkte, welche auf die entsprechenden Bildpunkte abgebildet werden. Diese dreidimensionale Rekonstruktion ist rein projektiv, und die projektiven Koordinaten werden in dem Vektor (X,Y,Z,W) zurückgegeben. Diese Art der Rekonstruktion wird auch projektive Triangulierung genannt. Sind zusätzlich die Kovarianzen CovRR1, CovRC1, CovCC1 und CovRR2, CovRC2, CovCC2 der Bildpunkte bekannt, so werden auch die Kovarianzen CovXYZW der rekonstruierten Punkte berechnet. Ist n die Anzahl der Korrespondenzen, so werden die Kovarianzen in einem $9 \times n$ langen Tupel aneinandergehängt. Die Berechnung der Kovarianzen wird noch genauer, wenn weiterhin die Kovarianz der Fundamental-Matrix CovFMat als Eingabe zu Verfügung steht.

Der Operator reconst3d_from_fundamental_matrix wird typischerweise nach match_fundamental_matrix_ransac aufgerufen, wenn eine 3D Rekonstruktion gewünscht ist. Dies spart Rechenzeit gegenüber der Verwendung des Operators vector_to_fundamental_matrix.

reconst3d_from_fundamental_matrix ist das projektive Pendant zur euklidischen Rekonstruktion mittels des Operators intersect_lines_of_sight.

Parameter

▷ **Rows1** (input_control) ..... number(-array) ⇝ *real* / integer
Eingabepunkte in Bild 1 (Zeilenkoordinate).

▷ **Cols1** (input_control) ..... number(-array) ⇝ *real* / integer
Eingabepunkte in Bild 1 (Spaltenkoordinate).

▷ **Rows2** (input_control) ..... number(-array) ⇝ *real* / integer
Eingabepunkte in Bild 2 (Zeilenkoordinate).

▷ **Cols2** (input_control) ..... number(-array) ⇝ *real* / integer
Eingabepunkte in Bild 2 (Spaltenkoordinate).

▷ **CovRR1** (input_control) ..... number(-array) ⇝ *real* / integer
Varianz in Zeilenrichtung der Punkte in Bild 1.
Defaultwert: []

▷ **CovRC1** (input_control) ..... number(-array) ⇝ *real* / integer
Kovarianz der Punkte in Bild 1.
Defaultwert: []

▷ **CovCC1** (input_control) ..... number(-array) ⇝ *real* / integer
Varianz in Spaltenrichtung der Punkte in Bild 1.
Defaultwert: []

▷ **CovRR2** (input_control) ..... number(-array) ⇝ *real* / integer
Varianz in Zeilenrichtung der Punkte in Bild 2.
Defaultwert: []

▷ **CovRC2** (input_control) ..... number(-array) ⇝ *real* / integer
Kovarianz der Punkte in Bild 2.
Defaultwert: []

▷ **CovCC2** (input_control) ..... number(-array) ⇝ *real* / integer
Varianz in Spaltenrichtung der Punkte in Bild 2.
Defaultwert: []

▷ **FMatrix** (input_control) ..... hom_mat2d ⇝ *real*
Fundamental-Matrix.

▷ **CovFMat** (input_control) ..... real-array ⇝ *real*
9×9 Kovarianzmatrix der Fundamental-Matrix.
Defaultwert: []

▷ **X** (output_control) ..... real(-array) ⇝ *real*
X-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **Y** (output_control) ..... real(-array) ⇝ *real*
Y-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **Z** (output_control) ..... real(-array) ⇝ *real*
Z-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **W** (output_control) ..... real(-array) ⇝ *real*
W-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **CovXYZW** (output_control) ..... real(-array) ⇝ *real*
Kovarianzmatrizen der rekonstruierten Punkte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`match_fundamental_matrix_ransac`

Alternativen

`vector_to_fundamental_matrix`, `intersect_lines_of_sight`

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2000.

Modul

3D Metrology

rel_pose_to_fundamental_matrix (: : RelPose, CovRelPose, CamPar1, CamPar2 : FMatrix, CovFMat)

Berechnung der Fundamental-Matrix aus der relativen Orientierung zweier Kameras.

Kameras mit Linsenverzeichnungen werden durch den folgenden Parametersatz beschrieben: Der Brennweite f, den beiden Skalierungsfaktoren s_x, s_y, den Koordinaten des Bildhauptpunktes (c_x, c_y) und dem Verzeichnungskoeffizienten κ. Für eine detailliertere Beschreibung siehe das Kapitel Kalibrierung. Nur Kameras mit einem Verzeichnungskoeffizienten gleich null bilden gerade Linien in Raum auf gerade Linien im Bild ab. Dies trifft auch auf Kameras mit telezentrischen Objektiven und mit Schwenkobjektiven zu. `rel_pose_to_fundamental_matrix` behandelt telezentrische Objektive und Schwenkobjektive korrekt.

Diese Objektivtypen werden jedoch zur Vereinfachung in den untenstehenden Formeln ignoriert. Ist der Verzeichnungskoeffizient gleich null, dann ist die Projektion eine lineare Abbildung, und die internen Kameraparameter können in einer Kameramatrix $CamMat$ zusammengefasst werden:

$$CamMat \quad = \quad \begin{pmatrix} f/s_x & 0 & c_x \\ 0 & f/s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad .$$

Ein lineares Kameramodell ist eine Approximation der realen, nichtlinearen Kamera. Für ein Vielzahl von Kameraobjektiven, besonders Teleobjektiven, ist der Fehler, welcher durch die lineare Approximation entsteht, vernachlässigbar. Gemäß der Formel $E = ([t]_\times R)^T$ wird die Essential-Matrix aus der Translation t und Rotation R der relativen Orientierung `RelPose` abgeleitet (siehe auch Operator `vector_to_rel_pose`). In Rahmen der linearen Approximation ist dann die Fundamental-Matrix gemäß der Formel präsentiert unter `essential_to_fundamental_matrix` aus der relativen Lage zu berechnen:

$$\texttt{FMatrix} \quad = \quad CamMat2^{-T} \cdot ([t]_\times R)^T \cdot CamMat1^{-1} \quad .$$

Die Transformation geht einher mit einer Fortpflanzung der Kovarianzmatrizen von `CovRelPose` zu `CovFMat`. Sollte `CovRelPose` leer sein, so wird `CovFMat` ebenfalls leer sein.

Der Konvertierungsoperator `rel_pose_to_fundamental_matrix` wird besonders zur Visualisierung der Epipolarlinienstruktur mittels der Fundamental-Matrix verwendet, was die zu Grunde liegende Stereogeometrie verdeutlicht.

Parameter

- ▷ **`RelPose`** (input_control) . . . . . pose ⇝ *real* / integer
  Relative Orientierung der Kameras (3D-Lage).
- ▷ **`CovRelPose`** (input_control) . . . . . number-array ⇝ *real* / integer
  6×6 Kovarianzmatrix der relativen Orientierung.
  Defaultwert: []
- ▷ **`CamPar1`** (input_control) . . . . . campar ⇝ *real* / integer / string
  Parameter der 1. Kamera.
- ▷ **`CamPar2`** (input_control) . . . . . campar ⇝ *real* / integer / string
  Parameter der 2. Kamera.
- ▷ **`FMatrix`** (output_control) . . . . . hom_mat2d ⇝ *real*
  Berechnete Fundamental-Matrix.
- ▷ **`CovFMat`** (output_control) . . . . . real-array ⇝ *real*
  9×9 Kovarianzmatrix der Fundamental-Matrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`vector_to_rel_pose`

Alternativen

`essential_to_fundamental_matrix`

Siehe auch

`calibrate_cameras`

Modul

3D Metrology

vector_to_essential_matrix (: : Rows1, Cols1, Rows2, Cols2, CovRR1, CovRC1, CovCC1, CovRR2, CovRC2, CovCC2, CamMat1, CamMat2, Method : EMatrix, CovEMat, Error, X, Y, Z, CovXYZ)

Bestimmung der Essential-Matrix unter Verwendung vorgegebener Punktkorrespondenzen und bekannter Kameramatrizen sowie Rekonstruktion der 3D Raumpunkte.

vector_to_essential_matrix ermittelt aus im allgemeinen mindestens sechs vorgegebenen Punktkorrespondenzen (Rows1,Cols1), (Rows2,Cols2) diejenige Essential-Matrix EMatrix, welche die Epipolargleichung

$$\begin{pmatrix} X_2 \\ Y_2 \\ 1 \end{pmatrix}^T \cdot EMatrix \cdot \begin{pmatrix} X_1 \\ Y_1 \\ 1 \end{pmatrix} = 0$$

am besten erfüllt. Der Operator vector_to_essential_matrix ist für ein lineares Kameramodell konzipiert. Die internen Kameraparameter werden durch die Argumente CamMat1, CamMat2 übergeben, welche 3×3 obere Dreiecksmatrizen sind und eine affine Transformation beschreiben. Die Beziehung zwischen einem 3D Richtungsvektor (X,Y,1) und dessen (projektiven) 2D Bildkoordinaten (col,row,1) ist:

$$\begin{pmatrix} col \\ row \\ 1 \end{pmatrix} = CamMat \cdot \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \quad \text{mit} \quad CamMat = \begin{pmatrix} f/s_x & s & c_x \\ 0 & f/s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} .$$

Zu beachten ist hier die Reihenfolge (Spalte/Zeile), welche mit der (x,y) Notation im Kamerakoordinatensystem konform gehen muss. Die Brennweite ist mit f bezeichnet, s_x, s_y sind Skalierungsfaktoren, s beschreibt eine Scherung und (c_x, c_y) gibt den Bildhauptpunkt an. Hauptsächlich sind dies die Elemente aus den Kameraparametern, wie sie z.B. in calibrate_cameras verwendet werden. Alternativ kann eine Kameramatrix auch durch andere Elemente beschrieben werden, siehe z.B. stationary_camera_self_calibration. Multipliziert man die (projektiven) Bildkoordinaten mit der invertierten Kameramatrix, so erhält man die 3D Richtungsvektoren für die Epipolargleichung.

Der Parameter Method gibt an, ob die Kameras sich in einer besonderen relativen Orientierung zueinander befinden, und bestimmt auch das Berechnungsverfahren. Für *'normalized_dlt'* und *'gold_standard'* kann die relative Lage der Kameras zueinander beliebig sein. Für *'trans_normalized_dlt'* und *'trans_gold_standard'* ist die relative Lage eine reine Translation. In diesem speziellen Fall ist die minimale Anzahl an notwendigen Punktkorrespondenzen nicht sechs, sondern nur zwei.

Wird *'normalized_dlt'* oder *'trans_normalized_dlt'* gewählt, so ist das Berechnungsverfahren ein lineares Verfahren. Wird *'gold_standard'* oder *'trans_gold_standard'* gewählt, so ist das Berechnungsverfahren im statistischen Sinne optimal. Beide Verfahren liefern die Koordinaten (X,Y,Z) der rekonstruierten Raumpunkte. Die optimalen Verfahren berechnen zusätzlich die 3×3 Kovarianzmatrizen CovXYZ der Raumpunkte. Ist n die Anzahl der Korrespondenzen, so werden diese Kovarianzen in einem $9 \times n$ Tupel aneinandergehängt. Weiterhin geben die optimalen Verfahren die Kovarianz der Essential-Matrix CovEMat an, welche eine 9×9 Matrix ist.

Falls die Bildpunkte mit einem Operator wie points_foerstner, der die Kovarianzmatrix für jeden Punkt zurückliefert, extrahiert wurden, kann dies in der Berechnung berücksichtigt werden, indem die Kovarianzen in CovRR1, CovRC1, CovCC1 für die Punkte des ersten Bildes und in CovRR2, CovRC2, CovCC2 für die Punkte des zweiten Bildes übergeben werden. Die Kovarianzmatrizen sind symmetrische 2×2 Matrizen. CovRR1/CovRR2 und CovCC1/CovCC2 sind dabei die Diagonalelemente der Matrizen, während CovRC1/CovRC2 die beiden nicht diagonalen Elemente angeben. Sind die Kovarianzen unbekannt, so werden zur Berechnung intern Einheits-Kovarianzmatrizen angenommen, und in den Kovarianzparametern können leere Tupel übergeben werden.

Die Größe Error ist ein Gütemaß für die Berechnung der Essential-Matrix und gibt den mittleren euklidischen Abstand in Pixeln zwischen den Punkten und ihren korrespondierenden Epipolarlinien an.

Bei dem Operator vector_to_essential_matrix ist folgender Spezialfall zu beachten: Liegen alle abgebildeten Raumpunkte in einer einzigen Ebene und liegen zusätzlich alle Raumpunkte näher zu einer der beiden Kameras, so gibt es insgesamt zwei Lösungen. Das heißt, dass in diesem Fall das Problem der Berechnung der Essential-Matrix nicht eindeutig lösbar ist. Es werden daher auch beide Lösungen ausgegeben. Das bedeutet, dass alle Ausgabeparameter von doppelter Länge sind, wobei die Werte der zweiten Lösung an die Werte der ersten Lösung hinten angehängt sind.

Sind die Korrespondenzen zwischen den Punkten noch nicht bekannt, so ist match_essential_matrix_ransac zur Bestimmung der Korrespondenzen sowie der Stereo-Geometrie zu verwenden.

Parameter

- ▷ **Rows1** (input_control) . . . number-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Zeilenkoordinate).
  Restriktion: length(Rows1) >= 6 || length(Rows1) >= 2
- ▷ **Cols1** (input_control) . . . number-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Spaltenkoordinate).
  Restriktion: length(Cols1) == length(Rows1)
- ▷ **Rows2** (input_control) . . . number-array ⇝ *real* / integer
  Eingabepunkte in Bild 2 (Zeilenkoordinate).
  Restriktion: length(Rows2) == length(Rows1)
- ▷ **Cols2** (input_control) . . . number-array ⇝ *real* / integer
  Eingabepunkte in Bild 2 (Spaltenkoordinate).
  Restriktion: length(Cols2) == length(Rows1)
- ▷ **CovRR1** (input_control) . . . number-array ⇝ *real* / integer
  Varianz in Zeilenrichtung der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovRC1** (input_control) . . . number-array ⇝ *real* / integer
  Kovarianz der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovCC1** (input_control) . . . number-array ⇝ *real* / integer
  Varianz in Spaltenrichtung der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovRR2** (input_control) . . . number-array ⇝ *real* / integer
  Varianz in Zeilenrichtung der Punkte in Bild 2.
  Defaultwert: []
- ▷ **CovRC2** (input_control) . . . number-array ⇝ *real* / integer
  Kovarianz der Punkte in Bild 2.
  Defaultwert: []
- ▷ **CovCC2** (input_control) . . . number-array ⇝ *real* / integer
  Varianz in Spaltenrichtung der Punkte in Bild 2.
  Defaultwert: []
- ▷ **CamMat1** (input_control) . . . hom_mat2d ⇝ *real* / integer
  Kameramatrix der 1. Kamera.
- ▷ **CamMat2** (input_control) . . . hom_mat2d ⇝ *real* / integer
  Kameramatrix der 2. Kamera.
- ▷ **Method** (input_control) . . . string ⇝ *string*
  Algorithmus zur Berechnung der Essential-Matrix und zur Auswahl spezieller relativer Orientierungen.
  Defaultwert: 'normalized_dlt'
  Werteliste: Method ∈ {'normalized_dlt', 'gold_standard', 'trans_normalized_dlt', 'trans_gold_standard'}
- ▷ **EMatrix** (output_control) . . . hom_mat2d ⇝ *real*
  Berechnete Essential-Matrix.
- ▷ **CovEMat** (output_control) . . . real-array ⇝ *real*
  9×9 Kovarianzmatrix der Essential-Matrix.
- ▷ **Error** (output_control) . . . real(-array) ⇝ *real*
  Mittlerer, quadratischer Epipolar Abstand.
- ▷ **X** (output_control) . . . real-array ⇝ *real*
  X-Koordinaten der rekonstruierten Punkte.
- ▷ **Y** (output_control) . . . real-array ⇝ *real*
  Y-Koordinaten der rekonstruierten Punkte.
- ▷ **Z** (output_control) . . . real-array ⇝ *real*
  Z-Koordinaten der rekonstruierten Punkte.
- ▷ **CovXYZ** (output_control) . . . real-array ⇝ *real*
  Kovarianzmatrizen der rekonstruierten 3D Punkte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

match_essential_matrix_ransac

Nachfolger

essential_to_fundamental_matrix

Alternativen

vector_to_rel_pose, vector_to_fundamental_matrix

Siehe auch

stationary_camera_self_calibration

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

J.Chris McGlone (editor): „Manual of Photogrammetry“; American Society for Photogrammetry and Remote Sensing ; 2004.

Modul

3D Metrology

vector_to_fundamental_matrix (: : Rows1, Cols1, Rows2, Cols2, CovRR1, CovRC1, CovCC1, CovRR2, CovRC2, CovCC2, Method : FMatrix, CovFMat, Error, X, Y, Z, W, CovXYZW)

Bestimmung der Fundamental-Matrix unter Verwendung vorgegebener Punktkorrespondenzen sowie Rekonstruktion der 3D Raumpunkte.

vector_to_fundamental_matrix ermittelt aus im allgemeinen mindestens acht vorgegebenen Punktkorrespondenzen (Rows1,Cols1), (Rows2,Cols2) diejenige Fundamental-Matrix FMatrix, welche die Epipolargleichung

$$\begin{pmatrix} \mathtt{Cols2} \\ \mathtt{Rows2} \\ 1 \end{pmatrix}^T \cdot \mathtt{FMatrix} \cdot \begin{pmatrix} \mathtt{Cols1} \\ \mathtt{Rows1} \\ 1 \end{pmatrix} = 0 \quad .$$

am besten erfüllt. Zu beachten ist hier die Position von Spalte bzw. Zeile in den Punktkoordinaten. Da die Fundamental-Matrix die projektive Beziehung zweier Stereobilder im 3D Raum wiedergibt, muss die X/Y Notation mit dem Kamera-Koordinaten-System konform sein. (X,Y) Koordinaten entsprechen daher einem (Column,Row) Paar.

Für eine beliebige relative Lage der Kameras zueinander ist die minimale Anzahl an Punktkorrespondenzen acht. Method ist dann gleich *'normalized_dlt'* oder *'gold_standard'* zu wählen. Sind linke und rechte Kamera identisch und ist zusätzlich die relative Lage zwischen den Kameras eine reine Translation, so ist Method gleich *'trans_normalized_dlt'* oder *'trans_gold_standard'* zu wählen. In diesem speziellen Fall ist die minimale Anzahl an korrespondierenden Punkte nur zwei.

Die Fundamental-Matrix wird so berechnet, dass ein Fehlermaß minimiert wird. Um dieses Ergebnis zu erreichen, sind verschiedene Algorithmen verfügbar. Welches Verfahren verwendet werden soll, kann ebenfalls mittels des Parameters Method gesteuert werden. Für Method gleich *'normalized_dlt'* bzw. *'trans_normalized_dlt'* wird ein lineares Verfahren verwendet, das einen algebraischen Fehler basierend auf obigen Epipolargleichungen minimiert. Dieses Verfahren bietet im allgemeinen einen guten Kompromiss zwischen Geschwindigkeit und Genauigkeit. Für Method gleich *'gold_standard'* bzw. *'trans_gold_standard'* wird eine mathematisch optimale, dafür aber langsamere, Optimierung durchgeführt, die den geometrischen Projektionsfehler von rekonstruierten projektiven 3D Punkten minimiert. In diesem Fall werden neben der Kovarianz-Matrix CovFMat der Fundamental-Matrix

auch die projektiven Koordinaten (X,Y,Z,W) der rekonstruierten Punkte inklusive ihrer Kovarianzen CovXYZW zurückgegeben. Sei n die Anzahl der Punkte, dann werden die aneinandergehängten Kovarianzen in einem $16 \times n$ Tuple übergeben.

Die Größe Error ist ein Gütemaß für die Schätzung der Fundamental- Matrix und gibt den mittleren euklidischen Abstand in Pixeln zwischen den Punkten und ihren korrespondierenden Epipolarlinien an.

Falls die Methode *'gold_standard'* bzw. *'trans_gold_standard'* verwendet wird und die Punkte mit einem Operator wie points_foerstner, der die Kovarianzmatrix für jeden Punkt zurückliefert, extrahiert wurden, kann dies in der Berechnung berücksichtigt werden, indem die Kovarianzen in CovRR1, CovRC1, CovCC1 für die Punkte des ersten Bildes und in CovRR2, CovRC2, CovCC2 für die Punkte des zweiten Bildes übergeben werden. Die Kovarianzmatrizen sind symmetrische 2×2 Matrizen. CovRR1/CovRR2 und CovCC1/CovCC2 sind dabei die Diagonalelemente der Matrizen, während CovRC1/CovRC2 die beiden nicht diagonalen Elemente angeben. Sind die Kovarianzen unbekannt und wird die *'gold_standard'* bzw. *'trans_gold_standard'* Methode verwendet, so werden zur Berechnung intern Einheits-Kovarianzmatrixen angenommen. Falls *'normalized_dlt'* bzw. *'trans_normalized_dlt'* verwendet wird oder die Kovarianzen unbekannt sind, können in den Kovarianzparametern leere Tupel übergeben werden.

Sind die Korrespondenzen zwischen den Punkten noch nicht bekannt, so ist match_fundamental_matrix_ransac zur Bestimmung der Korrespondenzen sowie der Stereo-Geometrie zu verwenden.

Parameter

▷ **Rows1** (input_control) . . . . . number-array ⇝ *real* / integer
Eingabepunkte in Bild 1 (Zeilenkoordinate).
Restriktion: length(Rows1) >= 8 || length(Rows1) >= 2

▷ **Cols1** (input_control) . . . . . number-array ⇝ *real* / integer
Eingabepunkte in Bild 1 (Spaltenkoordinate).
Restriktion: length(Cols1) == length(Rows1)

▷ **Rows2** (input_control) . . . . . number-array ⇝ *real* / integer
Eingabepunkte in Bild 2 (Zeilenkoordinate).
Restriktion: length(Rows2) == length(Rows1)

▷ **Cols2** (input_control) . . . . . number-array ⇝ *real* / integer
Eingabepunkte in Bild 2 (Spaltenkoordinate).
Restriktion: length(Cols2) == length(Rows1)

▷ **CovRR1** (input_control) . . . . . number-array ⇝ *real* / integer
Varianz in Zeilenrichtung der Punkte in Bild 1.
Defaultwert: []

▷ **CovRC1** (input_control) . . . . . number-array ⇝ *real* / integer
Kovarianz der Punkte in Bild 1.
Defaultwert: []

▷ **CovCC1** (input_control) . . . . . number-array ⇝ *real* / integer
Varianz in Spaltenrichtung der Punkte in Bild 1.
Defaultwert: []

▷ **CovRR2** (input_control) . . . . . number-array ⇝ *real* / integer
Varianz in Zeilenrichtung der Punkte in Bild 2.
Defaultwert: []

▷ **CovRC2** (input_control) . . . . . number-array ⇝ *real* / integer
Kovarianz der Punkte in Bild 2.
Defaultwert: []

▷ **CovCC2** (input_control) . . . . . number-array ⇝ *real* / integer
Varianz in Spaltenrichtung der Punkte in Bild 2.
Defaultwert: []

▷ **Method** (input_control) . . . . . string ⇝ *string*
Schätzalgorithmus.
Defaultwert: 'normalized_dlt'
Werteliste: Method ∈ {'normalized_dlt', 'gold_standard', 'trans_normalized_dlt', 'trans_gold_standard'}

▷ **FMatrix** (output_control) . . . . . hom_mat2d ⇝ *real*
Berechnete Fundamental-Matrix.

▷ **CovFMat** (output_control) . . . . . real-array ⇝ *real*
9×9 Kovarianzmatrix der Fundamental-Matrix.

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlerer, quadratischer Epipolar Abstand.

▷ **X** (output_control) . . . . . real-array ⇝ *real*
X-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **Y** (output_control) . . . . . real-array ⇝ *real*
Y-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **Z** (output_control) . . . . . real-array ⇝ *real*
Z-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **W** (output_control) . . . . . real-array ⇝ *real*
W-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **CovXYZW** (output_control) . . . . . real-array ⇝ *real*
Kovarianzmatrizen der rekonstruierten 3D Punkte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

match_fundamental_matrix_ransac

Nachfolger

gen_binocular_proj_rectification

Alternativen

vector_to_essential_matrix, vector_to_rel_pose

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2000.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

3D Metrology

vector_to_fundamental_matrix_distortion (: : Rows1, Cols1, Rows2, Cols2, CovRR1, CovRC1, CovCC1, CovRR2, CovRC2, CovCC2, ImageWidth, ImageHeight, Method : FMatrix, Kappa, Error, X, Y, Z, W)

Bestimmung der Fundamental-Matrix und des radialen Verzeichnungskoeffizienten unter Verwendung vorgegebener Punktkorrespondenzen sowie Rekonstruktion der 3D Raumpunkte.

vector_to_fundamental_matrix_distortion ermittelt aus vorgegebenen Punktkorrespondenzen (Rows1,Cols1), (Rows2,Cols2) die Fundamental-Matrix FMatrix und den radialen Verzeichnungskoeffizienten Kappa (κ), welche die Epipolargleichung

$$\begin{pmatrix} c_2 \\ r_2 \\ 1 \end{pmatrix}^T \cdot \texttt{FMatrix} \cdot \begin{pmatrix} c_1 \\ r_1 \\ 1 \end{pmatrix} = 0 \ .$$

am besten erfüllen. Hierbei sind (r_1, c_1) und (r_2, c_2) Bildpunkte, die durch Entzerrung der Eingabebildpunkte mit dem Divisionsmodell entstehen (siehe Kalibrierung):

$$r = \frac{\tilde{r}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)} \qquad c = \frac{\tilde{c}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)}$$

Hierbei bezeichnen $(\tilde{r}_1, \tilde{c}_1) = (\texttt{Rows1} - 0.5(\texttt{ImageHeight} - 1), \texttt{Cols1} - 0.5(\texttt{ImageWidth} - 1))$ und $(\tilde{r}_2, \tilde{c}_2) = (\texttt{Rows2} - 0.5(\texttt{ImageHeight} - 1), \texttt{Cols2} - 0.5(\texttt{ImageWidth} - 1))$ die verzerrten Bildpunkte relativ zum Bildmittelpunkt. `vector_to_fundamental_matrix_distortion` nimmt also an, dass der Hauptpunkt, d.h. das Zentrum der radialen Verzeichnungen, im Bildmittelpunkt liegt.

Das zurückgelieferte `Kappa` kann dazu verwendet werden, Kameraparameter zu konstruieren, die zur Entzerrung von Bildern oder Bildpunkten verwendet werden können (siehe `change_radial_distortion_cam_par`, `change_radial_distortion_image` und `change_radial_distortion_points`):

$$\begin{aligned}\texttt{CamPar} \quad = \quad & [0.0, \texttt{Kappa}, 1.0, 1.0, 0.5(\texttt{ImageWidth} - 1), 0.5(\texttt{ImageHeight} - 1), \\ & \texttt{ImageWidth}, \texttt{ImageHeight}]\end{aligned}$$

Zu beachten bei den obigen Formeln ist die Position von Spalte bzw. Zeile in den Punktkoordinaten. Da die Fundamental-Matrix die projektive Beziehung zweier Stereobilder im 3D Raum wiedergibt, muss die X/Y Notation mit dem Kamerakoordinatensystem konform sein. (X,Y) Koordinaten entsprechen daher einem (Column,Row) Paar.

Für eine beliebige relative Lage der Kameras zueinander ist die minimale Anzahl an Punktkorrespondenzen neun. `Method` ist dann gleich *'linear'* oder *'gold_standard'* zu wählen. Sind die linke und rechte Kamera identisch und ist zusätzlich die relative Lage zwischen den Kameras eine reine Translation, so ist `Method` gleich *'trans_linear'* oder *'trans_gold_standard'* zu wählen. In diesem speziellen Fall ist die minimale Anzahl an korrespondierenden Punkten nur vier.

Die Fundamental-Matrix wird so berechnet, dass ein Fehlermaß minimiert wird. Um dieses Ergebnis zu erreichen, sind verschiedene Algorithmen verfügbar. Welches Verfahren verwendet werden soll, kann ebenfalls mittels des Parameters `Method` gesteuert werden. Für `Method` gleich *'linear'* bzw. *'trans_linear'* wird ein lineares Verfahren verwendet, das einen algebraischen Fehler basierend auf obigen Epipolargleichungen minimiert. Dieses Verfahren ist sehr schnell. Für den Fall der reinen Translation (`Method` = *'trans_linear'*) liefert das lineare Verfahren genaue Ergebnisse für geringes bis mittleres Rauschen der Punktkoordinaten und für die meisten Verzeichnungen (außer sehr kleinen Verzeichnungen). Für eine beliebige Lage der Kameras (`Method` = *'linear'*) liefert das lineare Verfahren nur für sehr geringes Rauschen der Punktkoordinaten und hinreichend große Verzeichnungen genaue Ergebnisse. Für `Method` gleich *'gold_standard'* bzw. *'trans_gold_standard'* wird eine mathematisch optimale, dafür aber langsamere, Optimierung durchgeführt, die den geometrischen Reprojektionsfehler von rekonstruierten projektiven 3D Punkten minimiert. In diesem Fall werden neben der Fundamental-Matrix und dem radialen Verzeichnungskoeffizienten auch die projektiven Koordinaten (`X`,`Y`,`Z`,`W`) der rekonstruierten Punkte zurückgegeben. Für beliebige Lage der Kameras sollte im allgemeinen `Method` = *'gold_standard'* gewählt werden.

Falls die Methode *'gold_standard'* bzw. *'trans_gold_standard'* verwendet wird und die Punkte mit einem Operator wie `points_foerstner`, der die Kovarianzmatrix für jeden Punkt zurückliefert, extrahiert wurden, kann dies in der Berechnung berücksichtigt werden, indem die Kovarianzen in `CovRR1`, `CovRC1`, `CovCC1` für die Punkte des ersten Bildes und in `CovRR2`, `CovRC2`, `CovCC2` für die Punkte des zweiten Bildes übergeben werden. Sind die Kovarianzen unbekannt und wird die Methode *'gold_standard'* bzw. *'trans_gold_standard'* verwendet, so werden zur Berechnung intern Einheits-Kovarianzmatrizen angenommen. Falls *'linear'* bzw. *'trans_linear'* verwendet wird oder die Kovarianzen unbekannt sind, können in den Kovarianzparametern leere Tupel übergeben werden.

Die Größe `Error` ist ein Gütemaß für die Schätzung der Fundamental-Matrix und gibt den mittleren symmetrischen euklidischen Abstand in Pixeln zwischen den Punkten und ihren korrespondierenden Epipolarlinien an.

Sind die Korrespondenzen zwischen den Punkten noch nicht bekannt, so ist `match_fundamental_matrix_distortion_ransac` zur Bestimmung der Korrespondenzen sowie der Stereo-Geometrie zu verwenden.

Parameter

▷ **`Rows1`** (input_control) . . . . . point.y-array ⇝ *real* / integer
Eingabepunkte in Bild 1 (Zeilenkoordinate).
Restriktion: length(Rows1) >= 9 || length(Rows1) >= 4

▷ **`Cols1`** (input_control) . . . . . point.x-array ⇝ *real* / integer
Eingabepunkte in Bild 1 (Spaltenkoordinate).
Restriktion: length(Cols1) == length(Rows1)

▷ **`Rows2`** (input_control) . . . . . point.y-array ⇝ *real* / integer
Eingabepunkte in Bild 2 (Zeilenkoordinate).
Restriktion: length(Rows2) == length(Rows1)

▷ **Cols2** (input_control) . . . . . point.x-array ⇝ *real* / integer
Eingabepunkte in Bild 2 (Spaltenkoordinate).
Restriktion: length(Cols2) == length(Rows1)

▷ **CovRR1** (input_control) . . . . . number-array ⇝ *real* / integer
Varianz in Zeilenrichtung der Punkte in Bild 1.
Defaultwert: []

▷ **CovRC1** (input_control) . . . . . number-array ⇝ *real* / integer
Kovarianz der Punkte in Bild 1.
Defaultwert: []

▷ **CovCC1** (input_control) . . . . . number-array ⇝ *real* / integer
Varianz in Spaltenrichtung der Punkte in Bild 1.
Defaultwert: []

▷ **CovRR2** (input_control) . . . . . number-array ⇝ *real* / integer
Varianz in Zeilenrichtung der Punkte in Bild 2.
Defaultwert: []

▷ **CovRC2** (input_control) . . . . . number-array ⇝ *real* / integer
Kovarianz der Punkte in Bild 2.
Defaultwert: []

▷ **CovCC2** (input_control) . . . . . number-array ⇝ *real* / integer
Varianz in Spaltenrichtung der Punkte in Bild 2.
Defaultwert: []

▷ **ImageWidth** (input_control) . . . . . integer ⇝ *integer*
Breite der Bilder, aus denen die Punkte extrahiert wurden.
Restriktion: ImageWidth > 0

▷ **ImageHeight** (input_control) . . . . . integer ⇝ *integer*
Höhe der Bilder, aus denen die Punkte extrahiert wurden.
Restriktion: ImageHeight > 0

▷ **Method** (input_control) . . . . . string ⇝ *string*
Schätzalgorithmus.
Defaultwert: 'gold_standard'
Werteliste: `Method` ∈ {'linear', 'gold_standard', 'trans_linear', 'trans_gold_standard'}

▷ **FMatrix** (output_control) . . . . . hom_mat2d ⇝ *real*
Berechnete Fundamental-Matrix.

▷ **Kappa** (output_control) . . . . . real ⇝ *real*
Berechneter Verzeichnungskoeffizient.

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlerer quadratischer Epipolarabstand.

▷ **X** (output_control) . . . . . real-array ⇝ *real*
X-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **Y** (output_control) . . . . . real-array ⇝ *real*
Y-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **Z** (output_control) . . . . . real-array ⇝ *real*
Z-Koordinaten der rekonstruierten Punkte im projektiven Raum.

▷ **W** (output_control) . . . . . real-array ⇝ *real*
W-Koordinaten der rekonstruierten Punkte im projektiven Raum.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`match_fundamental_matrix_distortion_ransac`

Nachfolger

`change_radial_distortion_cam_par`, `change_radial_distortion_image`, `change_radial_distortion_points`, `gen_binocular_proj_rectification`

Alternativen

vector_to_fundamental_matrix, vector_to_essential_matrix, vector_to_rel_pose

Siehe auch

calibrate_cameras

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

3D Metrology

vector_to_rel_pose (: : Rows1, Cols1, Rows2, Cols2, CovRR1, CovRC1, CovCC1, CovRR2, CovRC2, CovCC2, CamPar1, CamPar2, Method : RelPose, CovRelPose, Error, X, Y, Z, CovXYZ)

Bestimmung der relativen Orientierung zweier Kameras unter Verwendung vorgegebener Punktkorrespondenzen und bekannter Kameraparameter sowie Rekonstruktion der 3D Raumpunkte.

vector_to_rel_pose ermittelt aus im allgemeinen mindestens sechs vorgegebenen Punktkorrespondenzen eines Stereobildpaares die relative Orientierung der beiden Kameras zueinander. RelPose gibt die Orientierung der Kamera 1 relative zur Kamera 2 an (Siehe create_pose für weitere Information über die Beschreibung von Orientierungen.). Dies steht in Übereinstimmung mit der expliziten Kalibrierung einer Stereokonfiguration mit dem Operator calibrate_cameras. Sei R, t die Rotation beziehungsweise Translation der relativen Orientierung. Die Essential-Matrix E ist dann durch $E = ([t]_\times R)^T$ definiert, wobei $[t]_\times$ eine 3×3 schiefsymmetrische Matrix ist, welche das Kreuzprodukt mit dem Vektor t beschreibt. Die gesuchte Orientierung kann mittels der Epipolar-Gleichungen bestimmt werden:

$$\begin{pmatrix} X_2 \\ Y_2 \\ 1 \end{pmatrix}^T \cdot ([t]_\times R)^T \cdot \begin{pmatrix} X_1 \\ Y_1 \\ 1 \end{pmatrix} = 0 \quad \text{mit} \quad [t]_\times = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix} .$$

Zu beachten ist, dass die Essential-Matrix eine homogene Größe ist, d.h. sie ist bis auf einen Skalierungsfaktor bestimmt. Somit kann der Translationsvektor der relativen Orientierung auch nur bis auf eine Skalierung berechnet werden. Der Operator wird diesen Vektor auf die Länge eins normiert berechnen. Als Konsequenz ergibt sich, dass die Rekonstruktion der betrachteten Szene, hier in Form der Punkte gegeben durch deren 3D Koordinaten (X,Y,Z), auch nur bis auf einen globalen Skalierungsfaktor durchgeführt werden kann. Sollen die absoluten 3D Koordinaten der Szenenpunkte rekonstruiert werden, so ist es notwendig in beiden Bildern einen bekannten Maßstab zu sehen. In der Praxis kann dieser Maßstab z.B. durch den Abstand zweier beliebiger Punkte gegeben sein.

Der Operator vector_to_rel_pose ist für ein nichtlineares Kameramodell konzipiert und beschreibt somit auch radiale Linsenverzeichnungen. Dies steht im Gegensatz zum Operator vector_to_essential_matrix, welcher nur Geraden erhaltende, d.h. lineare, Kameras beschreibt. Kameraparameter werden mit den Argumenten CamPar1 und CamPar2 übergeben. Die 3D Richtungsvektoren $(X_1, Y_1, 1)$ und $(X_2, Y_2, 1)$ werden aus den Punktkoordinaten (Rows1,Cols1) und (Rows2,Cols2) durch Invertierung der Kameraprojektion berechnet (siehe Kalibrierung).

Der Parameter Method gibt an, ob die Kameras sich in einer besonderen relativen Orientierung zueinander befinden, und bestimmt auch das Berechnungsverfahren. Für *'normalized_dlt'* und *'gold_standard'* kann die relative Lage der Kameras zueinander beliebig sein. Für *'trans_normalized_dlt'* und *'trans_gold_standard'* ist die relative Lage eine reine Translation. In diesem speziellen Fall ist die minimale Anzahl an notwendigen Punktkorrespondenzen nicht sechs, sondern nur zwei.

Wird *'normalized_dlt'* oder *'trans_normalized_dlt'* gewählt, so ist das Berechnungsverfahren ein lineares Verfahren. Wird *'gold_standard'* oder *'trans_gold_standard'* gewählt, so ist das Berechnungsverfahren im statistischen Sinne optimal. Beide Verfahren liefern die Koordinaten (X,Y,Z) der rekonstruierten Raumpunkte. Die optimalen Verfahren berechnen zusätzlich die 3×3 Kovarianzmatrizen CovXYZ der Raumpunkte. Ist n die Anzahl der Korrespondenzen, so werden diese Kovarianzen in einem $9 \times n$ Tupel aneinandergehängt. Weiterhin geben die optimalen Verfahren auch die Kovarianz der relativen Lage CovRelPose an, welche eine 6×6 Matrix ist.

Falls die Bildpunkte mit einem Operator wie points_foerstner, der die Kovarianzmatrix für jeden Punkt zurückliefert, extrahiert wurden, kann dies in der Berechnung berücksichtigt werden, indem die Kovarianzen in CovRR1, CovRC1, CovCC1 für die Punkte des ersten Bildes und in CovRR2, CovRC2, CovCC2 für die Punkte des zweiten Bildes übergeben werden. Die Kovarianzmatrizen sind symmetrische 2×2 Matrizen. CovRR1/CovRR2 und CovCC1/CovCC2 sind dabei die Diagonalelemente der Matrizen, während CovRC1/CovRC2 die beiden nicht diagonalen Elemente angeben. Sind die Kovarianzen unbekannt, so werden zur Berechnung intern Einheits-Kovarianzmatrizen angenommen, und in den Kovarianzparametern können leere Tupel übergeben werden.

Die Größe Error ist ein Gütemaß für die Berechnung der relativen Lage und gibt den mittleren euklidischen Abstand in Pixeln zwischen den Punkten und ihren korrespondierenden Epipolarlinien an.

Bei dem Operator vector_to_rel_pose ist folgender Spezialfall zu beachten: Liegen alle abgebildeten Raumpunkte in einer einzigen Ebene und liegen zusätzlich alle Raumpunkte näher zu einer der beiden Kameras, so gibt es insgesamt zwei Lösungen. Das heißt, dass in diesem Fall das Problem der Berechnung der relativen Lage nicht eindeutig lösbar ist. Es werden daher auch beide Lösungen ausgegeben. Das bedeutet, dass alle Ausgabeparameter von doppelter Länge sind, wobei die Werte der zweiten Lösung an die Werte der ersten Lösung hinten angehängt sind.

Sind die Korrespondenzen zwischen den Punkten noch nicht bekannt, so ist match_rel_pose_ransac zur Bestimmung der Korrespondenzen sowie der Stereo-Geometrie zu verwenden.

Parameter

- ▷ **Rows1** (input_control) . . . number-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Zeilenkoordinate).
  Restriktion: length(Rows1) >= 6 || length(Rows1) >= 2
- ▷ **Cols1** (input_control) . . . number-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Spaltenkoordinate).
  Restriktion: length(Cols1) == length(Rows1)
- ▷ **Rows2** (input_control) . . . number-array ⇝ *real* / integer
  Eingabepunkte in Bild 2 (Zeilenkoordinate).
  Restriktion: length(Rows2) == length(Rows1)
- ▷ **Cols2** (input_control) . . . number-array ⇝ *real* / integer
  Eingabepunkte in Bild 2 (Spaltenkoordinate).
  Restriktion: length(Cols2) == length(Rows1)
- ▷ **CovRR1** (input_control) . . . number-array ⇝ *real* / integer
  Varianz in Zeilenrichtung der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovRC1** (input_control) . . . number-array ⇝ *real* / integer
  Kovarianz der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovCC1** (input_control) . . . number-array ⇝ *real* / integer
  Varianz in Spaltenrichtung der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovRR2** (input_control) . . . number-array ⇝ *real* / integer
  Varianz in Zeilenrichtung der Punkte in Bild 2.
  Defaultwert: []
- ▷ **CovRC2** (input_control) . . . number-array ⇝ *real* / integer
  Kovarianz der Punkte in Bild 2.
  Defaultwert: []
- ▷ **CovCC2** (input_control) . . . number-array ⇝ *real* / integer
  Varianz in Spaltenrichtung der Punkte in Bild 2.
  Defaultwert: []
- ▷ **CamPar1** (input_control) . . . campar ⇝ *real* / integer / string
  Kameraparameter der 1. Kamera.
- ▷ **CamPar2** (input_control) . . . campar ⇝ *real* / integer / string
  Kameraparameter der 2. Kamera.

▷ **Method** (input_control) . . . . . string ⇝ *string*
Algorithmus zur Berechnung der relativen Lage und zur Auswahl spezieller relativer Lagen.
Defaultwert: 'normalized_dlt'
Werteliste: Method ∈ {'normalized_dlt', 'gold_standard', 'trans_normalized_dlt', 'trans_gold_standard'}

▷ **RelPose** (output_control) . . . . . pose ⇝ *real* / integer
Berechnete relative Lage der Kameras zueinander (3D-Lage).

▷ **CovRelPose** (output_control) . . . . . real-array ⇝ *real*
6×6 Kovarianzmatrix der relativen Lage.

▷ **Error** (output_control) . . . . . real(-array) ⇝ *real*
Mittlerer, quadratischer Epipolar Abstand.

▷ **X** (output_control) . . . . . real-array ⇝ *real*
X-Koordinaten der rekonstruierten Punkte.

▷ **Y** (output_control) . . . . . real-array ⇝ *real*
Y-Koordinaten der rekonstruierten Punkte.

▷ **Z** (output_control) . . . . . real-array ⇝ *real*
Z-Koordinaten der rekonstruierten Punkte.

▷ **CovXYZ** (output_control) . . . . . real-array ⇝ *real*
Kovarianzmatrizen der rekonstruierten 3D Punkte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

match_rel_pose_ransac

Nachfolger

gen_binocular_rectification_map, rel_pose_to_fundamental_matrix

Alternativen

vector_to_essential_matrix, vector_to_fundamental_matrix, binocular_calibration

Siehe auch

camera_calibration

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

J.Chris McGlone (editor): „Manual of Photogrammetry“; American Society for Photogrammetry and Remote Sensing ; 2004.

Modul

3D Metrology

5.2 Depth From Focus

```
depth_from_focus ( MultiFocusImage : Depth, Confidence : Filter,
    Selection : )
```

Bestimmung der Tiefe aus mehreren Fokusebenen.

depth_from_focus bestimmt aus mehreren Bildern einer Fokusserie die Tiefe. Die Bilder der Fokusserie werden dabei als ein mehrkanaliges Bild in MultiFocusImage übergeben. Die Tiefe für jeden Bildpunkt wird als die Nummer des Kanals in Depth zurückgeben. Der Parameter Confidence gibt die Sicherheit an, mit der die Tiefe für die Bildpunkte geschätzt werden konnte. Große Werte bedeuten dabei eine große Zuverlässigkeit der Schätzung.

Es wird dasjenige Pixel einer Fokusebene ausgewählt, das lokal am schärfsten ist. Das verwendete Verfahren wird über die Parameter `Filter` und `Selection` festgelegt.

Dem Parameter `Filter` können die Werte *'highpass'* und *'bandpass'* übergeben werden. Um die Schärfe im Bild zu bestimmen, kann entweder ein Highpassfilter oder ein Bandpassfilter verwendet werden. Je größer der resultierende Wert ist, desto schärfer ist das Bild an dieser Stelle. Beim Bandpassfilter werden - im Vergleich zum Highpassfilter - die ganz hohen Frequenzen unterdrückt. Dies ist z.B. bei stark verrauschten Bildern nützlich.

Das gefilterte Bild kann optional noch mit dem Mittelwertfilter geglättet werden, indem man zwei zusätzliche Integer-Werte für die Maskengröße im Parameter `Filter` übergibt (z.B. [*'highpass'*, *7*, *7*]). Dadurch wird die Schärfeinformation über die benachbarten Pixel "verschmiert", so dass texturlose Bereiche im Bild überbrückt werde können. Bitte beachten Sie, dass diese Glättung kein Rauschen in den Originalbildern unterdrückt, da sie erst nach der High- oder Bandpassfilterung angewendet wird.

Der Parameter `Selection` bestimmt, wie die optimale Fokusebene ausgewählt wird. Wenn der Wert *'next_maximum'* übergeben wird, wird das nächstliegende Schärfemaximum in der Umgebung jedes Pixels verwendet. Bei *'local'* werden dagegen die für das Pixel ermittelten Schärfewerte aller Fokusebenen verwendet. Mit *'next_maximum'* erhält man i.d.R. ein etwas robusteres und leicht geglättetes Ergebnis.

Eine solche zusätzliche Glättung ist nützlich, wenn zur Aufnahme der Fokussequenz keine telezentrische Optik verwendet wird. In diesem Fall verändert sich die Position eines Punktes innerhalb der Sequenz leicht. Durch geeignete Wahl einer zusätzlichen Glättung kann dieser Effekt teilweise kompensiert werden.

Achtung

Besteht die Fokusserie `MultiFocusImage` aus mehr als 255 Fokusleveln (Kanälen) so wird die Tiefe für Werte über 255 abgeschnitten, d.h. Tiefenwerte größer als 255 werden ignoriert.

Werden für die Filtermaske von `Filter` gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Wird `Selection` auf *'local'* gesetzt und als `Filter` *'highpass'* oder *'bandpass'* gewählt, kann `depth_from_focus` auf OpenCL-Geräten ausgeführt werden. Wird die zusätzliche Glättung aktiviert, gelten die selben Einschränkungen und Voraussetzungen wie für `mean_image`.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`MultiFocusImage`** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte
  Mehrkanaliges Graubild bestehend aus meheren Fokusebenen.
- ▷ **`Depth`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Tiefenbild.
- ▷ **`Confidence`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Sicherheit der Tiefenschätzung.
- ▷ **`Filter`** (input_control) ..... string(-array) ⇝ *string* / integer
  Filter, um scharfe Pixel zu finden.
  Defaultwert: 'highpass'
  Wertevorschläge: `Filter` ∈ {'highpass', 'bandpass', 3, 5, 7, 9}
- ▷ **`Selection`** (input_control) ..... string(-array) ⇝ *string*
  Methode scharfe Pixel zu finden.
  Defaultwert: 'next_maximum'
  Werteliste: `Selection` ∈ {'next_maximum', 'local'}

Beispiel

```
compose3(Focus0,Focus1,Focus2,&MultiFocus);
depth_from_focus(MultiFocus,&Depth,&Confidence,'highpass','next_maximum');
mean_image(Depth,&Smooth,15,15);
select_grayvalues_from_channels(MultiChannel,Smooth,SharpImage);
threshold(Confidence,HighConfidence,10,255);
reduce_domain(SharpImage,HighConfidence,ConfidentSharp);
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

compose2, compose3, compose4, add_channels, read_image, read_sequence

Nachfolger

select_grayvalues_from_channels, mean_image, binomial_filter, gauss_filter, threshold

Siehe auch

count_channels

Modul

3D Metrology

select_grayvalues_from_channels (MultichannelImage, IndexImage : Selected : :)

Auswahl von Grauwerten eines mehrkanaligen Bildes mit Hilfe eines Indexbildes.

select_grayvalues_from_channels wählt aus einem mehrkanaligen Bild MultichannelImage Grauwerte aus verschiedenen Kanälen aus. Die Pixelwerte im Bild IndexImage geben dabei an, aus welchem Kanal der Grauwert selektiert werden soll. Es gilt zu beachten, dass IndexImage aus Gründen der der Rückwärtskompatibilität eine beliebige Anzahl Kanäle haben darf, es wird jedoch stets nur der erste Kanal berücksichtigt.

Parameter

- ▷ **MultichannelImage** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Mehrkanaliges Grauwertbild.
- ▷ **IndexImage** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Bild, bei dem die Pixelwerte als Index interpretiert werden.
  Parameteranzahl: IndexImage == MultichannelImage || IndexImage == 1
- ▷ **Selected** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Ergebnisbild.

Beispiel

```
compose3(Focus0,Focus1,Focus2,&MultiFocus);
depth_from_focus(MultiFocus,&Depth,&Confidence,'highpass','next_maximum');
mean_image(Depth,&Smooth,15,15);
select_grayvalues_from_channels(MultiChannel,Smooth,SharpImage);
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

depth_from_focus, mean_image

Nachfolger

disp_image

Siehe auch

count_channels

Modul

Foundation

5.3 Mehrbild-Stereo

Diese Kapitel enthält Operatoren für die Mehrbild-Stereo 3D-Rekonstruktion.

Konzept der Mehrbild-Stereo 3D-Rekonstruktion

Mit der Mehrbild-Stereo 3D-Rekonstruktion ist es möglich, aus 2D-Bildern von mehreren Kameras 3D-Objekte zu generieren. Es kann entweder die komplette 3D-Oberfläche eines Objekts rekonstruiert werden, oder es können auch nur einzelne 3D-Punkte rekonstruiert werden.

Im Folgenden werden kurz die notwendigen Schritte beschrieben, um 3D-Oberflächen oder 3D-Punkte zu rekonstruieren. Beachten Sie, dass ein gut kalibriertes Kamerasetup die Grundvoraussetzung für eine präzise 3D-Rekonstruktion ist; siehe dazu Kalibrierung für mehr Details. Außerdem wird in dem HDevelop Beispiel `reconstruct_surface_mixed_camera_types.hdev` der typische Ablauf einer Kalibrierung nachvollzogen (vom Kalibrierdatenmodell über das Kameraaufbaumodell hin zum Stereomodell).

Erstellen des Stereomodells: Zuerst muss das Stereomodell erstellt werden. Dies geschieht mit dem Operator

- create_stereo_model.

Falls 3D-Punkte rekonstruiert werden sollen, wählen Sie die Methode `'points_3d'`.



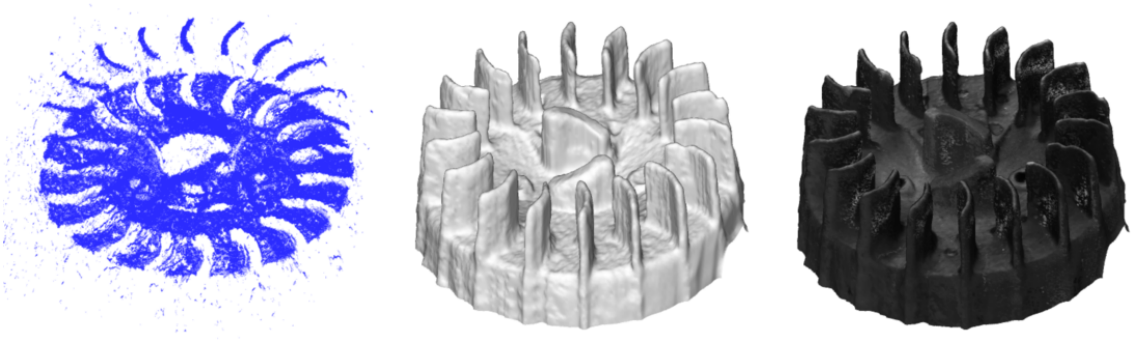
3D-Punktrekonstruktion mit `'points_3d'`.

Für die Rekonstruktion von Oberflächen stehen zwei Methoden zur Auswahl: `'surface_pairwise'` und `'surface_fusion'`. Detaillierte Informationen zu beiden Methoden können in der Referenz von reconstruct_surface_stereo nachgelesen werden.



(1) (2) (3)

Diese drei 3D-Bilder werden für die Oberflächenrekonstruktion verwendet, deren Ergebnisse in den Bildern unten zu sehen sind.



(1) (2) (3)

(1) Oberflächenrekonstruktion mit `'surface_pairwise'`. (2) Oberflächenrekonstruktion mit `'surface_fusion'`. (3) Oberflächenrekonstruktion mit `'surface_fusion'`, wobei die Farbinformationen aus den verwendeten 2D-Bildern extrahiert wurden. In dem HDevelop Beispiel `reconstruct_surface_mixed_camera_types.hdev` kann die hier gezeigte 3D-Rekonstruktion nachvollzogen werden.

Setzen der Bildpaare (nur für Oberflächenrekonstruktion): Bei der 3D-Rekonstruktion von Oberflächen werden mehrere binokulare Stereorekonstruktionen ausgeführt und dann kombiniert. Für diese binokulare Rekonstruktion müssen Bildpaare spezifiziert werden. Beispielsweise könnten die Bildpaare für die Bilder oben `[0,1]` und `[1,2]` sein. Die Bildpaare können mit

- `set_stereo_model_image_pairs`

spezifiziert, und mit

- `get_stereo_model_image_pairs`

abgefragt werden.

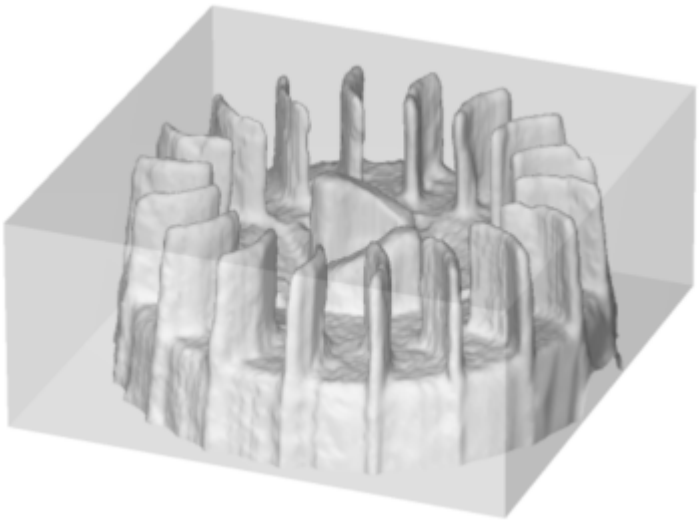
Mehr Informationen können in den Referenzen von `reconstruct_surface_stereo` und den oben genannten Operatoren nachgelesen werden.

Modifizieren der Stereomodell-Parameter: Mit

- `set_stereo_model_param`

können die Einstellungen der 3D-Rekonstruktion für das jeweils individuelle Setup optimiert werden.

Bei der Rekonstruktion von Oberflächen ist es sehr empfehlenswert, die 3D-Rekonstruktion mit einem umschließenden Quader (einer `'bounding box'`) einzuschränken. Diese sollte das zu rekonstruierende Objekt so eng wie möglich umschließen.



Der umschließende Quader, der mit set_stereo_model_param spezifiziert wurde, schränkt den Bereich ein, in welchem das Objekt rekonstruiert wird. So kann die Laufzeit stark reduziert werden.

Wenn die Methode 'surface_fusion' in create_stereo_model verwendet wird, sollten dennoch zuerst die Parameter der 'surface_pairwise'-Methode optimiert werden, da deren Ergebnis als Grundlage hergenommen wird. Die einzelnen Parameter werden in den Beispielen reconstruct_surface_stereo_pairwise_workflow.hdev und reconstruct_surface_stereo_fusion_workflow.hdev näher betrachtet.

Die Parameter können mit

- get_stereo_model_param

abgefragt werden.

Ausführen der 3D-Rekonstruktion: Um dann die eigentliche Rekonstruktion auszuführen, stehen

- reconstruct_points_stereo oder
- reconstruct_surface_stereo

zur Verfügung.

Erhalten von Zwischenergebnissen (nur für Oberflächenrekonstruktion): Es ist zu beachten, dass die Zwischenergebnisse nur abgerufen werden können, wenn der *'persistence'*-Modus mit set_stereo_model_param aktiviert wurde.

Mit

- get_stereo_model_object

können die Zwischenergebnisse der Oberflächenrekonstruktion von reconstruct_surface_stereo abgerufen und inspiziert werden. Diese Bilder können benutzt werden, wenn ein Problem in der Rekonstruktion zurückverfolgt werden muss.

Mit

- get_stereo_model_object_model_3d

kann das 3D-Objektmodell abgerufen werden, dass als Zwischenergebnis mit reconstruct_surface_stereo unter Verwendung der Methode *'surface_fusion'* abgespeichert wurde.

clear_stereo_model (: : StereoModelID :)

Freigeben des vom Stereomodell angelegten Speichers.

clear_stereo_model gibt den Speicher eines Stereomodells wieder frei, das mit create_stereo_model angelegt wurde. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle StereoModelID ist nach dem Aufruf ungültig.

Parameter

▷ **StereoModelID** (input_control) ..... stereo_model ⇝ *handle*
Handle des Stereomodells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- StereoModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Modul

3D Metrology

create_stereo_model (: : CameraSetupModelID, Method, GenParamName, GenParamValue : StereoModelID)

Anlegen eines HALCON-Stereomodells.

Der Operator create_stereo_model erstellt ein HALCON-Stereomodell und gibt einen Handle in StereoModelID zurück. Das Modell bietet Funktionalität zur Stereorekonstruktion von 3D-Punkten oder Oberflächen mit dem kalibrierten Mehrbild-Kamerasetup, der im Parameter CameraSetupModelID übergeben wird (siehe Kalibrierung / Mehrbild für weitere Information übers Kalibrieren von Mehrkamerasetups).

Mit Method='*points_3d*' wird ein Stereomodell erstellt, das, nachdem es konfiguriert wurde, dem Operator reconstruct_points_stereo übergeben werden kann. Dieser Operator rekonstruiert 3D-Punkte, indem er die Sichtlinien schneidet, die von Punktkorrespondenzen aus den mehreren Bildern entstehen (siehe reconstruct_points_stereo für weitere Information).

Mit Method='*surface_pairwise*' oder Method='*surface_fusion*' wird ein Stereomodell erstellt, das, nachdem es konfiguriert wurde, dem Operator reconstruct_surface_stereo übergeben werden kann. Dieser Operator berechnet Disparitätsbilder für Bildpaare aus einem kalibrierten Mehrbild-Kameraaufbau und fusioniert die so erhaltene 3D-Information in eine Oberflächenrekonstruktion (siehe reconstruct_surface_stereo für weitere Information).

Die Parameter GenParamName und GenParamValue können benutzt werden, um generische Modellparameter einzustellen. Alternativ können diese Parameter im Nachhinein mit dem Operator set_stereo_model_param eingestellt werden, bevor der betroffene Rekonstruktionsoperator aufgerufen wird (siehe set_stereo_model_param für eine detaillierte Beschreibung der möglichen Parameter).

Parameter

▷ **CameraSetupModelID** (input_control) . . . . . camera_setup_model ⇝ *handle*
Handle des kalibrierten Kamerasetupmodells.

▷ **Method** (input_control) . . . . . string ⇝ *string*
Generische Rekonstruktionsmethode.
Defaultwert: 'surface_pairwise'
Werteliste: Method ∈ {'surface_pairwise', 'surface_fusion', 'points_3d'}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Modellparameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'bounding_box', 'persistence', 'sub_sampling_step', 'rectif_interpolation', 'rectif_method', 'disparity_method', 'binocular_method', 'binocular_num_levels', 'binocular_mask_width', 'binocular_mask_height', 'binocular_texture_thresh', 'binocular_score_thresh', 'binocular_filter', 'binocular_sub_disparity', 'binocular_mg_gray_constancy', 'binocular_mg_gradient_constancy', 'binocular_mg_smoothness', 'binocular_mg_initial_guess', 'binocular_mg_default_parameters', 'binocular_mg_solver', 'binocular_mg_cycle_type', 'binocular_mg_pre_relax', 'binocular_mg_post_relax', 'binocular_mg_initial_level', 'binocular_mg_iterations', 'binocular_mg_pyramid_factor', 'binocular_ms_surface_smoothing', 'binocular_ms_edge_smoothing', 'binocular_ms_consistency_check', 'binocular_ms_similarity_measure', 'binocular_ms_sub_disparity', 'point_meshing', 'resolution', 'surface_tolerance', 'min_thickness', 'smoothing', 'color', 'color_invisible', 'min_disparity', 'max_disparity'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Werte der generischen Modellparameter.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {-1, -2, -5, 0, 0.3, 0.5, 0.9, 1, 2, 3, 'census_dense', 'census_sparse', 'binocular', 'ncc', 'none', 'sad', 'ssd', 'bilinear', 'viewing_direction', 'geometric', 'false', 'very_accurate', 'accurate', 'fast_accurate', 'fast', 'v', 'w', 'none', 'gauss_seidel', 'multigrid', 'true', 'poisson', 'isosurface', 'interpolation', 'left_right_check', 'full_multigrid', 'binocular_mg', 'binocular_ms', 'smallest_distance', 'mean_by_distance', 'line_of_sight', 'mean_by_line_of_sight', 'median'}

▷ **StereoModelID** (output_control) . . . . . stereo_model ⇝ *handle*
Handle des Stereomodells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_stereo_model_param`, `set_stereo_model_image_pairs`, `reconstruct_surface_stereo`, `reconstruct_points_stereo`

Modul

3D Metrology

get_stereo_model_image_pairs (: : StereoModelID : From, To)

Liste der im Stereomodell gesetzten Bildpaare zurückgeben.

Der Operator `get_stereo_model_image_pairs` gibt die Liste der im Stereomodell gespeicherten Bildpaare zurück. Die Kameraindizes der *from*- und *to*-Kameras werden in den Parametern `From` bzw. `To` zurückgegeben (die Begriffe "from"und "to"bedeuten, dass bei der Rekonstruktion die Disparität *von* einem *zu* dem anderem Bild des Paars berechnet wird). Die Indizes bezeichnen Kameras aus dem Kamerasetupmodell, das an das Stereomodell während seiner Erstellung angehängt worden ist (siehe `create_stereo_model`).

Parameter

▷ **StereoModelID** (input_control) . . . . . stereo_model ⇝ *handle*
Handle des Stereomodells.

▷ **From** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der *from* - Kameras der Bildpaare.

▷ **To** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der *to* - Kameras der Bildpaare.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`set_stereo_model_image_pairs`

Nachfolger

`reconstruct_surface_stereo`

Siehe auch

`set_stereo_model_image_pairs`

Modul

3D Metrology

get_stereo_model_object (: Object : StereoModelID, PairIndex, ObjectName :)

Ikonische Zwischenergebnisse der Stereorekonstruktion abfragen.

Mit dem Operator get_stereo_model_object können ikonische Zwischenergebnisse der Oberflächenrekonstruktion, welche vom Operator reconstruct_surface_stereo fürs Stereomodell StereoModelID durchgeführt wird, abgefragt werden. Diese Ergebnisse können insbesondere dann nützlich sein, wenn ein Problem in der Rekonstruktion zurückverfolgt werden muss. Es ist zu beachten, dass die ikonische Zwischenergebnisse erst im *'persistence'*-Modus im Stereomodell gespeichert werden. Dieser Modus muss *vor* dem Rekonstruktionsoperator eingeschaltet werden (siehe set_stereo_model_param). Die ikonische Ergebnisse werden einzelnen Bildpaaren zugeordnet (siehe get_stereo_model_image_pairs).

Die Kameraindizes [From, To] des gewünschten Bildpaares werden im Parameter PairIndex übergeben. Je nach Auswahl in ObjectName werden die folgenden ikonische Ergebnisse im Object zurückgegeben:

'from_image_rect', *'to_image_rect'*: Rektifizierte Bilder, die den Kameras *from* bzw. *to* entsprechen. Diese Bilder können genauer untersucht werden, um die Qualität der binokularen Stereobildrektifizierung zu inspizieren.

'disparity_image': Das Disparitätsbild für dieses Paar. Die Disparitätsqualität hat einen direkten Einfluss auf die Oberflächenrekonstruktion.

'score_image': Maßbild der Zuverlässigkeit des Disparitätsbilds.

Eine Diskrepanz in den rektifizierten Bildern, d.h. wenn entsprechende Objektmerkmale in unterschiedlichen Zeilen beider Bilder erscheinen, oder Fehler im Disparitäts- oder Maßbild haben direkten Einfluss auf die Qualität der Oberflächenrekonstruktion. Es ist deswegen empfehlenswert, alle festgestellten Fehler durch Anpassen der Stereomodellparameter (siehe set_stereo_model_param) zu beseitigen, insbesondere diejenige, welche die internen Aufrufe zu gen_binocular_rectification_map und binocular_disparity bzw. binocular_disparity_mg oder binocular_disparity_ms steuern (siehe set_stereo_model_image_pairs und reconstruct_surface_stereo für weitere Information).

Parameter

▷ **Object** (output_object) ..... object(-array) ⇝ *object*
Ikonisches Ergebnis.

▷ **StereoModelID** (input_control) ..... stereo_model ⇝ *handle*
Handle des Stereomodells.

▷ **PairIndex** (input_control) ..... number(-array) ⇝ *integer* / string / real
Kameraindizes des Bildpaars ([From, To]).
Wertevorschläge: PairIndex ∈ {0, 1, 2}

▷ **ObjectName** (input_control) ..... string ⇝ *string*
Name des ikonischen Ergebnisses.
Wertevorschläge: ObjectName ∈ {'from_image_rect', 'to_image_rect', 'disparity_image', 'score_image'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

reconstruct_surface_stereo

Modul

3D Metrology

get_stereo_model_object_model_3d (: : StereoModelID, GenParamName : ObjectModel3D)

3D-Objektmodell als Zwischenergebnis der Stereorekonstruktion abfragen.

Mit dem Operator get_stereo_model_object_model_3d kann ein mit reconstruct_surface_stereo rekonstruiertes 3D Objektmodell ObjectModel3D abgefragt werden, das als Zwischenergebnis bei der Methode=*'surface_fusion'* erzeugt wurde. Das zurückgegebene

Objektmodell entspricht dem Ergebnis von reconstruct_surface_stereo unter Verwendung der Methode='surface_pairwise'.

Dazu muss beim Aufruf von get_stereo_model_object_model_3d der Parameter GenParamName auf den Wert 'm3d_pairwise' gesetzt werden. Weiterhin ist zu beachten, dass das Modell nur mit eingeschaltetem persistence-Modus abgefragt werden kann. Dieser Modus muss *vor* dem Aufruf von reconstruct_surface_stereo mit Hilfe von set_stereo_model_param aktiviert werden. Weiterhin kann das Objektmodell nur abgefragt werden, wenn das Stereomodell mit der Methode Methode='surface_fusion' erzeugt wurde. Andernfalls wird ein Fehler zurückgegeben. Falls kein Objektmodell rekonstruiert wurde, so gibt der Operator den Wert -1 zurück.

Parameter

- ▷ **StereoModelID** (input_control) ..... stereo_model ⇝ *handle*
  Handle des Stereomodells.
- ▷ **GenParamName** (input_control) ..... string(-array) ⇝ *string*
  Namen der Modellparameter.
  Werteliste: GenParamName ∈ {'m3d_pairwise'}
- ▷ **ObjectModel3D** (output_control) ..... object_model_3d ⇝ *handle*
  Rekonstruiertes 3D-Objektmodell.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

reconstruct_surface_stereo, set_stereo_model_param

Siehe auch

set_stereo_model_param

Modul

3D Metrology

get_stereo_model_param (: : StereoModelID,
GenParamName : GenParamValue)

Stereomodellparameter abfragen

Der Operator get_stereo_model_param kann verwendet werden, um diverse Parameter des Stereomodells StereoModelID abzufragen. Der Name des gewünschten Parameters wird mit GenParamName übergeben und der entsprechende Parameterwert wird im GenParamValue zurückgegeben. Zwei Typen von Parametern werden unterschieden: generische Stereoparameter bzw. spezifische Parameter für die Oberflächenrekonstruktion. Es ist zu beachten, dass es keine spezifischen Parameter für die 3D-Punktrekonstruktion gibt.

Bis auf den Parameter 'binocular_mg_default_parameters' können alle Parameter, die mit set_stereo_model_param gesetzt werden durch einen Aufruf von get_stereo_model_param abgefragt werden - siehe set_stereo_model_param für weitere Beschreibung einzelner Parameter. Die folgenden Parameter werden dagegen von anderen Operatoren gesetzt und können nicht mit set_stereo_model_param geändert, aber von get_stereo_model_param abgefragt werden:

Generische Parameter

'type': Typ des Stereomodells (zur Zeit entweder 'surface_pairwise', 'surface_fusion' oder 'points_3d').

'camera_setup_model': Handle zu einer *Kopie* des dem Stereomodell zugeordneten Kamerasetupmodells. Änderungen in der Kopie ändern das Kamerasetupmodell im Stereomodell nicht.

Diese Parameter werden während Erstellens des Modells von create_stereo_model gesetzt.

Bemerkung über tupelwertige Modellparameter

Die meisten der Stereomodellparameter haben nur einen Wert, d.h. eine Liste (d.h. ein Tupel) kann als Parameternamen übergeben werden und eine Liste (auch Tupel) derselben Länge von Parameterwerten wird als Ausgabe zurückgegeben. Tupelwertige Parameter dagegen geben eine Liste von Werten für den einen Parameter zurück. Wenn so ein Parameter mit einwertigen Parametern zusammen abgefragt wird, kann nicht mehr unterschieden werden, welcher Wert zu welchem Parameternamen gehört. Deswegen müssen tupelwertige Parameter, wie z.B. *'bounding_box'*, *'min_disparity'* oder *'max_disparity'* immer einzeln mit get_stereo_model_param abgefragt werden.

Parameter

- ▷ **StereoModelID** (input_control) . . . . . stereo_model ⇝ *handle*
  Handle des Stereomodells.
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Namen der Modellparameter.
  Werteliste: GenParamName ∈ {'type', 'camera_setup_model', 'bounding_box', 'persistence', 'sub_sampling_step', 'rectif_interpolation', 'rectif_sub_sampling', 'rectif_method', 'disparity_method', 'binocular_method', 'binocular_num_levels', 'binocular_mask_width', 'binocular_mask_height', 'binocular_texture_thresh', 'binocular_score_thresh', 'binocular_filter', 'binocular_sub_disparity', 'binocular_mg_gray_constancy', 'binocular_mg_gradient_constancy', 'binocular_mg_smoothness', 'binocular_mg_initial_guess', 'binocular_mg_solver', 'binocular_mg_cycle_type', 'binocular_mg_pre_relax', 'binocular_mg_post_relax', 'binocular_mg_initial_level', 'binocular_mg_iterations', 'binocular_mg_pyramid_factor', 'binocular_ms_surface_smoothing', 'binocular_ms_edge_smoothing', 'binocular_ms_consistency_check', 'binocular_ms_similarity_measure', 'binocular_ms_sub_disparity', 'min_disparity', 'max_disparity', 'point_meshing', 'poisson_depth', 'poisson_solver_divide', 'poisson_samples_per_node', 'resolution', 'surface_tolerance', 'min_thickness', 'smoothing', 'color', 'color_invisible'}
- ▷ **GenParamValue** (output_control) . . . . . attribute.value-array ⇝ *real* / integer / string
  Werte der Modellparameter.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_stereo_model, set_stereo_model_param

Nachfolger

reconstruct_surface_stereo, reconstruct_points_stereo

Siehe auch

set_stereo_model_param

Modul

3D Metrology

reconstruct_points_stereo (: : StereoModelID, Row, Column, CovIP, CameraIdx, PointIdx : X, Y, Z, CovWP, PointIdxOut)

3D-Punktrekonstruktion mit kalibriertem Mehrbild-Stereoaufbau.

Der Operator reconstruct_points_stereo rekonstruiert 3D-Punkte von Punktkorrespondenzen, die in einem vom kalibrierten Kamerasetup aufgenommenen Bildarray extrahiert worden sind. Das kalibrierte Kamerasetup wird an das Stereomodell StereoModelID während seines Erstellens angehängt (siehe create_stereo_model). Es ist zu beachten, dass das Stereomodell mit dem Typ *'points_3d'* erstellt werden muss, ansonsten gibt der Operator reconstruct_points_stereo einen Fehler zurück.

Die Punktkorrespondenzen werden in den Parametern `Row`, `Column`, `CameraIdx`, and `PointIdx` in der Form vier gleich langer Tupel übergeben. Jede Menge `(Row[I],Column[I],CameraIdx[I],PointIdx[I])` entspricht den Bildkoordinaten (`Row`, `Column`) eines 3D-Punktes (`PointIdx`) im Bild einer Kamera (`CameraIdx`).

Die rekonstruierten 3D-Punktkoordinaten werden in den drei gleich langen Ausgabetupeln `X`, `Y`, and `Z` zurückgegeben. Die Koordinaten sind im Koordinatensystems des am Stereomodell angehängten Kamerasetups definiert (siehe `create_camera_setup_model`). Das Tupel `PointIdxOut` enthält die entsprechenden Punktindizes.

Der Rekonstruktionsalgorithmus funktioniert wie folgt: Erst werden die Punktkorrespondenzen eines 3D-Punkts gesucht, indem alle Eingabemengen mit demselben `PointIdx` aufgesammelt werden. Danach werden die Informationen `Row`, `Column`, und `CameraIdx` von den angesammelten Mengen benutzt, um Sichtlinien von der jeweiligen Kamera durch den entsprechenden Bildpunkt `[Row,Column]` zu projizieren. Wenn 2 oder mehr Sichtlinien für einen Punkt `PointIdx` gefunden werden, werden diese alle in der Welt geschnitten und der Schnittpunkt als Ausgabemenge `(X[J],Y[J],Z[J],PointIdxOut[J])` gespeichert. Der Schnittpunkt wird durch die Methode der kleinsten Quadrate berechnet, ohne potenzielle falsche Sichtlinien zu suchen und sie zu unterdrücken (z.B. wenn ein Bildpunkt fälschlicherweise einem 3D-Punkt zugeordnet wird).

Zur Berechnung der Kovarianzmatrizen der rekonstruierten 3D-Punkte müssen statistische Informationen über die extrahierten Bildkoordinaten (d.h. Kovarianzmatrizen der Bildpunkte; siehe z.B. `points_foerstner`) verfügbar sein und in dem Parameter `CovIP` übergeben werden. Falls jedoch keine Kovarianzmatrizen für die rekonstruierten 3D-Punkte benötigt werden oder keine Kovarianzmatrizen für die Bildpunkte verfügbar sind, kann ein leeres Tupel an `CovIP` übergeben werden. Somit wird dann keine Kovarianzmatrix für die 3D Punkte berechnet.

Die Kovarianzmatrix eines Bildpunktes ist:

$$\texttt{CovIP} = \begin{bmatrix} (sigma_r)^2 & sigma_rc \\ sigma_rc & (sigma_c)^2 \end{bmatrix}$$

Die Kovarianzmatrizen sind symmetrische `2x2` Matrizen, bei denen die Einträge der Hauptdiagonalen die Varianz der Bildpunkte in Zeilenrichtung bzw. Spaltenrichtung repräsentieren. Für jeden Bildpunkt muss im Parameter `CovIP` eine Kovarianzmatrix in Form eines Tupels mit 4 Elementen übergeben werden:

$$[(sigma_r)^2, sigma_rc, sigma_rc, (sigma_c)^2].$$

Somit ist `|CovIP|=4*|Row|` und `CovIP[I*4:I*4+3]` sind die 4 Kovarianzmatrixwerte des `I.` Bildpunktes.

Die berechnete Kovarianzmatrix für einen erfolgreich rekonstruierten 3D-Punkt ist durch eine symmetrische `3x3` Matrix definiert:

$$\texttt{CovWP} = \begin{bmatrix} (sigma_x)^2 & sigma_xy & sigma_xz \\ sigma_yx & (sigma_y)^2 & sigma_yz \\ sigma_zx & sigma_zy & (sigma_z)^2 \end{bmatrix}$$

Die Einträge der Hauptdiagonalen repräsentieren die Varianzen der rekonstruierten 3D-Punkte in Richtung der x-, y- und z-Achse. Die berechneten Matrizen werden im Parameter `CovWP` in Form von Tupeln, bestehend aus 9 Elementen, zurückgegeben:

$$[(sigma_x)^2, sigma_xy, sigma_xz, sigma_yx, (sigma_y)^2, sigma_yz, sigma_zx, sigma_zy, (sigma_z)^2].$$

Somit ist `|CovWP|=9*|X|` und `CovWP[J*9:J*9+8]` sind die 9 Kovarianzmatrixwerte des `J.` 3D-Punktes. Es ist zu beachten, dass, wenn der an dem Stereomodell angehängte Kamerasetup Kovarianzmatrizen für die internen Kameraparameter enthält, diese in der Berechnung der `CovWP` auch miteinbezogen werden.

Wenn im Stereomodell ein gültiger kleinster umschließender Quader definiert ist (siehe `set_stereo_model_param`), werden Ausgabe-3D-Punkte außerhalb des Quaders ausgeschnitten. Ist der Quader ungültig, wird er nicht berücksichtigt und alle Punkten zurückgegeben, die gefunden werden könnten.

Parameter

- ▷ **StereoModelID** (input_control) . . . . . stereo_model ⇝ *handle*
  Handle des Stereomodells.
- ▷ **Row** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Zeilenkoordinaten der extrahierten Bildmerkmale.
- ▷ **Column** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Spaltenkoordinaten der extrahierten Bildmerkmale.
- ▷ **CovIP** (input_control) . . . . . number-array ⇝ *real* / integer
  Kovarianzmatrizen der extrahierten Bildmerkmale.
  Defaultwert: []
- ▷ **CameraIdx** (input_control) . . . . . number ⇝ *integer*
  Kameraindizes der Kameras, in welchen die Bildmerkmale extrahiert worden sind.
  Wertevorschläge: `CameraIdx` ∈ {0, 1, 2}
- ▷ **PointIdx** (input_control) . . . . . number ⇝ *integer*
  Zuordnung der extrahierten Bildmerkmale zu den beobachteten 3D-Punkten.
  Wertevorschläge: `PointIdx` ∈ {0, 1, 2}
- ▷ **X** (output_control) . . . . . real(-array) ⇝ *real*
  X-Koordinaten der rekonstruierten 3D-Punkte.
- ▷ **Y** (output_control) . . . . . number(-array) ⇝ *real*
  Y-Koordinaten der rekonstruierten 3D-Punkte.
- ▷ **Z** (output_control) . . . . . number(-array) ⇝ *real*
  Z-Koordinaten der rekonstruierten 3D-Punkte.
- ▷ **CovWP** (output_control) . . . . . number(-array) ⇝ *real*
  Kovarianzmatrizen der rekonstruierten 3D-Punkte.
- ▷ **PointIdxOut** (output_control) . . . . . number(-array) ⇝ *integer*
  Indizes der rekonstruierten 3D-Punkte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`reconstruct_surface_stereo`, `intersect_lines_of_sight`

Modul

3D Metrology

reconstruct_surface_stereo (
    Images : : StereoModelID : ObjectModel3D)

Oberflächenrekonstruktion mit kalibriertem Mehrbild-Stereoaufbau.

Der Operator `reconstruct_surface_stereo` rekonstruiert eine Oberfläche aus den Bildern `Images`, welche vom am Stereomodell `StereoModelID` angehängten kalibrierten Kamerasetup aufgenommen worden sind. Die rekonstruierte Oberfläche wird im HALCON-3D-Objektmodell `ObjectModel3D` zurückgegeben.

Vorbereitung und Voraussetzungen

Die Zusammenfassung der notwendigen Vorbereitungsschritte:

1. Kalibriertes Kamerasetup erstellen oder einlesen (mit `calibrate_cameras` kann beispielsweise ein Mehrkamerasystem kalibriert, mit `create_camera_setup_model` erstellt und konfiguriert werden).
2. Stereomodell mit `create_stereo_model` unter Angabe der Methode `Method`=*'surface_pairwise'* oder *'surface_fusion'* erstellen (siehe 'Rekonstruktionsalgorithmus').
3. Parameter für die Bildrektifizierung mit `set_stereo_model_param` setzen und anschließend die Bildpaare mit `set_stereo_model_image_pairs` auswählen.

4. Umschließenden Quader mit set_stereo_model_param konfigurieren (GenParamName='bounding_box').
5. Stereomodellparameter für paarweise Rekonstruktion mit set_stereo_model_param konfigurieren.
6. Für Modelle mit Method='surface_fusion' die Parameter für den Fusionsalgorithmus mit set_stereo_model_param konfigurieren.
7. Bilder mit dem kalibrierten Kamerasetup aufnehmen und ins Bildarray Images packen.
8. Oberflächenrekonstruktion mit reconstruct_surface_stereo durchführen.
9. Zwischenergebnisse mit get_stereo_model_object und get_stereo_model_object_model_3d abfragen und analysieren.
10. Nachjustieren der Stereomodellparameter für Verbesserungen in Qualität und Laufzeit mit set_stereo_model_param.

Ein **Kamerasetupmodell** wird dem Stereomodell StereoModelID bei seinem Erstellen mit create_stereo_model angehängt. Das Kamerasetup muss *kalibrierte Information* über die Kameras enthalten, mit denen die Bilder in dem Bildarray Images aufgenommen worden sind: Das I. Bild des Arrays entspricht der Kamera mit Index I-1 des Kamerasetups; die Anzahl der Bilder muss mit der Anzahl von Kameras im Setup übereinstimmen. Die Bilder in Images müssen eine statische Szene darstellen oder alle gleichzeitig aufgenommen werden, ansonsten kann sich die Oberflächenrekonstruktion als unmöglich erweisen.

Ein gut kalibriertes Kamerasetup ist die wichtigste Voraussetzung für eine präzise Oberflächenrekonstruktion. Die Kamerakalibrierung für ein Mehrkamerasetup muss daher besonders sorgfältig durchgeführt werden. HALCON bietet Kalibrierung von Mehrkamerasystemen mit dem Operator calibrate_cameras an. Das resultierende kalibrierte Kamerasetup kann mit einem nachfolgenden Aufruf von get_calib_data gelesen werden. Sollten Kameraparameter und -posen über einen anderen Weg gewonnen worden sein, so kann alternativ ein Kamerasetup mit dem Operator create_camera_setup_model erstellt werden.

Ein gut ausgewählter Satz von **Bildpaaren** (siehe set_stereo_model_image_pairs) spielt eine wichtige Rolle für die allgemeine Qualität der Rekonstruktion. Einerseits werden Kamerapaare mit kleinerem Abstand untereinander bevorzugt, weil sie sich besser für binokulare Disparitätsalgorithmen eignen. Andererseits sollen die Paare möglichst unterschiedliche Blickwinkel der Szene liefern, sodass, wenn ein Paar einen bestimmten Bereich der Oberfläche nicht *sieht*, dieser von einem anderen Paar *abgedeckt* wird. Es ist außerdem zu beachten, dass die Laufzeit der paarweisen Rekonstruktion linear abhängig von der Anzahl der Bildpaare ist. Man soll deswegen so viele Bildpaare wie nötig und so wenig wie möglich auswählen, um einen Kompromiss zwischen Vollständigkeit der Oberfläche und Laufzeit des Operators finden zu können.

Dem Stereomodell wird ein **umschließender Quader** zugeordnet; ein gültiger Quader ist eine wichtige Voraussetzung für die Oberflächenrekonstruktion (siehe set_stereo_model_param für weitere Informationen). Der Rekonstruktionsalgorithmus braucht den umschließenden Quader aus drei Gründen:

- Erstens, falls die Werte für die Parameter MinDisparity und MaxDisparity nicht über die Operatoren create_stereo_model oder set_stereo_model_param gesetzt wurden, wird die Projektion des Quaders in jedes Bild eines Bildpaares berechnet, um die Werte für die Parameter MinDisparity und MaxDisparity des internen Aufrufs von binocular_disparity bzw. binocular_disparity_ms zu schätzen. Für den Fall, dass als Methode binocular_disparity_mg ausgewählt wurde, wird aus den Disparitätswerten eine geeignete Schätzung für die Parameter InitialGuess und InitialLevel abgeleitet. Die automatische Schätzung wird für diese Methode nur dann verwendet, wenn sie mit den Standardwerten für diese beiden Parameter aufgerufen wird. Andernfalls werden die Werte benutzt, die der Nutzer mit set_stereo_model_param gesetzt hat.
- Zweitens werden die standardmäßig gesetzten Werte für die Parameter der Fusionsmethode basierend auf dem umschließenden Quader berechnet. Sie werden bei einer Anpassung des umschließenden Quaders ebenfalls umgesetzt. Des Weiteren sollte der umschließende Quader sehr eng um das zu rekonstruierende Objekt gesetzt werden, um eine unnötige und oft sehr deutliche Erhöhung der Laufzeit zu vermeiden.
- Drittens werden Oberflächenfragmente, welche außerhalb des Quaders liegen, am Ende des Algorithmus von der Rekonstruktion herausgeschnitten. Ein viel zu großer Quader kann dazu führen, dass MinDisparity und MaxDisparity zu weit voneinander geschätzt werden, und damit den internen Aufruf von binocular_disparity, binocular_disparity_ms bzw. binocular_disparity_mg und demzufolge von reconstruct_surface_stereo verlangsamen. Ein viel zu kleiner Quader kann dazu führen, dass auch erwünschte Punkte aus der Rekonstruktion herausgeschnitten werden.

Der Algorithmus wird versuchen, eine geschlossene Oberfläche zu rekonstruieren. Falls das Objekt nur von einer Seite aus betrachtet und rekonstruiert wird, legt normalerweise das entgegengesetzte Ende des umschließenden Quaders fest, wo das rekonstruierte Objekt abgeschnitten wird.

Die **Stereomodellparameter** der paarweisen Rekonstruktion vor den Fusionsparametern gut zu setzen ist essentiell, da die paarweise Rekonstruktion des Objekts als Input für den Fusionsalgorithmus dient. Die Parameter werden bei `set_stereo_model_param` genau beschrieben. Großen Einfluss hat die Wahl der *'disparity_method'*. Die Objekte in der beobachteten Szene sollen auch bestimmte Oberflächeneigenschaften aufweisen, um die Rekonstruktion zu ermöglichen. Erstens soll die Oberfläche Lambertsche Eigenschaften aufweisen, d.h. die Beleuchtung der Oberfläche erscheint konstant unabhängig vom Blickwinkel. Zweitens sollte die Oberfläche ein ausreichendes Muster aufweisen, das möglichst keine wiederholenden Fragmente hat.

`get_stereo_model_object` liefert **Zwischenergebnisse** der Rekonstruktion, wie rektifizierte, Disparitäts- und Scorebilder. `get_stereo_model_object_model_3d` kann für Modelle mit `Method`=*'surface_fusion'* benutzt werden, um das Ergebnis der paarweisen Rekonstruktion anzuzeigen. Der Absatz "Problembehandlung bei der Konfiguration des Stereomodells"erklärt, wie die Zwischenergebnisse benutzt werden können.

Rekonstruktionsalgorithmus

Der Operator `reconstruct_surface_stereo` führt mehrere paarweise Stereorekonstruktionen durch und fügt die Ergebnisse anschließend zusammen. Die Bildpaare dieser paarweisen Rekonstruktion sind in `StereoModelID` als Kamerapaare des zugehörigen kalibrierten Kamerasetups gesetzt.

Für jedes Bildpaar werden die Bilder zunächst rektifiziert, bevor intern einer der Operatoren `binocular_disparity`, `binocular_disparity_mg` oder `binocular_disparity_ms` aufgerufen wird. Die Disparitätsinformation wird weiter durch einen internen Aufruf von `disparity_image_to_xyz` in Punkte im Koordinatensystem der `from`-Kamera umgewandelt. Im nächsten Schritt werden die Punkte in das allgemeine Koordinatensystem des an `StereoModelID` angehängten Kamerasetups transformiert und in einer Punktwolke zusammen mit den rekonstruierten Punkten aus den anderen Bildpaaren gespeichert.

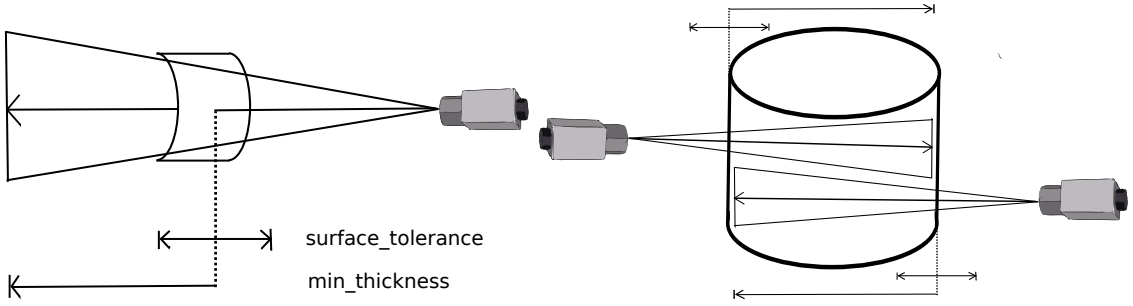
'surface_pairwise' Falls für das Stereomodell der Typ *'surface_pairwise'* gewählt wurde (siehe `create_stereo_model`), wird die so erhaltene Punktwolke direkt in `ObjectModel3D` zurückgegeben. Für jeden Punkt wird der Normalenvektor durch das Legen einer ausgleichenden Ebene durch die benachbarten 3D-Punkte berechnet. Im Gegensatz zu `surface_normals_object_model_3d` werden die benachbarten Punkte aber nicht in 3D, sondern einfach in 2D bestimmt, indem die in den X-, Y- und Z-Koordinatenbildern benachbarten Punkte verwendet werden. Der Normalenvektor jedes 3D-Punktes wird nun auf den Normalenvektor der entsprechenden Ebene gesetzt. Für jeden rekonstruierten 3D-Punkt wird das Qualitätsmaß seiner Disparität als erweitertes Attribut mit abgespeichert. Zusätzlich dazu ist ein Subsampling der transformierten Koordinatenbilder möglich. Wenn nur ein Bildpaar bearbeitet wird und keine Punktvermaschung stattfindet, speichert `reconstruct_surface_stereo` das Attribut `'xyz_mapping'` im `ObjectModel3D`. Dieses Attribut liefert die Zuordnung einzelner rekonstruierter 3D-Punkte zu den Bildkoordinaten des ersten Bildes aus dem Paar und ist eine Voraussetzung, um Operatoren wie `segment_object_model_3d` oder `object_model_3d_to_xyz` (mit `Type`=*'from_xyz_map'*) aufzurufen. Es ist zu beachten, dass, wenn mehrere Bildpaare bearbeitet werden, das Attribut nicht mehr mitgeliefert wird. Dies liegt daran, dass einzelne 3D-Punkte aus unterschiedlichen Bildpaaren stammen. Damit ist die Zuordnung zu allgemeinen Bildkoordinaten nicht mehr möglich. Ob das Attribut im Objekt gespeichert wurde, kann mit `get_object_model_3d_params` mit `GenParamName`=*'has_xyz_mapping'* abgefragt werden.

Die rekonstruierte Punktwolke `ObjectModel3D` kann in einem Nachverarbeitungsschritt vermascht werden. Für Stereomodelle vom Typ *'surface_pairwise'* wird nur der Poisson-basierte Vermaschungsalgorithmus unterstützt, der sich durch ein Setzen des Parameters *'point_meshing'* auf den Wert *'poisson'* einschalten lässt. Dieser Algorithmus generiert eine sogenannte wasserdichte Vermaschung, d.h. auch Regionen, welche keine Punktinformation enthalten, werden von der Vermaschung bedeckt.

'surface_fusion' Falls für das Stereomodell der Typ *'surface_fusion'* gewählt wurde, wird die resultierende Punktwolke der paarweisen Rekonstruktion noch weiter verarbeitet. Ziel ist es, eine möglichst glatte und trotzdem formtreue Oberfläche zu erhalten. Dazu wird der umschließende Quader gesampelt und jedem Sample-Punkt ein Distanzwert zu einer sogenannten Isosurface (bestehend aus Punkten mit Distanzwert 0) zugewiesen. Die endgültigen Distanzwerte (und somit die Isosurface) erhält man durch Minimierung einer Fehlerfunktion basierend auf den paarweise rekonstruierten Punkten. Dadurch erreicht man eine Fusion der paarweisen Einzel-Rekonstruktionen (siehe das an zweiter Stelle der Referenzen erwähnte Paper).

Die Berechnung der Isosurface kann in `set_stereo_model_param` mit den Parametern *'resolution'*, *'surface_tolerance'*, *'min_thickness'* und *'smoothing'* beeinflusst werden. Der Abstand zwischen Abtastpunkten im umschließenden Quader (in jeder Koordinatenrichtung) wird mit dem Parameter *'resolution'*

eingestellt. Der Parameter *'smoothing'* reguliert die 'Unruhe' der Distanzfunktion, indem er die zwei Terme der Fehlerfunktion zueinander gewichtet: Datentreue gegenüber der ursprünglichen Punktwolke aus der paarweisen Rekonstruktion einerseits und Totalvariation der Distanzfunktion andererseits. Der gewünschte Wert des Parameters für einen bestimmten Datensatz muss durch Ausprobieren gefunden werden. Zu kleine Werte für diesen Parameter führen dazu, dass die Oberfläche viele Ausreißer enthält. Dies ist sogar dann der Fall, wenn die Oberfläche deswegen große Sprünge aufweist. Zu große Werte hingegen führen dazu, dass der Algorithmus sich nicht mehr auf die Eingangspunktwolke, welche durch die paarweise Rekonstruktion erstellt wurde, stützt. Datentreue gegenüber der ursprünglichen Oberfläche aus der paarweisen Rekonstruktion wird nicht im gesamten umschließenden Quader betrachtet, sondern nur in Sichtkegeln von den Kameras zur paarweise rekonstruierten Oberfläche. Ein Abtastpunkt in einem solchen Kegel wird als sicher außerhalb des Objekts (vor der Oberfläche) oder sicher innerhalb des Objekts (hinter der Oberfläche) angesehen bezüglich dieser Kamera, wenn sein Abstand zur ursprünglichen Oberfläche einen bestimmten Wert übersteigt, der durch den Parameter *'surface_tolerance'* gesetzt werden kann. Die Länge des zu berücksichtigenden Sichtkegels einer Kamera hinter der mit der paarweisen Rekonstruktion erstellten Oberfläche kann durch den Parameter *'min_thickness'* gesetzt werden (siehe auch `set_stereo_model_param`). *'min_thickness'* darf nicht kleiner als *'surface_tolerance'* werden.



Die Parameter *'surface_tolerance'* und *'min_thickness'* regulieren die Datentreue gegenüber der ursprünglichen Punktwolke aus der paarweisen Rekonstruktion. Punkte im Sichtkegel einer Kamera werden als sicher außerhalb des Objekts (vor der Oberfläche) oder sicher innerhalb des Objekts (hinter der Oberfläche) angesehen bezüglich dieser Kamera, wenn ihr Abstand zur ursprünglichen Oberfläche *'surface_tolerance'* übersteigt. Punkte hinter der Oberfläche (von der gegebenen Kamera aus gesehen) werden nur als sicher innerhalb des Objekts angesehen, wenn ihr Abstand zur ursprünglichen Oberfläche *'min_thickness'* nicht übersteigt.

Die Punkte des in `ObjectModel3D` zurückgegebenen Objektmodells werden auf der Isofläche extrahiert, auf der die Distanzfunktion Null ist. Die Normalen werden aus dem Gradienten der Distanzfunktion berechnet. Für die Methode *'surface_fusion'* müssen zwar mehr Parameter eingestellt werden als für rein paarweise Rekonstruktion, die Nachbearbeitung der erzeugten Punktwolke wird aber meist stark vereinfacht. Besonders Ausreißerunterdrückung, Glättung, gleichmäßiges Subsampling und Füllen von Löchern kann mit dieser Methode in hoher Qualität erreicht werden. Dasselbe gilt für die mögliche Vermaschung der rekonstruierten Punktwolke, siehe den nächsten Absatz. Zu beachten ist auch, dass der Algorithmus versuchen wird, eine geschlossene Oberfläche zu rekonstruieren. Falls das Objekt nur von einer Seite aus betrachtet und rekonstruiert wird, legt normalerweise das entgegengesetzte Ende des umschließenden Quaders fest, wo das rekonstruierte Objekt abgeschnitten wird. Die Methode *'surface_fusion'* kann deutlich langsamer als einfache paarweise Rekonstruktion sein, was vor allem vom Parameter *'resolution'* beeinflusst wird.

Die rekonstruierte Punktwolke `ObjectModel3D` kann in einem Nachverarbeitungsschritt vermascht werden. Für Stereomodelle vom Typ *'surface_fusion'* wird der Algorithmus 'marching tetrahedra' verwendet, welcher durch ein Setzen des erwähnten Parameters auf den Wert *'isosurface'* aktiviert wird. Dabei ist zu beachten, dass das Objektmodell in `ObjectModel3D` bei eingeschalteter Vermaschung mehr Punkte aufweist also ohne diese Option. Dies ist auch bei gleichbleibender Auflösung der Fall.

Texturierung der Punktwolke

Es ist möglich, das mit `reconstruct_surface_stereo` rekonstruierte 3D-Objektmodell mit Farbinformationen aus den Eingabebildern zu versehen. Die Berechnung der Farbinformation erfolgt mit der in `set_stereo_model_param` gewählten Methode (siehe dortige Auflistung). Jeder 3D-Punkt erhält einen Farbwert bestehend aus einem Rot-, Grün-, und Blaukanal, die im ausgegebenen 3D-Objektmodell `ObjectModel3D` als Attribute mit den Namen *'red'*, *'green'* und *'blue'* gespeichert werden. Diese Attribute können zum Beispiel in der Prozedur `visualize_object_model_3d` mit `GenParamName` =

'red_channel_attrib', 'green_channel_attrib' und 'blue_channel_attrib' benützt werden. Die Attribute können auch mittels get_object_model_3d_params abgerufen werden oder mittels select_points_object_model_3d oder anderen Operatoren, die erweiterte Attribute verwenden, weiterverarbeitet werden. Für den Fall, dass zur Rekonstruktion Grauwertbilder verwendet werden, so wird der Farbwert in den drei Kanälen auf denselben Wert gesetzt. Falls Mehrkanalbilder verwendet werden, so wird für die Rekonstruktion des 3D-Objektmodells lediglich der erste Kanal verwendet. Die verbleibenden Kanäle werden ausschließlich zur Bestimmung der Farbinformation verwendet.

Falls bei der Rekonstruktion ein Stereomodell vom Typ *'surface_fusion'* verwendet wurde, so werden Punkte rekonstruiert, die keine direkte Korrespondenz zu Punkten im Bild haben. Diese Punkte sind somit unsichtbar, da sie von keiner Kamera aus gesehen werden. Diese 3D-Punkte erhalten ebenfalls Farbinformationen, indem ihnen der Farbwert ihres nächsten sichtbaren Nachbarn zugewiesen wird. Im Normalfall ist diese Berechnung nicht sehr zeitaufwändig und kann aktiviert bleiben. Es kann jedoch passieren, dass der Wert für den Parameter *'resolution'* viel zu fein für die vorliegende Bildauflösung eingestellt wurde. Dadurch werden sehr viele unsichtbare 3D-Punkte rekonstruiert. Dies führt dann zu einer sehr zeitaufwändigen Nachbarschaftssuche und damit zu einer stark erhöhten Laufzeit. In diesen Fällen sollte entweder der Parameters *'resolution'* angepasst werden oder die Bestimmung der Farbwerte für unsichtbare Punkte deaktiviert werden. Dies lässt sich realisieren, indem man den Wert des Parameters *'color_invisible'* über einen Aufruf von set_stereo_model_param auf *'false'* setzt. In diesem Fall werden die Farbwerte von unsichtbaren Punkten auf 255 gesetzt.

Problembehandlung bei der Konfiguration des Stereomodells

Eine richtige Vorbereitung eines Stereomodells ist nicht immer einfach. Bitte halten Sie sich an obenstehenden Ablauf. Wenn die Rekonstruktionsergebnisse nicht befriedigend sind, können die folgenden Verbesserungsmaßnahmen versucht werden:

persistence-Modus einschalten Wenn der *'persistence'*-Modus des Stereomodells eingeschaltet ist (Aufruf von set_stereo_model_param mit GenParamName=*'persistence'*), speichert der nachfolgende Aufruf von reconstruct_surface_stereo ikonische Zwischenergebnisse, welche zusätzliche Informationen zur Verfügung stellen. Diese Zwischenergebnisse können mit get_stereo_model_object und get_stereo_model_object_model_3d abgefragt werden.

Die Qualität der Kamerakalibrierung testen

- Wenn das Kamerasetup mit calibrate_cameras kalibriert worden ist, speichert dieser Operator auch Qualitätsinformationen, z.B. in Form von Standardabweichungen im Kamerasetupmodell ab. Das dem Stereomodell angehängte Kamerasetupmodell kann mit dem Operator get_stereo_model_param und GenParamName=*'camera_setup_model'* abgefragt werden, die Kameraparameter-Standardabweichungen danach mit dem Operator get_camera_setup_param. Ungewöhnlich hohe Werte der Standardabweichungen können auf eine schlechte Kamerakalibrierung hindeuten.
- Mit eingeschaltetem *'persistence'*-Modus lassen sich die rektifizierten Bilder jedes Bildpaares inspizieren. Sie werden vom Operator get_stereo_model_object mit dem entsprechenden Kameraindexpaar [From, To] im Parameter PairIndex und *'from_image_rect'* bzw. *'to_image_rect'* im ObjectName zurückgegeben. Die Bilder deuten auf eine gute Rektifizierung hin, wenn alle markanten Objektmerkmale in derselben Zeile beider Bilder erscheinen. Auch eine Diskrepanz in der Größenordnung von nur einigen Zeilen ist ein klarer Hinweis dafür, dass die Kalibrierung nicht optimal ist.

Den umschließenden Quader überprüfen Der umschließende Quader sollte sehr eng um das zu rekonstruierende Objekt bzw. das relevante Volumen gesetzt sein. Falls die Parameter MinDisparity und MaxDisparity nicht über create_stereo_model oder set_stereo_model_param manuell gesetzt wurden, benutzt der Algorithmus die Projektion des Quaders in jedes Bild jedes Bildpaares, um die Werte für die Parameter MinDisparity und MaxDisparity des internen Aufrufs von binocular_disparity bzw. binocular_disparity_ms zu schätzen. Die Disparitätswerte können mit get_stereo_model_param abgefragt werden und anschließend gegebenenfalls mit set_stereo_model_param angepasst werden, wodurch der kleinste umschließende Quader dann nur noch dafür verwendet wird, um die rekonstruierten 3D Punkte einzuschränken. Für den Fall, dass als Methode binocular_disparity_mg ausgewählt wurde, wird aus den berechneten Werten eine geeignete Schätzung für die Parameter InitialGuess und InitialLevel abgeleitet. Diese können mit set_stereo_model_param aber auch überschrieben werden.

Die Qualität der Disparitätsbilder verbessern Mit eingeschaltetem *'persistence'*-Modus lassen sich das Disparitäts- und das Maßbild jedes Bildpaares inspizieren. Sie werden vom Operator get_stereo_model_object mit dem entsprechenden Kameraindexpaar [From, To] im Parameter PairIndex und *'disparity_image'* bzw. *'score_image'* im ObjectName zurückgegeben. Wenn

beide Bilder erhebliche Mängel aufweisen (z.B. die tatsächliche Form der aufgenommenen Oberfläche lässt sich im Disparitätsbild gar nicht erkennen), müssen die Parameter, die den internen Aufruf des binocular_disparity steuern (d.h. alle Stereomodellparameter, die mit dem Präfix *'binocular_'* beginnen; siehe set_stereo_model_param), angepasst werden, bis eine Verbesserung erreicht wird.

Eine Alternative dazu stellt die Wahl einer alternativen Methode zur Berechnung der Disparitäten dar. HALCON stellt dabei neben dem bereits erwähnten internen Aufruf von binocular_disparity auch die die beiden anderen Methoden binocular_disparity_mg und binocular_disparity_ms zur Verfügung. Diese Verfahren zeichnen sich beispielsweise durch eine Berechnung von Disparitäten in texturlosen Bereichen aus, sind aber in den meisten Fällen etwas langsamer als die auf Kreuzkorrelation basierende Methode. Allerdings kann es auch für diese beiden Verfahren nötig sein, eine an die Anwendung spezifische Anpassung der Parameter vorzunehmen. Je nach gewähltem Verfahren kann der Nutzer die Parameter mit dem Präfix *'binocular_mg_'* bzw. *'binocular_ms_'* setzen, bis eine Verbesserung erzielt wurde. Eine ausführliche Beschreibung der Methoden und ihrer Parameter findet sich in binocular_disparity bzw. binocular_disparity_mg oder binocular_disparity_ms.

Fusionsparameter Wenn das mit get_stereo_model_object_model_3d inspizierte Ergebnis der paarweisen Rekonstruktion nicht weiter verbessert werden kann, kann begonnen werden, die Fusionsparameter anzupassen. Für ihre Beschreibung siehe auch set_stereo_model_param. In manchen Fällen ist die paarweise Rekonstruktion des Objekts kaum erkennbar, kann aber vom Fusionsalgorithmus noch zu einem vernünftigen Ergebnis verarbeitet werden. In jedem Fall sollte die paarweise Rekonstruktion genug Punkte als Input für die Fusion erzeugen.

Laufzeit

Um die Laufzeit zu verbessern, können die folgenden Verbesserungsmaßnahmen ausprobiert werden:

Größe des umschließenden Quaders Der umschließende Quader sollte sehr eng um das zu rekonstruierende Objekt bzw. das relevante Volumen gesetzt sein. Andernfalls wird die Laufzeit unnötig und - für die Methode *'surface_fusion'* - drastisch erhöht.

Verkleinern des Definitionsbereichs der Eingabebilder Das Verkleinern des Definitionsbereichs der Eingabebilder (z.B mit reduce_domain) auf den für die Rekonstruktion relevanten Bereich kann den Algorithmus, vor allem für große Bilder, deutlich beschleunigen.

Abtastfaktor im Rektifikationsschritt Der Stereomodellparameter *'rectif_sub_sampling'* (siehe set_stereo_model_param) steuert den Abtastfaktor im Rektifikationsschritt. Ein Faktor > 1.0 mindert die Auflösung der rektifizierten Bilder im Vergleich zu den originalen Bildern. Damit wird der nachfolgende interne Aufruf der ausgewählten Disparitätsmethode wesentlich beschleunigt; allerdings sinkt die Bild- bzw. Rekonstruktionsauflösung. Der Parameter *'rectif_interpolation'* kann auch eine Rolle bezüglich der Geschwindigkeit im Rekonstruktionsschritt spielen, ist dabei aber eher weniger wichtig.

Disparitätsparameter Das Ziel ist ein Kompromiss zwischen der Vollständigkeit der Oberflächenrekonstruktion und der Laufzeit. Im Stereomodell können drei verschiedene Methoden zur Berechnung der Disparitätsbilder gesetzt werden. Je nachdem, welche dieser Methoden ausgewählt wurde, steht dem Stereomodell ein spezieller Parametersatz zur Verfügung, der ein präzises Anpassen der Methode an den verwendeten Datensatz ermöglicht. Ist die Methode binocular_disparity gewählt, so können nur die Parameter, deren Name mit dem Präfix *'binocular_'* beginnen, gesetzt werden. Für die Methode binocular_disparity_mg müssen die Parameter mit dem Präfix *'binocular_mg_'* und für die Methode binocular_disparity_ms die Parameter mit dem Präfix *'binocular_ms_'* verwendet werden.

Parameter bei Verwendung von binocular_disparity

NumLevels, MaskWidth, MaskHeight, Filter, SubDisparity Jeder dieser Parameter von binocular_disparity hat einen entsprechenden Stereomodellparameter, dessen Name mit dem Präfix *'binocular_'* beginnt. Diese Parameter haben auch Einfluss auf die Laufzeit. Daher kann ein richtiges Anpassen dieser Parameter die Laufzeit des Operators verbessern.

Parameter bei Verwendung von binocular_disparity_mg

GrayConstancy, GradientConstancy, Smoothness, InitialGuess, MGSolver, MGCycleType, MGPreRel
Jeder dieser Parameter von binocular_disparity_mg hat einen entsprechenden Stereomodellparameter, dessen Name mit dem Präfix *'binocular_mg_'* beginnt. Diese Parameter haben neben einer Auswirkung auf das Ergebnis auch Einfluss auf die Laufzeit. Daher kann eine richtige Anpassung dieser Parameter die Laufzeit des Operators beeinflussen.

Parameter bei Verwendung von binocular_disparity_ms

SurfaceSmoothing, EdgeSmoothing, ConsistencyCheck, SimilarityMeasure, SubDisparity
Jeder dieser Parameter von binocular_disparity_ms hat einen entsprechenden Stereomodellparameter, dessen Name mit dem Präfix *'binocular_ms_'* beginnt. Diese Parameter haben neben einer Auswirkung auf das Ergebnis auch Einfluss auf die Laufzeit. Daher kann eine richtige Anpassung dieser Parameter die Laufzeit des Operators beeinflussen.

Rekonstruktion von Punkten mit gutem Qualitätsmaß Neben der Anpassung des Abtastschrittes ist es auch möglich, Punkte aufgrund ihres berechneten Qualitätsmaßes von der 3D-Rekonstruktion auszuschließen. Dazu sollten zuerst mit Hilfe eines Aufrufs von get_stereo_model_object unter Verwendung von GenParamName = 'score_image' die Maßbilder für die Disparitätswerte abgefragt werden. Der Nutzer kann anhand der Verteilung dieser Werte nun entscheiden, ob Disparitäten unterhalb einer bestimmten Schwelle von der Rekonstruktion ausgeschlossen werden sollen. Dies lässt sich über das Setzen des Parameters GenParamName = 'binocular_score_thresh' im Operator set_stereo_model_param bewerkstelligen. Wenn Punkte aufgrund ihres Qualitätsmaßes von der Rekonstruktion ausgeschlossen werden, kann die Laufzeit leicht verbessert werden, da nicht mehr der komplette Datensatz weiterverarbeitet werden muss. Da, wie eingangs erwähnt, für jeden 3D-Punkt das berechnete Qualitätsmaß als zugehöriges Punktattribut mit abgespeichert wird, können auch nach dem Aufruf von reconstruct_surface_stereo noch Punkte aufgrund ihres Qualitätsmaßes von der Punktwolke entfernt werden. Dies hat den Vorteil, dass ein erneutes Aufrufen von reconstruct_surface_stereo nicht nötig ist. Allerdings muss die Punktwolke während der Rekonstruktion im Gesamten berechnet werden, was eine höhere Laufzeit zur Folge hat.

Abtastschritt der X-,Y-,Z-Daten Für die Methode *'surface_pairwise'* kann ein größerer Abtastschritt der X-,Y-,Z-Daten im letzten Rekonstruktionsschritt verwendet werden, indem GenParamName=*'sub_sampling_step'* mit set_stereo_model_param gesetzt wird. Damit werden die rekonstruierten Punktwolken deutlich dünner, was ihre Nachbearbeitung beschleunigt.

Fusionsparameter Die Methode *'surface_fusion'* kann durch Erhöhen des Parameters *'resolution'* deutlich beschleunigt werden.

Parameter

▷ **Images** (input_object) ..... singlechannelimage-array ⇝ *object :* byte
Bildarray, das von dem dem Stereomodell angehängten kalibrierten Kamerasetup aufgenommen wurden.

▷ **StereoModelID** (input_control) ..... stereo_model ⇝ *handle*
Handle des Stereomodells.

▷ **ObjectModel3D** (output_control) ..... object_model_3d ⇝ *handle*
Handle der rekonstruierten Oberfläche.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

create_stereo_model, get_calib_data, set_stereo_model_image_pairs

Nachfolger

get_stereo_model_object_model_3d

Alternativen

reconstruct_points_stereo

Literatur

M. Kazhdan, M. Bolitho, and H. Hoppe: „Poisson Surface Reconstruction.“ Symposium on Geometry Processing (June 2006).,

C. Zach, T. Pock, and H. Bischof: „A globally optimal algorithm for robust TV-L1 range image integration.“ Proceedings of IEEE International Conference on Computer Vision (ICCV 2007).

Modul

3D Metrology

set_stereo_model_image_pairs (: : StereoModelID, From, To :)

Definition der Bildpaare für Oberflächenrekonstruktion

Der Operator set_stereo_model_image_pairs speichert eine Liste von Bildpaaren ins Stereomodell StereoModelID des Typs *'surface_pairwise'* oder *'surface_fusion'*. Der Operator gibt einen Fehler zurück, falls er für ein Stereomodell von einem anderen Typ aufgerufen wird. Ein Stereomodell vom Typ *'surface_pairwise'* oder *'surface_fusion'* rekonstruiert Oberflächen anhand der Disparitätsbilder der definierten Bildpaare. Die Oberflächenrekonstruktion berechnet dann das Disparitätsbild vom Bild der Kamera mit dem Index From[0] zu den Kamera mit dem Index To[0].

Die Kameraindizes müssen gültig für das dem Stereomodell angehängte Kamerasetupmodell sein (siehe create_stereo_model), ansonsten wird ein Fehler zurückgegeben. Falls im Modell bereits eine Liste von Bildpaaren definiert ist, wird sie durch die neue ersetzt.

Neben dem Speichern der Bildpaarliste bereitet der Operator set_stereo_model_image_pairs ein Paar von Rektifizierungs-Abbildungsvorschriften vor, welche danach wiederholt im Operator reconstruct_surface_stereo verwendet werden, um die entsprechende Bildpaare zu rektifizieren; siehe gen_binocular_rectification_map für weitere Information. Drei der gen_binocular_rectification_map-Parameter werden als Stereomodellparameter zugänglich gemacht und können mit set_stereo_model_param gesetzt bzw. mit get_stereo_model_param abgefragt werden:

'rectif_interpolation': Interpolationsmodus, entspricht dem Parameter MapType des Operators gen_binocular_rectification_map.

'rectif_sub_sampling': Abtastfaktor, entspricht dem Parameter SubSampling des Operators gen_binocular_rectification_map.

'rectif_method': Rektifizierungsmethode, entspricht dem Parameter Method des Operators gen_binocular_rectification_map.

Es ist zu beachten, dass nach Änderung eines von dieser Parameter, der Operator set_stereo_model_image_pairs erneut ausgeführt werden muss, damit die Änderungen eine Wirkung haben.

Die aktuelle Liste der Bildpaare kann mit dem Operator get_stereo_model_image_pairs abgefragt werden.

Parameter

▷ **StereoModelID** (input_control) ..... stereo_model ⇝ *handle*
Handle des Stereomodells.

▷ **From** (input_control) ..... integer-array ⇝ *integer*
Indizes der *from*-Kameras der Bildpaare.
Parameteranzahl: From > 0

▷ **To** (input_control) ..... integer-array ⇝ *integer*
Indizes der *to*-Kameras der Bildpaare.
Parameteranzahl: To == From

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- StereoModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_stereo_model

Nachfolger

reconstruct_surface_stereo

Siehe auch

set_stereo_model_param, get_stereo_model_image_pairs

Modul

3D Metrology

set_stereo_model_param (: : StereoModelID, GenParamName, GenParamValue :)

Parameter eines Stereomodells setzen.

Mit dem Operator set_stereo_model_param können diverse Parameter im Stereomodell StereoModelID gesetzt werden. Dabei werden in Abhängigkeit des mit create_stereo_model gesetzten Typs des Stereomodells verschiedene Typen von Parametern zur Verfügung gestellt. Weiterhin ist zu beachten, dass keine spezifischen Parameter für die Methode *'points_3d'* zur Verfügung stehen.

Generische Parameter:

Je nach Auswahl im GenParamName können die folgenden generischen Modellparameter auf den Wert im Parameter GenParamValue gesetzt werden:

'bounding_box': Durch Übergabe des Tupels [x1,y1,z1,x2,y2,z2] können die zwei gegenüberliegenden Eckpunkte P1=[x1,y1,z1] bzw. P2=[x2,y2,z2] des kleinsten umschließenden Quaders für die Stereorekonstruktion übergeben werden. Dieser Quader ist achsenparallel im Koordinatensystem des Kamerasetups definiert (vorgegeben mit dem Handle CameraSetupModelID beim Erstellen des Stereomodells mit create_stereo_model). Der entsprechende Rekonstruktionsalgorithmus schneidet Rekonstruktionsergebnisse weg, welche außerhalb dieses Quaders liegen.

Falls der Disparitätsbereich nicht über die Operatoren create_stereo_model oder set_stereo_model_param gesetzt wurde, wird der umschließende Quader vom Operator reconstruct_surface_stereo benötigt, um den Disparitätsbereich einzelner Bildpaare bestimmen zu können (siehe set_stereo_model_image_pairs für weitere Informationen).

Weiterhin ist zu beachten, dass für Stereomodelle vom Typ *'surface_fusion'* die Werte für die Parameter beim Umsetzen des umschließenden Quaders auf den Standardwert zurückgesetzt werden.

Die Prozedur estimate_bounding_box_3d_reconstruction kann benutzt werden, um initiale Werte für den umschließenden Quader Ihrer 3D Rekonstruktion zu erhalten. Dieser umschließende Quader basiert dann auf der Pose einer Referenzkalibrierplatte und den Sichtkegeln der Kameras. Später sollte der umschließende Quader dann so eng wie möglich um das zu rekonstruierende Objekt eingestellt werden.

Außerdem können die Prozeduren gen_bounding_box_object_model_3d und gen_camera_setup_object_model_3d benutzt werden, um das Kamerasetup zu visualisieren.

Bei einem gültigen Quader liegt der Punkt P1 an der Ecke vorne links unten und der Punkt P2 hinten rechts oben; d.h. x1<x2, y1<y2 und z1<z2. Während die Oberflächenrekonstruktion (siehe reconstruct_surface_stereo) bei ungültigen Quadereinstellung abbricht und einen Fehler zurückgibt, wird der Punktrekonstruktionsalgorithmus (siehe reconstruct_points_stereo) einen ungültigen Quader einfach auslassen und wird alle Punkte rekonstruieren, die er kann. Daher kann das Abschneiden von 3D-Punkten ausgeschaltet werden, indem ein Tupel [0,0,0,0,0,-1] als kleinster umschließender Quader eingegeben wird.

Es ist zu beachten, dass *'bounding_box'* ein tupelwertiger Parameter ist und daher nicht zusammen mit anderen Parametern in einem Aufruf des Operators set_stereo_model_param gesetzt werden kann (siehe Paragraph "Bemerkung über tupelwertige Modellparameter").

Um das Einstellen des umschließenden Quaders zu erleichtern, stehen die Prozeduren gen_bounding_box_object_model_3d und gen_camera_setup_object_model_3d zur Verfügung. Mithilfe dieser Prozeduren kann man das Kamerasetup visualisieren und beispielsweise den umschließenden Quader anfänglich so einstellen, dass nur der Bereich rekonstruiert wird, in dem sich die Sichtkegel der Kameras überschneiden. Später sollte der umschließende Quader dann so eng wie möglich um das zu rekonstruierende Object eingestellt werden.

Tupelformat: [x1,y1,z1,x2,y2,z2].

'persistence': Schaltet den *'persistence'*-Modus des Stereomodells ein (GenParamValue=*1*) oder aus (GenParamValue=*0*). Im eingeschalteten Modus speichert das Modell Zwischenergebnisse des Rekonstruktionsverfahren (nur für reconstruct_surface_stereo), die im Nachhinein mit dem Operator get_stereo_model_object und get_stereo_model_object_model_3d abgefragt werden können.

Es ist zu beachten, dass das Modell in diesem Modus einen erheblichen Speicherbedarf aufweisen kann. Das kann die Leistung des Rekonstruktionsalgorithmus mindern oder sogar dazu führen, dass kein Speicher mehr angefordert werden kann. Das kann insbesondere der Fall sein bei Kamerasetups mit besonders vielen Kameras und/oder großen Bildern. Dementsprechend ist es empfehlenswert, den Modus nur zur Verifizierung der Ergebnisse einzuschalten.

Werteliste: 0, *1*.

Standardwert: 0.

Parameter für die Oberflächenrekonstruktion unter Verwendung der Methoden 'surface_pairwise' und 'surface_fusion':

Je nach Auswahl in GenParamName können die folgenden Parameter der Oberflächenrekonstruktion (d.h. für Modelle des Typs *'surface_pairwise'* sowie *'surface_fusion'*) zusätzlich mit dem Parameter GenParamValue gesetzt werden:

'color': Legt fest, ob das rekonstruierte 3D-Objektmodell Farbinformationen aufweisen soll oder nicht (*'none'*). Für den Fall, dass Farbinformationen gewünscht werden, so muss zur Farbgebung eine der bereitgestellten Methoden (siehe Werteliste) ausgewählt werden. Siehe reconstruct_surface_stereo zum Zugriff auf die berechnete Farbinformation.

'median' Der Farbwert eines 3D-Punktes wird über den Median der Farbwerte aller Kameras gebildet, in denen der 3D-Punkt sichtbar ist.

'smallest_distance' Der Farbwert eines 3D-Punktes entspricht dem Farbwert der Kamera, die die kürzeste Distanz zu diesem Punkt aufweist.

'mean_weighted_distances' Die Kameras werden gemäß ihres Abstandes zum rekonstruierten 3D-Punkt gewichtet. Dabei erhalten Kameras mit einem kürzeren Abstand ein höheres Gewicht als solche, die weiter weg sind. Der Farbwert eines 3D-Punktes wird dann über Mittelung der gewichteten Farbwerte bestimmt.

'line_of_sight' Der Farbwert eines 3D-Punktes entspricht dem Farbwert der Kamera, die den kleinsten Winkel zwischen der Oberflächennormalen und dem Sichtstrahl aufweist.

'mean_weighted_lines_of_sight' Die Kameras werden gemäß ihres Winkels zwischen der Oberflächennormalen und dem Sichtstrahl gewichtet. Dabei erhalten Kameras mit einem kleineren Winkel ein höheres Gewicht. Der Farbwert eines 3D-Punktes wird dann über Mittelung der gewichteten Farbwerte bestimmt.

Werteliste: 'none', *'smallest_distance'*, *'mean_weighted_distances'*, *'line_of_sight'*, *'mean_weighted_lines_of_sight'*, *'median'*.

Standardwert: 'none'.

'color_invisible': Falls bei der Rekonstruktion ein Stereomodell vom Typ *'surface_fusion'* verwendet wurde, so werden Punkte rekonstruiert, die keine direkte Korrespondenz zu Punkten im Bild haben. Diese Punkte sind somit unsichtbar, da sie von keiner Kamera aus gesehen werden. Für diese Punkte wird Farbinformation anhand von Punkten in der Umgebung berechnet. Einfärben der unsichtbaren Punkte kann durch Setzen dieses Parameters auf *'false'* abgeschaltet werden. In diesem Fall werden die Farbwerte von unsichtbaren Punkten auf 255 gesetzt. Im Normalfall ist das Einfärben der unsichtbaren Punkte nicht sehr zeitaufwändig und kann aktiviert bleiben. Es kann jedoch passieren, dass der Wert für den Parameter *'resolution'* viel zu fein für die vorliegende Bildauflösung eingestellt wurde. In diesem Fall werden sehr viele unsichtbare 3D-Punkte rekonstruiert, was zu einer sehr zeitaufwändigen Nachbarschaftssuche führt. Um eine stark erhöhte Laufzeit zu vermeiden, sollte entweder der Parameter *'resolution'* angepasst werden oder die Bestimmung der Farbwerte für unsichtbare Punkte deaktiviert werden. Es ist zu beachten, dass das Setzen dieses Parameters keine Auswirkungen auf Stereomodelle des Typs *'surface_pairwise'* hat.

Werteliste: 'true',*'false'*.

Standardwert: 'true'.

'rectif_interpolation': Interpolationsmodus für die Bildpaarrektifizierung (siehe set_stereo_model_image_pairs). Es ist zu beachten, dass nach Änderung dieser Einstellung der Operator set_stereo_model_image_pairs erneut aufgerufen werden muss, um diese Einstellung wirksam werden zu lassen.

Werteliste: 'none', 'bilinear'.
Standardwert: 'bilinear'.

'rectif_sub_sampling': Abtastfaktor für die Bildpaarrektifizierung (siehe set_stereo_model_image_pairs). Es ist zu beachten, dass, nach Änderung dieser Einstellung der Operator set_stereo_model_image_pairs erneut aufgerufen werden muss, um diese Einstellung wirksam werden zu lassen.
Wertevorschläge: 0.5, 0.66, 1.0, 1.5, 2.0, 3.0, 4.0.
Standardwert: '1.0'.

'rectif_method': Rektifizierungsmethode für die Bildpaarrektifizierung (siehe set_stereo_model_image_pairs). Es ist zu beachten, dass nach Änderung dieser Einstellung der Operator set_stereo_model_image_pairs erneut aufgerufen werden muss, um diese Einstellung wirksam werden zu lassen.
Werteliste: 'viewing_direction', 'geometric'.
Standardwert: 'viewing_direction'.

'disparity_method': Methode zur Berechnung der Disparitätsbilder der Bildpaare (siehe reconstruct_surface_stereo). Zur Berechnung stehen die drei Methoden *'binocular'*, *'binocular_mg'* und *'binocular_ms'* zur Verfügung. Je nachdem welche der drei Methoden ausgewählt wurde, wird intern der entsprechende HALCON-Operator binocular_disparity, binocular_disparity_mg bzw. binocular_disparity_ms aufgerufen.
Werteliste: 'binocular', 'binocular_mg', 'binocular_ms'.
Standardwert: 'binocular'.

'min_disparity', *'max_disparity'*: Minimale und maximale Disparitätswerte, welche den Suchraum im Operator reconstruct_surface_stereo beschränken. Die Anzahl der minimalen und maximalen Disparitätswerte muss dabei mit der Anzahl der Bildpaare übereinstimmen. Falls *'min_disparity'* und *'max_disparity'* nicht über die Operatoren create_stereo_model oder set_stereo_model_param gesetzt wurden, werden die Disparitätswerte mit Hilfe des kleinsten umschließenden Quaders (siehe 'bounding_box') intern bestimmt. Es ist zu beachten, dass *'min_disparity'* und *'max_disparity'* tupelwertige Parameter sind und daher nicht zusammen mit anderen Parametern in einem Aufruf des Operators set_stereo_model_param gesetzt werden können (siehe Paragraph "Bemerkung über tupelwertige Modellparameter").

'binocular_score_thresh': Bei den Methoden 'binocular_mg' und 'binocular_ms' werden die Disparitäten deren Qualitätsmaß oberhalb der angegebenen Schwelle liegen von der Weiterverarbeitung ausgeschlossen und landen nicht in der rekonstruierten 3D-Punktwolke. Bei der Methode 'binocular' werden die Disparitäten unterhalb der angegebenen Schwelle ausgeschlossen.

Für Stereomodelle mit der Methode *'binocular'*: *Werteliste:* Positive und negative Integer- oder Float-Werte.
Standardwert: 0.5.

Für Stereomodelle mit der Methode *'binocular_mg'* **oder** *'binocular_ms'*: *Werteliste:* Integer- oder Float-Werte größer gleich 0.0.
Standardwert: -1.

Je nachdem, welche Methode zur Berechnung der Disparitätsbilder ausgewählt wurde, stehen dem Benutzer zusätzliche Parameter zur Verfügung, die ein genaueres Anpassen an den verwendeten Datensatz erlauben. Eine genauere Beschreibung der Parameter kann in der jeweiligen Operator-Referenz von binocular_disparity, binocular_disparity_mg oder binocular_disparity_ms nachgelesen werden.

Parameter für Stereomodelle mit Methode = *'binocular_mg'*

'binocular_method': Setzt die gewünschte Matchingmethode.
Werteliste: 'ncc', 'sad', 'ssd'.
Standardwert: 'ncc'.

'binocular_num_levels': Anzahl der verwendeten Bildpyramiden.
Werteliste: Integer-Wert größer gleich 1.
Standardwert: 1.

'binocular_mask_width': Breite des Korrelationsfensters.
Werteliste: Ungerade Integer-Werte größer gleich 3.
Standardwert: 11.

'binocular_mask_height': Höhe des Korrelationsfensters.
Werteliste: Ungerade Integer-Werte größer gleich 3.
Standardwert: 11.

'binocular_texture_thresh': Schwellenwert der Grauwertvarianz.
Werteliste: Integer- oder Float-Werte größer gleich 0.0.
Standardwert: 0.0.

'binocular_filter': Nachgelagerter Ergebnisfilter.
Werteliste: 'none', *'left_right_check'*.
Standardwert: 'none'.

'binocular_sub_disparity': Subpixel Interpolation der Disparitäten.
Werteliste: 'none', *'interpolation'*.
Standardwert: 'none'.

Parameter für Stereomodelle mit Methode = *'binocular_mg'*

'binocular_mg_gray_constancy': Gewicht der Grauwerterhaltung im Datenterm.
Werteliste: Integer- oder Float-Werte größer gleich 0.0.
Standardwert: 1.0.

'binocular_mg_gradient_constancy': Gewicht der Gradientenerhaltung im Datenterm.
Werteliste: Integer- oder Float-Werte größer gleich 0.0.
Standardwert: 30.0.

'binocular_mg_smoothness': Gewicht des Glattheitsterms bezüglich des Datenterms.
Werteliste: Integer- oder Float-Werte größer 0.0.
Standardwert: 5.0.

'binocular_mg_initial_guess': Startwert für die Disparität.
Werteliste: Integer- oder Float-Werte.
Standardwert: 0.0.

Die nachfolgenden Parameter steuern das Verhalten des verwendeten Mehrgitterverfahrens.

'binocular_mg_default_parameters': Setzt vordefinierte Parameterwerte für die Parameter des verwendeten Mehrgitterverfahren. *'binocular_mg_solver'*, *'binocular_mg_cycle_type'*, *'binocular_mg_pre_relax'*, *'binocular_mg_post_relax'*, *'binocular_mg_initial_level'*, *'binocular_mg_iterations'*, *'binocular_mg_pyramid_factor'*. Die genauen Werte der Parameter lassen sich der Operator-Referenz von `binocular_disparity_mg` entnehmen.
Werteliste: 'very_accurate', *'accurate'*, *'fast_accurate'*, *'fast'*.
Standardwert: 'fast_accurate'.

'binocular_mg_solver': Lösungsverfahren für die linearen Gleichungssysteme.
Werteliste: 'multigrid',*'full_multigrid'*, *'gauss_seidel'*.
Standardwert: 'full_multigrid'.

'binocular_mg_cycle_type': Wählt den Rekursionstyp des Mehrgitterverfahrens aus.
Werteliste: 'v',*'w'*, *'none'*.
Standardwert: 'v'.

'binocular_mg_pre_relax': Setzt für Mehrgitterverfahren die Anzahl der Gauß-Seidel-Iterationen beim Vorglätten, bzw. beim Gauß-Seidel-Verfahren die Anzahl an Iterationen beim Lösen des Gleichungssystems.
Werteliste: Integer- oder Float-Werte größer 0.0.
Standardwert: 1.

'binocular_mg_post_relax': Setzt für Mehrgitterverfahren die Anzahl der Gauß-Seidel-Iterationen beim Nachglätten.
Werteliste: Integer- oder Float-Werte.
Standardwert: 1.

'binocular_mg_initial_level': Legt die gröbste zu verwendende Stufe der Bildpyramide beim Grob-zu-Fein-Verfahren fest.
Werteliste: Integer-Werte.
Standardwert: -2.

'binocular_mg_iterations': Anzahl an Fixpunkt-Iterationen pro Ebene.
Werteliste: Integer- oder Float-Werte größer gleich 0.
Standardwert: 1.

'binocular_mg_pyramid_factor': Skalierungsfaktor für die Bildpyramide des Grob-zu-Fein-Verfahrens.
Werteliste: Integer- oder Float-Werte im Bereich von 0.1 und 0.9.
Standardwert: 0.6.

Parameter für Stereomodelle mit Methode = *'binocular_ms'*

'binocular_ms_surface_smoothing': Glättung von Oberflächen.
Werteliste: Integer-Werte größer gleich 0
Standardwert: 50

'binocular_ms_edge_smoothing': Glättung von Kanten.
Werteliste: Integer-Werte größer gleich 0.
Standardwert: 50.

'binocular_ms_consistency_check': Steuert das Verhalten, ob nur noch diejenigen Disparitätswerte zurückgegeben werden, die bei einer Suche eines Referenzmusters in beiden Bildern übereinstimmend gefunden werden.
Werteliste: 'true', *'false'*.
Standardwert: 'true'.

'binocular_ms_similarity_measure': Legt die Methode des Ähnlichkeitsmaßes fest.
Werteliste: 'census_dense', *'census_sparse'*.
Standardwert: 'census_dense'.

'binocular_ms_sub_disparity': Sollen kontinuierliche Werte aus den bisher ganzzahligen Disparitätswerten geschätzt werden?
Werteliste: 'true', *'false'*.
Standardwert: 'true'.

'point_meshing': Schaltet den Schritt zur Nachbearbeitung der Vermaschung der Oberflächenrekonstruktion ein oder aus. Wird ein Stereomodell vom Typ *'surface_pairwise'* verwendet, wird ein Poisson-basierter Vermaschungsalgorithmus unterstützt. Für Stereomodelle vom Typ *'surface_fusion'* wird nur eine Vermaschung der Isofläche angeboten. (siehe `reconstruct_surface_stereo` für weitere Informationen).

`Werteliste`: *'none'* (`Standardwert`), *'poisson'*, *'isosurface'*

Wenn die Poisson-basierte Vermaschung eingeschaltet ist, können die folgenden Parameter eingestellt werden:

- *'poisson_depth'*: Tiefe des Poisson-Octrees. Ein tieferer Octree führt zu höherer Vermaschungauflösung, aber auch zu erheblich höherem Zeit- und Speicherbedarf.
  Wertevorschläge: 6, *8*, *10*.
  Standardwert: 8.
  Restriktion: 3 <= 'poisson_depth' <= 12
- *'poisson_solver_divide'*: Tiefe des eingesetzten Block-Gauss-Seidel-Algorithmus, der zur Lösung des Poissongleichungen benutzt wird. Bei kleinem zusätzlichen Zeitbedarf schafft dieser Algorithmus erhebliche Speicherersparnisse. Die optimalen Werte für diesen Parameter sind um 0 bis 2 kleiner als die Tiefe des Octrees.
  Wertevorschläge 6, *8*, *10*.
  Standardwert: 8.
  Restriktion: 3 <= 'poisson_solver_divide' <= 'poisson_depth'
- *'poisson_samples_per_node'*: Mindestanzahl von Punkten, welche in einem Octree-Blatt angesammelt werden. Dieser Parameter wird benutzt, um verrauschte Daten zu behandeln, d.h. rauschfreie Daten können auf mehrere Blätter verteilt werden, während verrauschte Daten eher auf weniger Blätter verteilt werden sollten, um das Rauschen kompensieren zu können. Ein Seiteneffekt ist, dass größere Werte dieses Parameters allgemein zu kleineren Octrees führen, was wiederum die Vermaschung beschleunigt, aber auch eine kleinere Rekonstruktionsauflösung nach sich zieht.
  Wertevorschläge: 1, *5*, *10*, *30*, *40*.
  Standardwert: 30.

Alle Parameter bis auf *'binocular_mg_default_parameters'* können im Nachhinein mit dem Operator `get_stereo_model_param` abgefragt werden.

Parameter für ein Stereomodell der Methode 'surface_pairwise':

Unter Verwendung eines Stereomodells des Typs *'surface_pairwise'* kann durch das Setzen von `GenParamName` auf den folgenden Wert, die Oberflächenrekonstruktion gesteuert werden:

'sub_sampling_step': Abtastschritt für die X-, Y- und Z-Bilddaten, welche als Ergebnis der paarweisen Disparitätsberechnung entstehen, bevor diese Daten in die Oberflächenrekonstruktion einfließen (siehe `reconstruct_surface_stereo`).

Wertevorschläge: 1, *2*, *3*.

Standardwert: 2.

Parameter für ein Stereomodell der Methode 'surface_fusion':

Unter Verwendung eines Stereomodells des Typs *'surface_fusion'* kann durch das Setzen von `GenParamName` auf einen der folgenden Werte, die Oberflächenrekonstruktion gesteuert werden:

'resolution': Legt die Distanz zwischen benachbarten Abtastpunkten in jede Koordinatenrichtung des diskretisierten umschließenden Quaders fest. *'resolution'* wird in [m] gesetzt. Siehe `reconstruct_surface_stereo` für weitere Details.

Zu kleine Werte erhöhen die Laufzeit der Rekonstruktion unnötigerweise. Zu große Werte haben eine Rekonstruktion mit fehlenden Details zur Folge. Standardmäßig wird die Auflösung in Abhängigkeit des umschließenden Quaders auf einen recht groben Wert gesetzt. Die Auflösung wird bei einem Umsetzen des umschließenden Quaders zurückgesetzt.

Eventuell muss *'smoothing'* angepasst werden, wenn *'resolution'* verändert wird. Der Wert für den Parameter *'surface_tolerance'* sollte in jedem Fall etwas größer als der Wert für *'resolution'* sein, um Diskretisierungseffekte zu vermeiden.

Wertevorschläge: 0.001, *0.01*.

'surface_tolerance': Legt fest, wie viel Rauschen um die Eingangspunktwolke zu einer Oberfläche zusammengefügt werden soll. Punkte im Sichtkegel einer Kamera werden als sicher außerhalb des Objekts (vor der Oberfläche) oder sicher innerhalb des Objekts (hinter der Oberfläche) angesehen bezüglich dieser Kamera, wenn ihr Abstand zur ursprünglichen Oberfläche *'surface_tolerance'* übersteigt. *'surface_tolerance'* wird in [m] gesetzt. Siehe `reconstruct_surface_stereo` für weitere Details und eine Abbildung.

Zu kleine Werte führen zu einer unebenen Oberfläche, während hingegen zu große Werte eigentlich getrennte Oberflächen zu einer verschmelzen lassen. Standardmäßig ist der Wert für *'surface_tolerance'* auf das dreifache der Auflösung gesetzt. Ein Umsetzen des umschließenden Quaders hat das Zurücksetzen des Parameters zur Folge.

Weiterhin sollte der Wert für *'surface_tolerance'* immer etwas größer als der Wert für die Auflösung sein, um Diskretisierungseffekte zu vermeiden. Zudem darf der Wert für *'min_thickness'* nie kleiner werden als der Wert für *'surface_tolerance'*. Wird *'min_thickness'* zu niedrig gesetzt, so wird *'surface_tolerance'* automatisch auf denselben Wert wie *'min_thickness'* gesetzt. Wird *'surface_tolerance'* zu hoch gesetzt, so wird ein Fehler zurückgegeben.

Wertevorschläge: 0.003, *0.03*.

Restriktion: 'surface_tolerance' < *'min_thickness'*

'min_thickness': Bestimmt die Länge des zu berücksichtigenden Sichtkegels einer Kamera hinter der mit der paarweisen Rekonstruktion erstellten Oberfläche. Punkte hinter der Oberfläche (von der gegebenen Kamera aus gesehen) werden nur als sicher innerhalb des Objekts angesehen, wenn ihr Abstand zur ursprünglichen Oberfläche *'min_thickness'* nicht übersteigt. *'min_thickness'* wird in [m] gesetzt. Siehe `reconstruct_surface_stereo` für weitere Details und eine Abbildung.

Falls davon ausgegangen werden kann, dass die Sichtstrahlen das zu rekonstruierende geschlossene Objekt nur ein einziges Mal durchschneiden (z.B. falls alle Kameras das Objekt nur von einer Seite aus betrachten), sollte der Parameter auf seinem Standardwert belassen werden. Falls davon ausgegangen werden kann, dass die Sichtstrahlen das zu rekonstruierende Objekt mehr als einmal durchschneiden (z.B. wenn die Kameras das Objekt von verschiedenen Seiten aus betrachten), so sollte nur das Innere des Objektes als hinter der Oberfläche liegend markiert werden. Der Parameter sollte dann zunächst auf einen Wert gesetzt werden, der kleiner ist als die Dicke des zu rekonstruierenden Objekts.

Die Methode *'surface_fusion'* versucht stets, geschlossene Oberflächen zu erzeugen. Werden mehrere verschiedene Objekte nur von einer Seite aus betrachtet, kann es sinnvoll sein, *'min_thickness'* zu verkleinern, um die Tiefe der rekonstruierten Objekte zu beschränken und damit zu verhindern, dass sie zu einer einzigen Oberfläche verschmelzen. Da in diesem Fall die Rückseite der Objekte nicht betrachtet wird, ist ihre Rekonstruktion wahrscheinlich nicht korrekt.

Wird der Parameters auf zu kleine Werte gesetzt, so weist die Punktwolke entweder Löcher oder doppelte Wände auf. Stellt man hingegen zu große Werte ein, so äußert sich das in einer verzerrten Punktwolke oder

einem Aufblasen der Oberfläche in Richtung der Außenseite des Objekts. Wird die Oberfläche so stark aufgeblasen, dass sie außerhalb des umschließenden Quaders liegt, so werden keine Punkte rekonstruiert. Der Wert des Parameters wird bei einem Umsetzen des umschließenden Quaders ebenfalls zurückgesetzt.

Der Wert für *'min_thickness'* darf nie kleiner werden als der Wert für *'surface_tolerance'*. Wird *'min_thickness'* zu niedrig gesetzt, so wird *'surface_tolerance'* automatisch auf denselben Wert wie *'min_thickness'* gesetzt. Wird *'surface_tolerance'* zu hoch gesetzt, so wird ein Fehler zurückgegeben.

Wertevorschläge: 0.005, *0.05*

'smoothing': Legt fest, wie wichtig eine kleine Totalvariation der Distanzfunktion gegenüber der Datentreue zur Eingabepunktwolke für den Algorithmus ist. Damit kann also die 'Unruhe' der rekonstruierten Oberfläche durch den Glättungsparameter *'smoothing'* gesteuert werden (siehe `reconstruct_surface_stereo` für weitere Details).

Der gewünschte Wert des Parameters für einen bestimmten Datensatz muss durch Ausprobieren gefunden werden. Zu kleine Werte für diesen Parameter führen dazu, dass die Oberfläche viele Ausreißer enthält. Dies ist sogar dann der Fall, wenn die Oberfläche deswegen große Sprünge aufweist. Zu große Werte hingegen führen dazu, dass der Algorithmus sich nicht mehr auf die Eingangspunktwolke, welche durch die paarweise Rekonstruktion erstellt wurde stützt. (Wie der Algorithmus die Distanzen zur Eingangspunktwolke behandelt hängt dabei stark von *'surface_tolerance'* und *'min_thickness'*) ab.

Der Wert des Parameters wird bei einem Umsetzen des umschließenden Quaders ebenfalls zurückgesetzt. Eventuell muss *'smoothing'* angepasst werden, wenn *'resolution'* verändert wird.

Wertevorschläge: 15.0, *1.0*, *0.1*.

Standardwert: 1.0.

Bemerkung über tupelwertige Modellparameter

Die meisten der Stereomodellparameter haben nur einen Wert. In diesem Fall kann man eine Liste (d.h. ein Tupel) von Parameternamen und eine Liste (Tupel) derselben Länge von Parameterwerten übergeben. Tupelwertige Parameter dagegen brauchen eine Liste von Werten. Wenn so ein Parameter mit einwertigen Parametern gemeinsam gesetzt wird, kann nicht mehr unterschieden werden, welcher Wert zu welchem Parameternamen gehört. Deswegen können tupelwertige Parameter wie *'bounding_box'*, *'min_disparity'* oder *'max_disparity'* immer nur einzeln mit `set_stereo_model_param` gesetzt werden.

Parameter

▷ **`StereoModelID`** (input_control) . . . . . stereo_model ⇝ *handle*
Handle des Stereomodells.

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der Modellparameter.
Werteliste: `GenParamName` ∈ {'bounding_box', 'persistence', 'sub_sampling_step', 'rectif_interpolation', 'rectif_sub_sampling', 'rectif_method', 'disparity_method', 'binocular_method', 'binocular_num_levels', 'binocular_mask_width', 'binocular_mask_height', 'binocular_texture_thresh', 'binocular_score_thresh', 'binocular_filter', 'binocular_sub_disparity', 'binocular_mg_gray_constancy', 'binocular_mg_gradient_constancy', 'binocular_mg_smoothness', 'binocular_mg_initial_guess', 'binocular_mg_default_parameters', 'binocular_mg_solver', 'binocular_mg_cycle_type', 'binocular_mg_pre_relax', 'binocular_mg_post_relax', 'binocular_mg_initial_level', 'binocular_mg_iterations', 'binocular_mg_pyramid_factor', 'binocular_ms_surface_smoothing', 'binocular_ms_edge_smoothing', 'binocular_ms_consistency_check', 'binocular_ms_similarity_measure', 'binocular_ms_sub_disparity', 'point_meshing', 'poisson_depth', 'poisson_solver_divide', 'poisson_samples_per_node', 'resolution', 'surface_tolerance', 'min_thickness', 'smoothing', 'color', 'color_invisible', 'min_disparity', 'max_disparity'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Werte der Modellparameter.
Wertevorschläge: `GenParamValue` ∈ {1, -2, -5, 0, 0.3, 0.5, 0.9, 1, 2, 3, 'census_dense', 'census_sparse', 'binocular', 'ncc', 'none', 'sad', 'ssd', 'bilinear', 'false', 'viewing_direction', 'geometric', 'very_accurate', 'accurate', 'fast_accurate', 'fast', 'v', 'w', 'none', 'gauss_seidel', 'multigrid', 'true', 'poisson', 'isosurface', 'interpolation', 'left_right_check', 'full_multigrid', 'binocular_mg', 'binocular_ms', 'smallest_distance', 'mean_weighted_distances', 'line_of_sight', 'mean_weighted_lines_of_sight', 'median'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- StereoModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_stereo_model`

Nachfolger

`reconstruct_surface_stereo`, `reconstruct_points_stereo`

Siehe auch

`get_stereo_model_param`, `set_stereo_model_image_pairs`

Modul

3D Metrology

5.4 Photometrisches Stereo

`estimate_al_am` (Image : : : Albedo, Ambient)

Berechnen des Reflexionskoeffizienten und der Stärke der indirekten Beleuchtung der Oberfläche.

Der Operator `estimate_al_am` berechnet aus dem Bild `Image` durch Maximum- und Minimumberechnung die Größe des Reflexionskoeffizienten, genannt `Albedo`, und die Stärke der indirekten Beleuchtung der Oberfläche, genannt `Ambient`.

Achtung

Es wird angenommen, dass im Bild mindestens ein Punkt enthalten ist, für die die Reflexionsfunktion ihr Minimum annimmt. Das sind z.B. Punkte, die in selbstbeschatteten Gebieten der Oberfläche liegen. Außerdem wird angenommen, dass ein Punkt im Bild vorkommt, für den die Reflexionsfunktion ihr Maximum annimmt. Ist dies nicht der Fall, wird ein falscher Wert geschätzt.

Parameter

▷ **`Image`** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
Das Bild, für das der Reflexionskoeffizient und die Hintergrundbeleuchtung geschätzt werden soll.

▷ **`Albedo`** (output_control) . . . . . real(-array) ⇝ *real*
Größe des Reflexionskoeffizienten der Oberfläche.

▷ **`Ambient`** (output_control) . . . . . real(-array) ⇝ *real*
Stärke der indirekten Beleuchtung der Oberfläche.

Ergebnis

`estimate_al_am` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `photometric_stereo`, `shade_height_field`

Modul

3D Metrology

estimate_sl_al_lr (Image : : : Slant, Albedo)

Berechnen des Slants der Lichtquelle und des Reflexionskoeffizienten der Oberfläche.

Der Operator estimate_sl_al_lr berechnet nach dem Algorithmus von Lee und Rosenfeld aus dem Bild Image den Slant der Lichtquelle, d.h. den Winkel, den die Richtung der Lichtquelle mit der positiven z-Achse bildet, und den Reflexionskoeffizienten Albedo der im Bild dargestellten Oberfläche, d.h. das Verhältnis von einfallender Strahlung zu austretender Strahlung.

Achtung

Der Reflexionskoeffizient Albedo wird für die gesamte im Bild dargestellte Oberfläche als konstant angenommen.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
  Das Bild, für das der Slant und der Reflexionskoeffizient geschätzt werden soll.
- ▷ **Slant** (output_control) . . . . . angle.deg(-array) ⇝ *real*
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
- ▷ **Albedo** (output_control) . . . . . real(-array) ⇝ *real*
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.

Ergebnis

estimate_sl_al_lr liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

sfs_mod_lr, sfs_orig_lr, sfs_pentland, photometric_stereo, shade_height_field

Modul

3D Metrology

estimate_sl_al_zc (Image : : : Slant, Albedo)

Berechnen des Slants der Lichtquelle und des Reflexionskoeffizienten der Oberfläche.

Der Operator estimate_sl_al_zc berechnet nach dem Algorithmus von Zheng und Chellappa aus dem Bild Image den Slant der Lichtquelle, d.h. den Winkel, den die Richtung der Lichtquelle mit der positiven z-Achse bildet, und den Reflexionskoeffizienten Albedo der im Bild dargestellten Oberfläche, d.h. das Verhältnis von einfallender Strahlung zu austretender Strahlung.

Achtung

Der Reflexionskoeffizient Albedo wird für die gesamte im Bild dargestellte Oberfläche als konstant angenommen.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
  Das Bild, für das der Slant und der Reflexionskoeffizient geschätzt werden soll.
- ▷ **Slant** (output_control) . . . . . angle.deg(-array) ⇝ *real*
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
- ▷ **Albedo** (output_control) . . . . . real(-array) ⇝ *real*
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.

Ergebnis

estimate_sl_al_zc liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `photometric_stereo`, `shade_height_field`

Modul

3D Metrology

estimate_tilt_lr (Image : : : Tilt)

Berechnen des Tilts der Lichtquelle.

Der Operator `estimate_tilt_lr` berechnet nach dem Algorithmus von Lee und Rosenfeld aus dem Bild `Image` den `Tilt` der Lichtquelle, d.h. den Winkel, den die Richtung der Lichtquelle nach Projektion in die xy-Ebene mit der x-Achse bildet.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
Das Bild, für das der Tilt geschätzt werden soll.

▷ **Tilt** (output_control) . . . . . angle.deg(-array) ⇝ *real*
Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).

Ergebnis

`estimate_tilt_lr` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `photometric_stereo`, `shade_height_field`

Modul

3D Metrology

estimate_tilt_zc (Image : : : Tilt)

Berechnen des Tilts der Lichtquelle.

Der Operator `estimate_tilt_zc` berechnet nach dem Algorithmus von Zheng und Chellappa aus dem Bild `Image` den `Tilt` der Lichtquelle, d.h. den Winkel, den die Richtung der Lichtquelle nach Projektion in die xy-Ebene mit der x-Achse bildet.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
Das Bild, für das der Tilt geschätzt werden soll.

▷ **Tilt** (output_control) . . . . . angle.deg(-array) ⇝ *real*
Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).

Ergebnis

`estimate_tilt_zc` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `photometric_stereo`, `shade_height_field`

Modul

3D Metrology

photometric_stereo (Images : HeightField, Gradient, Albedo : Slants, Tilts, ResultType, ReconstructionMethod, GenParamName, GenParamValue :)

Rekonstruieren einer Oberfläche mittels photometrischer Stereomessung.

`photometric_stereo` kann dazu genutzt werden, die dreidimensionale Form eines Objekts von seiner zweidimensionalen Textur zu trennen, z.B. von seinem Druckbild. Der Operator benötigt mindestens drei Bilder des gleichen Objekts, aufgenommen mit unterschiedlichen bekannten Beleuchtungsrichtungen. Die Ausrichtung der Kamera zum Objekt muss für alle Bilder gleich sein.

Die dreidimensionale Form des Objekts wird in erster Linie in Form von lokalen Gradienten der dreidimensionalen Oberfläche berechnet. Diese Gradienten können weiter integriert werden, so dass ein Höhenmodell entsteht, d.h., ein Bild, in dem jeder Pixelwert einer relativen Höhe entspricht. Die zweidimensionale Textur wird als Albedo bezeichnet und entspricht der lokalen Lichtabsorption und Reflexionscharakteristik der Oberfläche ohne Einfluss von Schatten.

Typische Anwendungen des photometrischen Stereoverfahrens

Typische Anwendungen des photometrischen Stereoverfahrens sind das Erkennen kleiner Inkonsistenzen in einer Oberfläche, z.B. Defekten, oder das Ausschließen des Einflusses der Beleuchtungsrichtung auf Bilder, die z.B. für die Druckbildinspektion von nicht flachen Zeichen genutzt werden. Photometrisches Stereo ist nicht geeignet für die Rekonstruktion absoluter Höhen, d.h., es ist keine Alternative zu typischen 3D Rekonstruktionsansätzen wie Depth from Focus oder das Lichtschnittverfahren.

Einschränkungen des photometrischen Stereoverfahrens

`photometric_stereo` basiert auf dem Algorithmus von Woodham und setzt daher zum einen eine orthoskopische Projektion der Kamera voraus, d.h., es muss eine telezentrische Linse oder eine Linse mit einer großen Brennweite benutzt werden. Zum anderen wird vorausgesetzt, dass jede der Lichtquellen parallele und gleichförmige Lichtstrahlen produziert, d.h., es müssen telezentrische Lichtquellen mit gleichförmiger Intensität oder, als Alternative, sehr weit entfernte Lichtquellen benutzt werden. Zusätzlich muss das Objekt Lambertsche Oberflächeneigenschaften aufweisen, d.h., es muss hereinkommendes Licht diffus reflektieren. Objekte oder Regionen eines Objekts, die spiegelnde Reflexionseigenschaften aufweisen (spiegelnde oder glänzende Oberflächen) können nicht korrekt verarbeitet werden und führen daher zu fehlerhaften Ergebnissen.

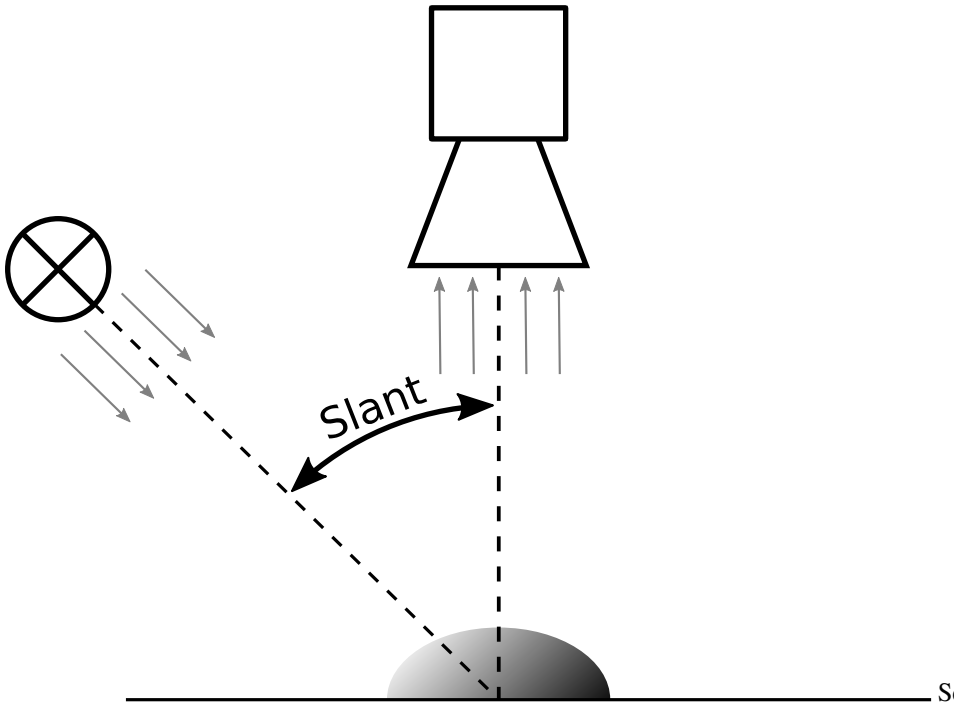
Der Aufbau

Die Kamera mit der telezentrischen Linse wird orthogonal, d.h., rechtwinklig, zu der zu konstruierenden Szene angebracht. Die Orientierung der Kamera in Bezug auf die Szene darf sich während der Bildaufnahme nicht verändern. Im Gegensatz dazu muss sich die Orientierung der Lichtquelle in Bezug auf die Kamera in mindestens drei Bildern ändern.

Spezifizieren der Beleuchtungsrichtungen

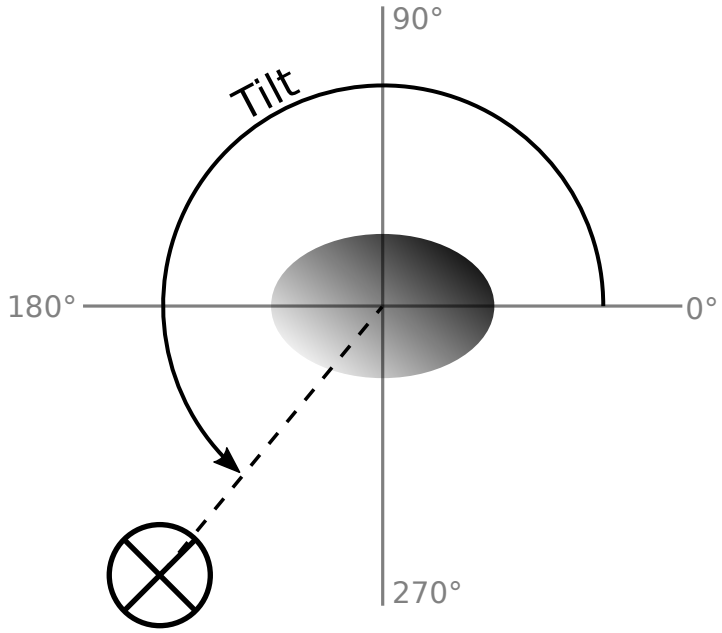
Für jedes Bild werden die Beleuchtungsrichtungen in Form von Winkeln innerhalb der Parameter `Slants` und `Tilts` angegeben. Diese beschreiben die Richtung der Beleuchtung in Bezug auf die Szene. Zum besseren Verständnis dieser Parameter ist zu bedenken, dass der Ansatz davon ausgeht, dass die Lichtquelle parallele Lichtstrahlen erzeugt, die Kamera eine telezentrische Linse hat und die Kamera orthogonal zu der zu konstruierenden Szene angeordnet ist:

Slants Der Neigungswinkel (`Slants`) ist der Winkel zwischen der optischen Achse der Kamera und der Beleuchtungsrichtung.



Seitenansicht

Tilts Der Winkel in `Tilts` wird innerhalb der Objektebene oder einer beliebigen parallelen Ebene, z.B. der Bildebene, gemessen. Insbesondere beschreibt er den Winkel zwischen der Richtung, die von der Mitte des Bildes nach rechts zeigt, und der in das Bild projizierten Beleuchtungsrichtung. D.h., beim Blick auf das Bild (oder die korrespondierende Szene), entspricht eine Beleuchtung von rechts einem Winkel von 0°, eine Beleuchtung von oben einem Winkel von 90°, eine Beleuchtung von links einem Winkel von 180°, usw.



Draufsicht

Wie oben erwähnt benötigt das photometrische Stereoverfahren mindestens drei Bilder mit verschiedenen Beleuchtungsrichtungen. Allerdings führt die dreidimensionale Geometrie der Objekte typischerweise zu einem Schattenwurf. In den Schattenregionen ist die Anzahl der effektiv verfügbaren Beleuchtungsrichtungen reduziert, so dass Mehrdeutigkeiten entstehen. Um trotzdem ein robustes Ergebnis zu erhalten, ist Redundanz nötig. Deshalb sollten normalerweise mehr als drei Lichtquellen mit unterschiedlichen Beleuchtungsrichtungen gewählt werden. Eine steigende Anzahl von Beleuchtungsrichtungen führt aber auch zu einer höheren Anzahl von zu verarbeitenden Bildern und somit zu einer höheren Verarbeitungszeit. In den meisten Anwendungen ist eine Anzahl von vier bis sechs Beleuchtungsrichtungen sinnvoll. Als Faustregel sollten die Winkel in `Slants` zwischen 30° und 60° betragen. Die Winkel in `Tilts` sollten gleichmäßig rund um das zu vermessende Objekt angeordnet sein. Es ist zu beachten, dass die Beleuchtungsrichtungen so gewählt werden müssen, dass sie nicht in der gleichen Ebene liegen, d.h., sie müssen unabhängig sein. Ist dies nicht der Fall, wird eine Exception-Behandlung durchgeführt.

Eingabebilder und Domänen

Die Eingabebilder müssen als Bildarray in dem Parameter `Images` übergeben werden. Jedes Bild muss dabei, wie bereits erwähnt, mit unterschiedlichen Beleuchtungsrichtungen aufgenommen worden sein. Liegen die Bilder in einem Mehrkanalbild vor, können sie mit Hilfe von `image_to_channels` in einen Bildarray konvertiert werden. Alternativ kann ein Bildarray mittels `concat_obj` erzeugt werden.

`photometric_stereo` stützt sich auf die Auswertung "photometrischer Information", d.h., den Grauwerten, die in den Bildern gespeichert sind. Diese Information sollte möglichst unverzerrt und genau sein. Es sollte daher sicher gestellt sein, dass die für die Bildaufnahme verwendete Kamera eine lineare Charakteristik aufweist. Mit dem Operator `radiometric_self_calibration` kann die Charakteristik der Kamera bestimmt werden und mit `lut_trans` können die Grauwerte im Falle einer nicht-linearen Charakteristik korrigiert werden. Wenn genaue Messungen benötigt werden, sollte außerdem der volle Dynamikumfang der Kamera genutzt werden, da hierdurch genauere Grauwertinformationen erzielt werden. Aus dem selben Grund führen Bilder mit einer Bit-Tiefe von mehr als 8 (z.B. uint2 Bilder statt Byte Bilder) zu einer besseren Genauigkeit.

Die gewählte Domäne der Eingabebilder legt fest, welcher Algorithmus intern zur Verarbeitung der Bilder benutzt wird. Es stehen drei verschiedene Algorithmen zur Verfügung:

- Hat jedes Bild die volle Domäne, wird der schnellste Algorithmus verwendet. Dieser Fall ist der Standardfall und sollte für die meisten Anwendungen verwendet werden.
- Teilen sich die Bilder die gleiche aber reduzierte Domäne, werden nur die Pixel der entsprechenden Domäne verarbeitet. Dieser Fall dient dazu, Bereiche des Objekts für alle Bilder auszuschließen. Typischerweise werden Bereiche ausgeschlossen, die keine Lambertschen Reflexionseigenschaften aufweisen oder die nicht von weiterem Interesse sind, z.B. dunkle Löcher in der Oberfläche.
- Haben die Bilder unterschiedliche Domänen, werden nur die Grauwerte der entsprechenden Domänen verwendet. Dabei ist zu beachten, dass nur diejenigen Pixel verarbeitet werden, die in mindesten drei Bildern unabhängige Beleuchtungsrichtungen aufweisen. Dieser Fall dient dazu, Bereiche für einzelne Bilder auszuschließen. Dies können z.B. Bereiche des Objekts sein, die verfälschte photometrische Informationen enthalten, z.B. Schatten. Das Ausschließen dieser Regionen führt zu einer höheren Genauigkeit. Allerdings ist der hier benutzte Algorithmus signifikant langsamer als die Algorithmen, die bei voller Domäne oder einer identischen Domäne für alle Bilder verwendet werden.

Die Ausgabebilder

Der Operator kann Bilder für die rekonstruierten Gradienten (`Gradient`), den Albedo (`Albedo`) und das Höhenmodell (`HeightField`) der Oberfläche zurückgeben:

- Das `Gradient`-Bild ist ein Vektorfeld, das die partiellen Ableitungen der Oberfläche beinhaltet. Es kann z.B. als Eingabe für `reconstruct_height_field_from_gradient` verwendet werden. Für eine Visualisierung kann man statt der Oberflächengradienten die normalisierten Oberflächengradienten berechnen, indem man `ResultType` auf *'normalized_gradient'* setzt. Hierbei entsprechen die Zeilen- und Spalten- Komponenten den Zeilen- und Spalten-Komponenten der normalisierten Oberflächengradienten. Falls `ResultType` auf *'all'* gesetzt ist, wird die Standardeinstellung von `Gradient` verwendet und nicht *'normalized_gradient'*.
- Das `Albedo`-Bild beschreibt die Relation des reflektierten Lichts zum einfallenden Licht und hat einen Wertebereich zwischen eins (weisse Oberfläche) und null (schwarze Oberfläche). Somit beschreibt das Albedo-Bild eine Charakteristik der Oberfläche. Zum Beispiel bei einer bedruckten Oberfläche entspricht es dem Druckbild mit Ausschluss des Einflusses durch einfallendes Licht (Schatten).
- Das Bild für das Höhenmodell `HeightField` ist ein Bild, in dem die Pixelwerte einer relativen Höhe entsprechen.

Im Standardfall werden alle diese ikonischen Objekte zurückgeliefert, d.h., der Parameter `ResultType` ist auf *'all'* gesetzt. Falls nur bestimmte Ausgabebilder benötigt werden, kann `ResultType` auf ein Tupel gesetzt werden, das festlegt, welche der Ausgabebilder benötigt werden. Zur Auswahl stehen die Werte *'gradient'*, *'albedo'* und *'height_field'*. In bestimmten Anwendungen, z.B. zur Oberflächeninspektion, werden nur die Bilder für `Gradient` und `Albedo` benötigt. Dann kann das Ausschließen der Oberflächenrekonstruktion die Verarbeitungszeit deutlich beschleunigen, d.h., hier würden für `ResultType` nur die Werte *'gradient'* und *'albedo'*, aber nicht *'height_field'*, übergeben werden.

Intern ermittelt `photometric_stereo` zunächst die Gradienten und, falls benötigt, integriert diese Werte dann, um ein Höhenmodell zu erhalten. Diese Integration wird mit den gleichen Algorithmen ausgeführt, die für den Operator `reconstruct_height_field_from_gradient` zur Verfügung stehen

und mit den Parametern ReconstructionMethod, GenParamName und GenParamValue kontrolliert werden. Für weitere Informationen zu den Parametern verweisen wir auf die Beschreibung des Operators reconstruct_height_field_from_gradient. Ist ResultType so gesetzt, dass kein Höhenmodell zurückgegeben wird, werden diese Parameter ignoriert.

Achtung

photometric_stereo setzt quadratische Pixel voraus. Außerdem wird vorausgesetzt, dass die Höhen auf einem Gitter der Schrittweite 1 im Objektraum vorliegen. Ist das nicht der Fall, d.h., wenn die Pixelgröße der Kamera projiziert in den Objektraum von 1 abweicht, müssen die zurückgegebenen Höhen mit der tatsächlichen Gitterschrittweite (Wert der Pixelgröße projiziert in den Objektraum) multipliziert werden. Die Größe des Pixels im Objektraum wird berechnet, indem die Größe des Pixels in der Kamera durch die Vergrößerung der (telezentrischen) Linse geteilt wird.

Parameter

- ▷ **Images** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2
  Array mit mindestens drei Bildern mit unterschiedlicher Beleuchtungsrichtung.
- ▷ **HeightField** (output_object) ..... image ⇝ *object :* real
  Rekonstruiertes Höhenmodell.
- ▷ **Gradient** (output_object) ..... image ⇝ *object :* vector_field
  Gradientenfeld der Oberfläche.
- ▷ **Albedo** (output_object) ..... image ⇝ *object :* real
  Albedo der Oberfläche.
- ▷ **Slants** (input_control) ..... angle.deg-array ⇝ *real* / integer
  Winkel zwischen der Kamera und der Richtung der Lichtquelle (im Gradmaß).
  Defaultwert: 45.0
  Wertevorschläge: Slants ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich: 0.0 ≤ Slants ≤ 180.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 10.0
- ▷ **Tilts** (input_control) ..... angle.deg-array ⇝ *real* / integer
  Winkel der Richtung der Lichtquelle innerhalb der Objektebene (im Gradmaß).
  Defaultwert: 45.0
  Wertevorschläge: Tilts ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich: 0.0 ≤ Tilts ≤ 360.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 10.0
- ▷ **ResultType** (input_control) ..... string-array ⇝ *string*
  Zu berechnende ikonische Ergebnisse.
  Defaultwert: 'all'
  Werteliste: ResultType ∈ {[], 'all', 'height_field', 'gradient', 'normalized_gradient', 'albedo'}
- ▷ **ReconstructionMethod** (input_control) ..... string ⇝ *string*
  Typ der Rekonstruktionsmethode.
  Defaultwert: 'poisson'
  Werteliste: ReconstructionMethod ∈ {'fft_cyclic', 'rft_cyclic', 'poisson'}
- ▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'optimize_speed', 'caching'}
- ▷ **GenParamValue** (input_control) ..... integer-array ⇝ *integer* / real / string
  Werte der generischen Parameter.
  Defaultwert: []
  Werteliste: GenParamValue ∈ {'standard', 'patient', 'exhaustive', 'use_cache', 'no_cache', 'free_cache'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert photometric_stereo den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

optimize_fft_speed

Modul

3D Metrology

reconstruct_height_field_from_gradient (
Gradient : HeightField : ReconstructionMethod, GenParamName,
GenParamValue :)

Rekonstruieren einer Oberfläche aus Oberflächengradienten.

reconstruct_height_field_from_gradient rekonstruiert eine Oberfläche aus den Gradienten, die in Gradient übergeben werden. Die Oberfläche wird als Höhenmodell zurückgegeben, d.h., die Grauwerte jedes einzelnen Bildpunktes entsprechen der relativen Höhe der Oberfläche.

Die Rekonstruktion erfolgt durch die Integration der Gradienten durch verschiedene Algorithmen, die im Parameter ReconstructionMethod ausgewählt werden können. Da Gradientenfelder typischerweise nicht integrierbar sind, z.B. aufgrund von Rauschen, liefern die verschiedenen zur Verfügung stehenden Algorithmen eine Lösung im Sinne der kleinsten Quadrate. Die verfügbaren Algorithmen unterscheiden sich in der Art, wie sie die Randbedingungen modellieren. Zur Zeit werden drei verschiedene Algorithmen unterstützt: *'fft_cyclic'*, *'rft_cyclic'* und *'poisson'*.

Rekonstruktion mittels schneller Fourier Transformation

Die Verfahren *'fft_cyclic'* und *'rft_cyclic'* nehmen an, dass die Bildfunktion an den Rändern zyklisch ist. Dies kann an den Bildrändern zu Artefakten führen, weswegen in den meisten Fällen die Rekonstruktion nach Poisson vorzuziehen ist.

Der Unterschied zwischen *'fft_cyclic'* und *'rft_cyclic'* ist, dass die rft Version eine schnellere Verarbeitungszeit hat und weniger Speicher benötigt als die fft Version. Dafür ist die fft Version, obwohl fft und rft eigentlich das gleiche Ergebnis liefern sollten, numerisch geringfügig genauer. Da reconstruct_height_field_from_gradient intern eine schnelle Fourier Transformation benutzt, kann die Laufzeit des Operators durch einen vorherigen Aufruf von optimize_fft_speed oder optimize_rft_speed beeinflusst werden.

Rekonstruktion nach Poisson

Der *'poisson'*-Algorithmus nimmt an, dass das Bild konstante Gradienten am Bildrand hat. In den meisten Fällen bietet er die geeignetere Rekonstruktionsmethode für reconstruct_height_field_from_gradient. Seine Laufzeit kann nur durch Setzen von GenParamName auf *'optimize_speed'* und GenParamValue auf *'standard'*, *'patient'* oder *'exhaustive'* beeinflusst werden. Für die Bedeutung dieser Parameter sei auf die Beschreibung von optimize_fft_speed verwiesen.

Der *'poisson'*-Algorithmus verwendet standardmäßig einen Cache, der von der Größe des Bildes abhängt und die Rekonstruktion deutlich beschleunigt, vorausgesetzt, alle zu verarbeitenden Bilder haben die gleiche Größe. Der Cache wird beim ersten Aufruf des *'poisson'*-Algorithmus aufgebaut. Deshalb benötigt der erste Aufruf immer länger als nachfolgende Aufrufe. Der zusätzlich benötigte Speicherplatz entspricht dem Speicher, der für ein Bild benötigt wird. Es ist zu beachten, dass der Aufruf des Operators mit verschiedenen Bildgrößen zum Reallokieren des Caches führt, was wiederum zu längeren Verarbeitungszeiten führt. In diesem Fall kann es von Vorteil sein, das Caching nicht zu verwenden. Zum Deaktivieren des Cachings kann der Parameter GenParamName auf *'caching'* und GenParamValue auf *'no_cache'* gesetzt werden. Der Cache kann explizit durch Setzen von GenParamName auf *'caching'* und GenParamValue auf *'free_cache'* deallokiert werden. In den meisten Fällen wird das Verwenden des Caches aber empfohlen, d.h., der Parameter *'caching'* kann ignoriert werden bzw. seinen Default-Wert beibehalten.

Speichern und Laden von Optimierungsparametern

Die Optimierungsparameter für alle Algorithmen können durch write_fft_optimization_data und read_fft_optimization_data gespeichert und geladen werden.

Nicht offensichtliche Anwendungen

reconstruct_height_field_from_gradient hat einige nicht offensichtliche Anwendungsfelder, insbesondere im Bereich der "gradient domain manipulation technique". In vielen Anwendungen haben die Werte der Gradienten, die dem Operator übergeben werden, nicht die Semantik von Oberflächengradienten (d.h., den ersten Ableitungen der Höhen) sondern entsprechen den ersten Ableitungen anderer Parametertypen, typischerweise Grauwerten. D.h., die Gradienten haben die Semantik von Grauwertkanten. Wenn diese Gradienten auf unterschiedliche Arten verarbeitet werden, z.B. durch das Addieren oder Subtrahieren von Bildern oder durch eine Filterung, werden die ursprünglichen Gradienten verändert und ein nachfolgender Aufruf von reconstruct_height_field_from_gradient liefert ein verändertes Bild, in dem z.B. unerwünschte Kanten entfernt wurden oder der Kontrast lokal geändert wurde. Typische Anwendungen sind das Entfernen von Rauschen, die nahtlose Verschmelzung von Bildern oder die HDR (high dynamic range) Kompression.

Achtung

reconstruct_height_field_from_gradient berücksichtigt die Werte aller Pixel von Gradient, auch außerhalb seiner Region. Falls Gradient einen eingeschränkten Definitionsbereich hat, könnte man den relevanten viereckigen Bereich ausschneiden und daraus ein neues Bild mit voller Region erzeugen.

Parameter

- ▷ **Gradient** (input_object) ..... singlechannelimage ⇝ *object :* vector_field
  Gradientenfeld der Oberfläche.
- ▷ **HeightField** (output_object) ..... image ⇝ *object :* real
  Rekonstruiertes Höhenmodell.
- ▷ **ReconstructionMethod** (input_control) ..... string ⇝ *string*
  Typ der Rekonstruktionsmethode.
  Defaultwert: 'poisson'
  Werteliste: ReconstructionMethod ∈ {'fft_cyclic', 'rft_cyclic', 'poisson'}
- ▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'optimize_speed', 'caching'}
- ▷ **GenParamValue** (input_control) ..... integer-array ⇝ *integer* / real / string
  Werte der generischen Parameter.
  Defaultwert: []
  Werteliste: GenParamValue ∈ {'standard', 'patient', 'exhaustive', 'use_cache', 'no_cache', 'free_cache'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert reconstruct_height_field_from_gradient den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Literatur

M. Kazhdan, M. Bolitho, and H. Hoppe: „Poisson Surface Reconstruction.“ Symposium on Geometry Processing (June 2006).

Modul

3D Metrology

sfs_mod_lr (Image : Height : Slant, Tilt, Albedo, Ambient :)

Rekonstruieren der relativen Höhe aus den Grauwerten.

Der Operator sfs_mod_lr berechnet nach dem modifizierten Algorithmus von Lee und Rosenfeld aus dem Bild Image und einer Lichtquelle, die durch die Parameter Slant, Tilt, Albedo und Ambient bestimmt wird, die entsprechenden Höhen, die bei der Bilderzeugung vorgelegen haben. Die Lichtquelle liegt dabei im Unendlichen in der Richtung, die durch Slant und Tilt bestimmt werden. Der Parameter Albedo gibt den

Reflexionskoeffizienten der Oberfläche an. Ambient ist ein Maß für die Stärke der Hintergrundbeleuchtung. Dieser Parameter kann auf einen Wert größer als 0 gesetzt werden, falls z.B. der Weißabgleich der Kamera nicht genau vorgenommen worden ist.

Achtung

sfs_mod_lr nimmt an, dass die Höhen auf einem Gitter der Schrittweite 1 vorliegen sollen. Ist das nicht der Fall, so müssen die berechneten Höhen nach Ausführung der Funktion noch mit der Gitterschrittweite multipliziert werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet. sfs_mod_lr ist nur für Byte-Bilder implementiert.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
  Das schattierte Bild.
- ▷ **Height** (output_object) . . . . . image(-array) ⇝ *object :* real
  Das rekonstruierte Höhenmodell.
- ▷ **Slant** (input_control) . . . . . angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
  Defaultwert: 45.0
  Wertevorschläge: Slant ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich: 0.0 ≤ Slant ≤ 180.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 10.0
- ▷ **Tilt** (input_control) . . . . . angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
  Defaultwert: 45.0
  Wertevorschläge: Tilt ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich: 0.0 ≤ Tilt ≤ 360.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 10.0
- ▷ **Albedo** (input_control) . . . . . number ⇝ *real* / integer
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.
  Defaultwert: 1.0
  Wertevorschläge: Albedo ∈ {0.1, 0.5, 1.0, 5.0}
  Typischer Wertebereich: 0.0 ≤ Albedo ≤ 5.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Albedo >= 0.0
- ▷ **Ambient** (input_control) . . . . . number ⇝ *real* / integer
  Stärke der indirekten Beleuchtung.
  Defaultwert: 0.0
  Wertevorschläge: Ambient ∈ {0.1, 0.5, 1.0}
  Typischer Wertebereich: 0.0 ≤ Ambient ≤ 1.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Ambient >= 0.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert sfs_mod_lr den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

estimate_al_am, estimate_sl_al_lr, estimate_sl_al_zc, estimate_tilt_lr, estimate_tilt_zc, optimize_fft_speed

Nachfolger

`shade_height_field`

Modul

3D Metrology

sfs_orig_lr (Image : Height : Slant, Tilt, Albedo, Ambient :)

Rekonstruieren der relativen Höhe aus den Grauwerten.

Der Operator `sfs_orig_lr` berechnet nach dem originalen Algorithmus von Lee und Rosenfeld aus dem Bild `Image` und einer Lichtquelle, die durch die Parameter `Slant`, `Tilt`, `Albedo` und `Ambient` bestimmt wird, die entsprechenden Höhen, die bei der Bilderzeugung vorgelegen haben. Die Lichtquelle liegt dabei im Unendlichen in der Richtung, die durch `Slant` und `Tilt` bestimmt werden. Der Parameter `Albedo` gibt den Reflexionskoeffizienten der Oberfläche an. `Ambient` ist ein Maß für die Stärke der Hintergrundbeleuchtung. Dieser Parameter kann auf einen Wert größer als 0 gesetzt werden, falls z.B. der Weißabgleich der Kamera nicht genau vorgenommen worden ist.

Achtung

`sfs_orig_lr` nimmt an, dass die Höhen auf einem Gitter der Schrittweite 1 vorliegen sollen. Ist das nicht der Fall, so müssen die berechneten Höhen nach Ausführung der Funktion noch mit der Gitterschrittweite multipliziert werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet. `sfs_mod_lr` ist nur für Byte-Bilder implementiert.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
  Das schattierte Bild.
- ▷ **Height** (output_object) . . . . . image(-array) ⇝ *object :* real
  Das rekonstruierte Höhenmodell.
- ▷ **Slant** (input_control) . . . . . angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
  Defaultwert: 45.0
  Wertevorschläge: `Slant` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich: 0.0 ≤ `Slant` ≤ 90.0
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 10.0
- ▷ **Tilt** (input_control) . . . . . angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
  Defaultwert: 45.0
  Wertevorschläge: `Tilt` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich: 0.0 ≤ `Tilt` ≤ 360.0
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 10.0
- ▷ **Albedo** (input_control) . . . . . number ⇝ *real* / integer
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.
  Defaultwert: 1.0
  Wertevorschläge: `Albedo` ∈ {0.1, 0.5, 1.0, 5.0}
  Typischer Wertebereich: 0.0 ≤ `Albedo` ≤ 5.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Albedo >= 0.0

▷ **Ambient** (input_control) . . . . . number ⇝ *real* / integer
Stärke der indirekten Beleuchtung.
Defaultwert: 0.0
Wertevorschläge: Ambient ∈ {0.1, 0.5, 1.0}
Typischer Wertebereich: 0.0 ≤ Ambient ≤ 1.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Ambient >= 0.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert sfs_orig_lr den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

estimate_al_am, estimate_sl_al_lr, estimate_sl_al_zc, estimate_tilt_lr, estimate_tilt_zc, optimize_fft_speed

Nachfolger

shade_height_field

Modul

3D Metrology

sfs_pentland (Image : Height : Slant, Tilt, Albedo, Ambient :)

Rekonstruieren der relativen Höhe aus den Grauwerten.

Der Operator sfs_pentland berechnet nach dem Algorithmus von Pentland aus dem Bild Image und einer Lichtquelle, die durch die Parameter Slant, Tilt, Albedo und Ambient bestimmt wird, die entsprechenden Höhen, die bei der Bilderzeugung vorgelegen haben. Die Lichtquelle liegt dabei im Unendlichen in der Richtung, die durch Slant und Tilt bestimmt werden. Der Parameter Albedo gibt den Reflexionskoeffizienten der Oberfläche an. Ambient ist ein Maß für die Stärke der Hintergrundbeleuchtung. Dieser Parameter kann auf einen Wert größer als 0 gesetzt werden, falls z.B. der Weißabgleich der Kamera nicht genau vorgenommen worden ist.

Achtung

sfs_pentland nimmt an, dass die Höhen auf einem Gitter der Schrittweite 1 vorliegen sollen. Ist das nicht der Fall, so müssen die berechneten Höhen nach Ausführung der Funktion noch mit der Gitterschrittweite multipliziert werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet. sfs_pentland ist nur für Byte-Bilder implementiert.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
Das schattierte Bild.

▷ **Height** (output_object) . . . . . image(-array) ⇝ *object :* real
Das rekonstruierte Höhenmodell.

▷ **Slant** (input_control) . . . . . angle.deg ⇝ *real* / integer
Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
Defaultwert: 45.0
Wertevorschläge: Slant ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
Typischer Wertebereich: 0.0 ≤ Slant ≤ 180.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Tilt** (input_control) . . . . . angle.deg ⇝ *real* / integer
Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
Defaultwert: 45.0
Wertevorschläge: `Tilt` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
Typischer Wertebereich: 0.0 ≤ `Tilt` ≤ 360.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Albedo** (input_control) . . . . . number ⇝ *real* / integer
Anteil der Strahlung, der von der Oberfläche reflektiert wird.
Defaultwert: 1.0
Wertevorschläge: `Albedo` ∈ {0.1, 0.5, 1.0, 5.0}
Typischer Wertebereich: 0.0 ≤ `Albedo` ≤ 5.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Albedo >= 0.0

▷ **Ambient** (input_control) . . . . . number ⇝ *real* / integer
Stärke der indirekten Beleuchtung.
Defaultwert: 0.0
Wertevorschläge: `Ambient` ∈ {0.1, 0.5, 1.0}
Typischer Wertebereich: 0.0 ≤ `Ambient` ≤ 1.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Ambient >= 0.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `sfs_pentland` den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`estimate_al_am`, `estimate_sl_al_lr`, `estimate_sl_al_zc`, `estimate_tilt_lr`, `estimate_tilt_zc`, `optimize_fft_speed`

Nachfolger

`shade_height_field`

Modul

3D Metrology

shade_height_field (ImageHeight : ImageShade : Slant, Tilt, Albedo, Ambient, Shadows :)

Schattieren eines Höhenmodells.

`shade_height_field` berechnet aus einem Höhenmodell (einer Matrix, deren Einträge Höhen auf einem Gitter angeben) ein Bild, das das betreffende Höhenmodell bei Beleuchtung durch eine unendlich weit entfernte Punktlichtquelle darstellt. Dabei wird angenommen, dass die durch das Höhenmodell beschriebene Oberfläche Lambertsche Reflexionseigenschaften besitzt. Es kann zusätzlich bestimmt werden, ob Schatten berechnet werden sollen oder nicht.

Achtung

`shade_height_field` nimmt an, dass die Höhen auf einem Gitter der x- und y-Schrittweite 1 vorliegen. Ist dies nicht der Fall, so sollten die Höhen vorher entsprechend skaliert werden (z.B. durch Division durch die Gitter-Schrittweite), da die Ableitungen sonst als „zu flach“ oder „zu steil“ geschätzt werden. Beispiel: Das Höhenmodell ist an 100*100 Werten auf dem Quadrat [0,1]*[0,1] gegeben. Dann sollten die Höhen zunächst durch 1/100 geteilt

werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet.

Parameter

- ▷ **ImageHeight** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int4 / real
  Zu schattierendes Höhenmodell.
- ▷ **ImageShade** (output_object) . . . . . image(-array) ⇝ *object :* byte
  Schattiertes Bild.
- ▷ **Slant** (input_control) . . . . . angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
  Defaultwert: 0.0
  Wertevorschläge: `Slant` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich: 0.0 ≤ `Slant` ≤ 180.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 10.0
- ▷ **Tilt** (input_control) . . . . . angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
  Defaultwert: 0.0
  Wertevorschläge: `Tilt` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich: 0.0 ≤ `Tilt` ≤ 360.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 10.0
- ▷ **Albedo** (input_control) . . . . . number ⇝ *real* / integer
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.
  Defaultwert: 1.0
  Wertevorschläge: `Albedo` ∈ {0.1, 0.5, 1.0, 5.0}
  Typischer Wertebereich: 0.0 ≤ `Albedo` ≤ 5.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Albedo >= 0.0
- ▷ **Ambient** (input_control) . . . . . number ⇝ *real* / integer
  Stärke der indirekten Beleuchtung.
  Defaultwert: 0.0
  Wertevorschläge: `Ambient` ∈ {0.1, 0.5, 1.0}
  Typischer Wertebereich: 0.0 ≤ `Ambient` ≤ 1.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Ambient >= 0.0
- ▷ **Shadows** (input_control) . . . . . string ⇝ *string*
  Berechnung von Schatten ein- bzw. ausschalten.
  Defaultwert: 'false'
  Wertevorschläge: `Shadows` ∈ {'true', 'false'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `shade_height_field` den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `photometric_stereo`

Modul

Foundation

uncalibrated_photometric_stereo (Images : NormalField, Gradient, Albedo : ResultType :)

Rekonstruieren einer Oberfläche von mehreren, unterschiedlich beleuchteten Bildern.

Der Operator uncalibrated_photometric_stereo kann benutzt werden um die feinen Details einer Oberfläche eines gegebenen Objektes zu extrahieren, ohne dass man Wissen über die Beleuchtung, Geometrie und Spiegelungseigenschaften des Objektes hat. Die Geometrien von Interesse können z.B. Dellen, Falten oder Kratzer sein. Der Operator kann normalerweise nicht verwendet werden, um die allgemeine Form eines Objektes zu vermessen. Der Operator liefert die Normalen der Oberfläche in NormalField als ein 3-kanaliges Bild, wobei jedes Bild eine Komponente der Normalen kodiert. Dieses Bild wird für die Visualisierung des Ergebnisses als farbkodiertes Bild verwendet. Außerdem werden die Gradienten und die Albedo der Objektoberfläche in Gradient bzw. Albedo zurückgegeben. Welches Ergebnis berechnet werden soll kann mit Hilfe von ResultType eingestellt werden. Dieser Operator ist ähnlich zu photometric_stereo, benötigt jedoch keine kalibrierten Lichtrichtungen. Es ist zu beachten, dass photometric_stereo schneller und genauer ist, aber die Lichtrichtungen bekannt sein müssen. Für sinnvolle Ergebnisse muss, sowohl für den kalibrierten wie auch den unkalibrierten Fall, eine orthografische Projektion der Kamera angenommen werden. Dies wird erreicht, indem ein telezentrisches Objektiv oder zumindest ein Objektiv mit langer Brennweite verwendet wird.

Der Operator benötigt mindestens drei unterschiedlich beleuchtete Bilder eines Objekts, welches mit einer statischen Kamera aufgenommen wurde und für jedes Bild eine unterschiedliche Beleuchtungsrichtung. Die besten Resultate werden erzielt, wenn das Objekt sog. Lambert'sche Reflexionseigenschaften und keine Reflexionen oder Schattenwurf aufweist.

Parameter

- ▷ **Images** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2
  Eingabebilder mit jeweils unterschiedlicher Beleuchtungsrichtung.
- ▷ **NormalField** (output_object) ..... image(-array) ⇝ *object :* real
  Oberflächennormalen pro Pixel.
- ▷ **Gradient** (output_object) ..... image ⇝ *object :* vector_field
  Gradienten der Oberfläche in x,y Richtung.
- ▷ **Albedo** (output_object) ..... image ⇝ *object :* real
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.
- ▷ **ResultType** (input_control) ..... string-array ⇝ *string*
  Zu berechnende ikonische Ergebnisse.
  Defaultwert: 'all'
  Werteliste: ResultType ∈ {[], 'all', 'normal_field', 'gradient', 'normalized_gradient', 'albedo'}

Beispiel

```
* read severally illuminated images
FName := 'photometric_stereo/pharma_braille_0' + [1:4] + '.png'
read_image(Images, FName)
* extract surface normals, gradients and albedo from images
uncalibrated_photometric_stereo(Images, NormalField, Gradient, Albedo, 'all')
derivate_vector_field (Gradient, Result, 0.1, 'mean_curvature')
reconstruct_height_field_from_gradient (Gradient, HeightField, 'poisson', [], [])
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert uncalibrated_photometric_stereo den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

photometric_stereo

Siehe auch

photometric_stereo

Literatur

H. Hayakawa: „Photometric stereo under a light source with arbitrary motion“. Journal Optical Society America, Vol. 11, No. 11/November 1994.

Modul

3D Metrology

5.5 Sheet-of-light

apply_sheet_of_light_calibration (
Disparity : : SheetOfLightModelID :)

Wendet die Kalibriertransformationen auf das Eingabe-Disparitätsbild an.

Der Operator apply_sheet_of_light_calibration liest das Disparitätsbild Disparity und speichert es in das von SheetOfLightModelID referenzierte Lichtschnitt-Modell. Anschließend werden die Kalibriertransformationen auf dieses Disparitätsbild angewendet und somit die kalibrierten 3D-Punktkoordinaten der rekonstruierten Oberfläche berechnet. Diese 3D-Koordinaten können mit dem Operator get_sheet_of_light_result abgerufen werden. Das entsprechende 3D-Objektmodell kann mit dem Operator get_sheet_of_light_result_object_model_3d abgerufen werden. Vor dem nächsten Aufruf von apply_sheet_of_light_calibration für ein Disparitätsbild kleinerer Höhe sollte reset_sheet_of_light_model aufgerufen werden.

Das Disparitätsbild Disparity kann mit dem Operator measure_profile_sheet_of_light gemessen werden oder auch mit Hilfe einer speziellen Kamera akquiriert werden, welche Disparitätsbilder direkt nach dem Lichtschnittverfahren liefert.

Um die kalibrierten 3D-Koordinaten mit apply_sheet_of_light_calibration berechnen zu können, müssen vorher die folgenden Parameter in das Lichtschnitt-Modell gesetzt worden sein. Dies kann mit dem Operator set_sheet_of_light_param erfolgen.

'calibration': gibt an, welche Kalibriertransformationen auf das Disparitätsbild angewendet werden sollen und muss auf *'xz'*, *'xyz'* oder *'offset_scale'* gesetzt worden sein.

'camera_parameter': die internen Parameter der Kamera des Messsystems.

'camera_pose': die Pose des Weltkoordinatensystems relativ zum Kamerakoordinatesystem.

'lightplane_pose': die Pose des Koordinatensystems der Lichtebene relativ zum Weltkoordinatensystem. Mit dem Koordinatensystem der Lichtebene ist ein Koordinatensystem gemeint, dessen Ebene definiert durch z=0 mit der Lichtebene übereinstimmt, die der Lichtlinienprojektor im Raum erzeugt.

'movement_pose': die Pose, die die Bewegung des Objekts relativ zum Messsystem während der Messung beschreibt. Diese Pose soll im Weltkoordinatensystem formuliert sein.

'scale': dieser Parameter legt fest mit welchem Faktor die kalibrierten 3D-Koordinaten X, Y und Z skaliert werden. *'scale'* muss als der Quotient *Zieleinheit/Originaleinheit* eingegeben werden. Die Originaleinheit wird durch das Kalibrierobjekt bestimmt, sie ist Meter wenn Sie eine Standardkalibrierplatte benutzen.

Die Referenzdokumentation des Operators set_sheet_of_light_param liefert weitere Details über die Bedeutung aller Modell-Parameter.

Parameter

▷ **Disparity** (input_object) . . . . . singlechannelimage ⇝ *object : real*
Zu kalibrierende Entfernungs- oder Höhenmessung.

▷ **SheetOfLightModelID** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des Lichtschnitt-Modells.

Beispiel

```
* ...
* Read an already acquired disparity map from file
read_image (Disparity, 'sheet_of_light/connection_rod_disparity.tif')
*
* Create a model and set the required parameters
gen_rectangle1 (ProfileRegion, 120, 75, 195, 710)
create_sheet_of_light_model (ProfileRegion, ['min_gray','num_profiles', \
                             'ambiguity_solving'], [70,290,'first'], \
                              SheetOfLightModelID)
set_sheet_of_light_param (SheetOfLightModelID, 'calibration', 'xyz')
set_sheet_of_light_param (SheetOfLightModelID, 'scale', 'mm')
set_sheet_of_light_param (SheetOfLightModelID, 'camera_parameter', \
                          CameraParameter)
set_sheet_of_light_param (SheetOfLightModelID, 'camera_pose', CameraPose)
set_sheet_of_light_param (SheetOfLightModelID, 'lightplane_pose', \
                          LightPlanePose)
set_sheet_of_light_param (SheetOfLightModelID, 'movement_pose', \
                          MovementPose)
*
* Apply the calibration transforms and
* get the resulting calibrated coordinates
apply_sheet_of_light_calibration (Disparity, SheetOfLightModelID)
get_sheet_of_light_result (X, SheetOfLightModelID, 'x')
get_sheet_of_light_result (Y, SheetOfLightModelID, 'y')
get_sheet_of_light_result (Z, SheetOfLightModelID, 'z')
*
```

Ergebnis

`apply_sheet_of_light_calibration` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SheetOfLightModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Nachfolger

`get_sheet_of_light_result`, `get_sheet_of_light_result_object_model_3d`

Modul

3D Metrology

`calibrate_sheet_of_light` (: : SheetOfLightModelID : Error)

Kalibrieren eines Lichtschnitt-Modells mit einem 3D Kalibrierkörper.

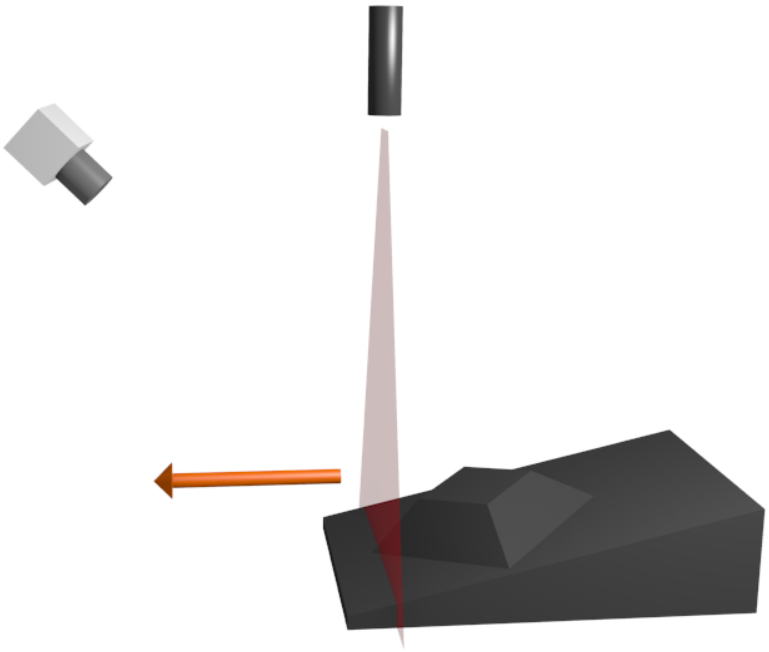
`calibrate_sheet_of_light` kalibriert das Lichtschnitt-Modell `SheetOfLightModelID` mit einem Disparitätsbild eines 3D Kalibrierkörpers und gibt den Rückprojektionsfehler der Optimierung in `Error` zurück.

Überblick

Die Kalibrierung eines Lichtschnitt Aufbaus mit `calibrate_sheet_of_light` ist einfacher als die Kalibrierung mit Standard HALCON Kalibrierplatten, wie es im HDevelop Beispiel calibrate_sheet_of_light_calplate.hdev

gezeigt wird. Es wird nur die unkalibrierte Rekonstruktion, das heißt ein Disparitätsbild, eines speziellen 3D Kalibrierkörpers benötigt um das Lichtschnitt-Modell zu kalibrieren.

Im folgenden werden die Schritte, die für die Kalibrierung nötig sind beschrieben.



Kalibrierung eines Lichtschnitt Aufbaus

3D Kalibrierkörper

Es wird ein spezieller 3D Kalibrierkörper benötigt. Dieser Kalibrierkörper muss dem mit `create_sheet_of_light_calib_object` erzeugten CAD Modell entsprechen. Der 3D Kalibrierkörper besteht aus einer schiefen Ebene, auf der sich ein Pyramidenstumpf befindet. Die dünne Seite wird im folgenden als die Vorderseite bezeichnet. Die dickere Seite als die Rückseite.

Die Dimensionen des Kalibrierkörpers sollten so gewählt sein, dass der Kalibrierkörper das Messvolumen abdeckt. Dabei ist zu beachten, dass nur der Bereich auf dem Kalibrierkörper, der über `HeightMin` liegt, in die Kalibrierung einfließt.

Das CAD-Modell, welches als DXF Datei geschrieben wurde, dient auch als Beschreibungsdatei für den Kalibrierkörper.

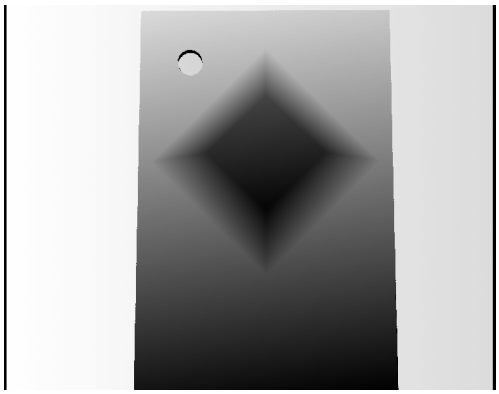
Vorbereitung des Lichtschnitt-Modells

Um ein Lichtschnitt-Modell auf die Kalibrierung vorzubereiten, müssen die folgenden Schritte durchgeführt werden:

- Erzeugen des Lichtschnitt-Modells mit `create_sheet_of_light_model` und Anpassen der Modelleinstellungen an die konkrete Messaufgabe.
- Setzen der Startkameraparameter mit `set_sheet_of_light_param`. Zunächst werden nur projektive Flächenkameras mit Divisionsmodell unterstützt, d.h. Kameras vom Typ *'area_scan_division'*.
- Setzen der Beschreibungsdatei des Kalibrierkörpers (erzeugt mit `create_sheet_of_light_calib_object`) mit `set_sheet_of_light_param`.

Unkalibrierte Rekonstruktion des 3D Kalibrierkörpers

Der 3D Kalibrierkörper muss mit dem (unkalibrierten) Lichtschnitt-Modell (das wie oben vorbereitet wurde) rekonstruiert werden, d.h. es muss ein Disparitätsbild des 3D Kalibrierkörper erzeugt werden.



Disparitätsbild eines 3D Kalibrierkörpers

Für diese Rekonstruktion muss der Kalibrierkörper so orientiert sein, dass entweder seine Vorderseite oder seine Rückseite die Lichtebene als erstes schneiden (das heißt, der Bewegungsvektor muss parallel zur Y Achse des Kalibrierkörpers sein, siehe `create_sheet_of_light_calib_object`). So weit wie möglich sollte die Region des Disparitätsbildes des Kalibrierkörpers auf den Kalibrierkörper eingeschränkt sein. Die Region des Disparitätsbildes sollte außerdem auf der abgeschnittenen Pyramide keine Löcher aufweisen. Alle vier Seiten der abgeschnittenen Pyramide müssen deutlich sichtbar sein.

Kalibrierung des Lichtschnitt Aufbaus

Die Kalibrierung wird mit `calibrate_sheet_of_light` durchgeführt. Der zurückgegebene Fehler `Error` ist der RMS der Distanz der rekonstruierten Punkte zum Kalibrierobjekt in Metern.

Bei mit `calibrate_sheet_of_light` kalibrierten Lichtschnitt-Modellen können die Parameter in seltenen Fällen ein unrealistisches Bild des Setups vermitteln, was jedoch die Messgenauigkeit der Kalibrierung nicht negativ beeinflusst.

Parameter

- ▷ **`SheetOfLightModelID`** (input_control) ..... sheet_of_light_model ⇝ *handle*
  Handle für den Zugriff auf das Lichtschnitt-Modell.
- ▷ **`Error`** (output_control) ..... number ⇝ *real*
  Durchschnittlicher verbleibender Fehler nach der Optimierung.

Beispiel

```
* Calibrate a sheet-of-light model with a 3D calibration object
gen_rectangle1 (Rectangle, 300, 0, 800, 1023)
CameraParam := ['area_scan_division', 0.016, 0, 4.65e-6, 4.65e-6, \
                640.0, 512.0, 1280, 1024]
create_sheet_of_light_model (Rectangle, 'min_gray', 50, SheetOfLightModelID)
set_sheet_of_light_param (SheetOfLightModelID, 'camera_parameter', \
                          CameraParam)
set_sheet_of_light_param (SheetOfLightModelID, 'calibration_object', \
                         'calib_object.dxf')
* Uncalibrated reconstruction of the calibration object
for ProfileIndex := 1 to 1000 by 1
    grab_image_async (Image, AcqHandle, -1)
    measure_profile_sheet_of_light (Image, SheetOfLightModelID, [])
endfor
* Calibration of the sheet-of-light-model
calibrate_sheet_of_light (SheetOfLightModelID, Error)
* Now get a calibrated reconstruction of the calibration object
get_sheet_of_light_result_object_model_3d (SheetOfLightModelID, \
                                           ObjectModel3D)
```

Ergebnis

`calibrate_sheet_of_light` liefert den Wert TRUE, falls die Kalibrierung erfolgreich war. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SheetOfLightModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_sheet_of_light_model`, `set_sheet_of_light_param`, `set_profile_sheet_of_light`, `measure_profile_sheet_of_light`

Nachfolger

`set_profile_sheet_of_light`, `apply_sheet_of_light_calibration`

Modul

3D Metrology

`clear_sheet_of_light_model ( : : SheetOfLightModelID : )`

Löschen eines Lichtschnitt-Modells und Freigabe des verwendeten Speichers.

Der Operator `clear_sheet_of_light_model` löscht ein mit `create_sheet_of_light_model` angelegtes Lichtschnitt-Modell und gibt den gesamten darin referenzierten Speicherplatz wieder frei. Das Handle des zu löschenden Lichtschnitt-Modells muss in `SheetOfLightModelID` übergeben werden. Nach dem Aufruf ist er ungültig.

Parameter

▷ **`SheetOfLightModelID`** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des zu löschenden Lichtschnitt-Modells.

Ergebnis

`clear_sheet_of_light_model` liefert den Wert TRUE, falls das übergebene Handle des Lichtschnitt-Modells gültig ist, und das darüber referenzierte Modell vollständig freigegeben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SheetOfLightModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

`create_sheet_of_light_model`

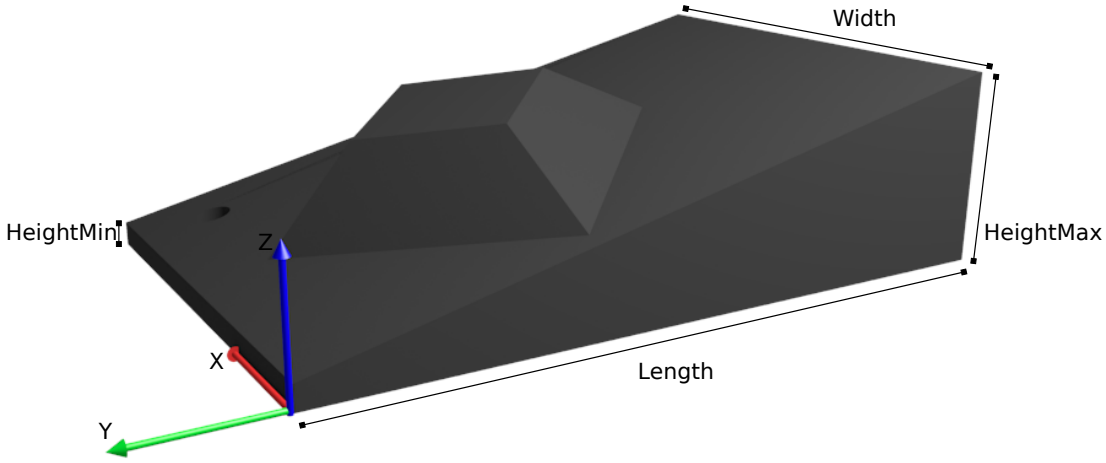
Modul

3D Metrology

create_sheet_of_light_calib_object (: : Width, Length, HeightMin, HeightMax, FileName :)

Erzeugen eines Kalibrierobjekts für die Lichtschnittkalibrierung.

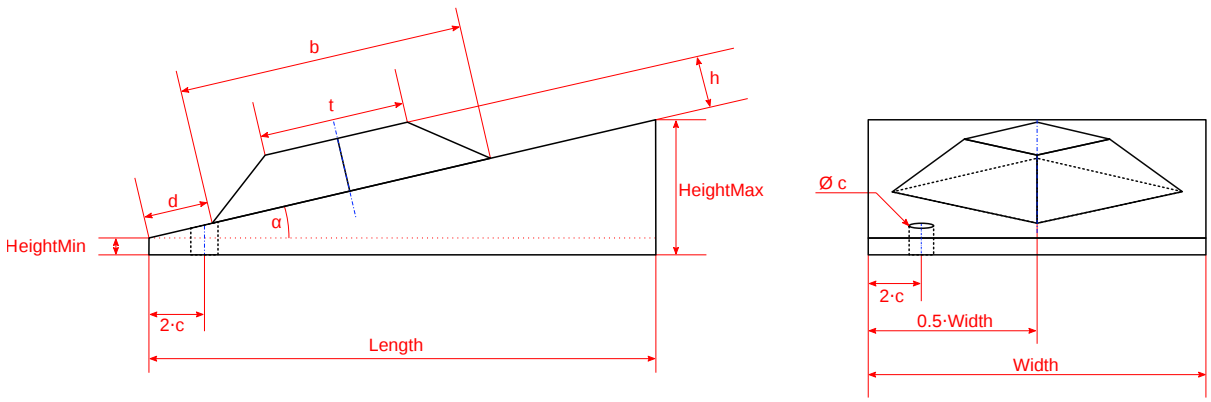
create_sheet_of_light_calib_object erzeugt ein CAD-Modell eines Kalibrierobjekts für die Lichtschnittkalibrierung mit calibrate_sheet_of_light und speichert es in FileName.



Ein Kalibrierobjekt für die Lichtschnittkalibrierung

Das Kalibrierobjekt besteht aus einer schiefen Ebene mit einem um 45 Grad gedrehten Pyramidenstumpf. In dem Kalibrierkörper befindet sich eine Orientierungsmarkierung in Form eines kreisförmigen Loches. Die Abmessungen (Width, Length, HeightMin und HeightMax) des Kalibrierobjekts müssen in Metern angegeben werden. Die Länge Length muss mindestens 10% größer sein als die Breite Width. Die Z-Koordinate des höchsten Punktes auf dem Pyramidenstumpf ist höchstens HeightMax. Das Kalibrierobjekt wird eventuell nicht von calibrate_sheet_of_light gefunden, wenn der Höhenunterschied zwischen dem gedrehten Pyramidenstumpf und der schiefen Ebene zu gering ist. In diesem Fall müssen entweder HeightMin und HeightMax entsprechend angepasst werden, oder es kann die Abtastrate bei der Aufnahme der Kalibrierdaten erhöht werden.

Die Abmessungen des Kalibrierobjekts sollten so gewählt werden, dass es möglich ist, damit das Messvolumen des Lichtschnittaufbaus abzudecken. Bei der Wahl der Länge (Length) des Objekts sollte außerdem die Geschwindigkeit des Lichtschnittaufbaus berücksichtigt werden, so dass das Kalibrierobjekt durch ausreichend viele Profilmessungen abgetastet wird.



Technische Zeichnung des Kalibrierkörpers, wobei c der Durchmesser der Orientierungsmarkierung ist, d ist der Abstand der Pyramide von der Vorderseite des Objekts, h ist die Höhe des Pyramidenstumpfes, b ist die Länge

der Diagonalen des Pyramidenstumpfes auf dessen Unterseite, `t` ist die entsprechende Länge auf der Oberseite und α ist der Winkel der schiefen Ebene.

Das erzeugte Kalibrierobjekt kann in `calibrate_sheet_of_light` verwendet werden, indem der Parameter *'calibration_object'* mit `set_sheet_of_light_param` auf `FileName` gesetzt wird.

Bitte beachten Sie, dass MVTec keine 3D-Kalibrierobjekte anbietet. Nutzen Sie `create_sheet_of_light_calib_object`, um ein individuelles CAD-Modell zu erstellen. Dieses können Sie dann nutzen, um das benötigte 3D-Kalibrierobjekt herzustellen. Es hat sich bewährt, dieses aus Aluminium zu fräsen. Je nach gewünschter Messgenauigkeit kann dabei aber die thermische Stabilität problematisch sein. Wichtig ist dabei, dass das Objekt eine helle Oberfläche hat. Die Farbe ist gegebenenfalls so anzupassen, dass genug Kontrast zur Farbe des Lasers besteht. Außerdem sollte die Oberfläche nicht durchsichtig sein und nicht spiegeln. Um dies zu erreichen, kann das Objekt eloxiert oder lackiert werden. Besonders beim Lackieren ist aber zu beachten, dass die Genauigkeit aufgrund der aufgetragenen Lackschicht sinken kann. Zu rau darf die Oberfläche jedoch auch nicht sein, sonst sinkt die Genauigkeit. Es empfiehlt sich, das Objekt nach der Herstellung nachmessen zu lassen, um festzustellen, ob die gewünschte Genauigkeit erreicht wird. Prinzipiell gilt, dass die Genauigkeit des Kalibrierkörpers um Faktor zehn besser sein sollte als die gewünschte Messgenauigkeit. Die bei der Messung ermittelten Werte können dann manuell in die DXF Datei eingetragen werden, welche dann für die Kalibrierung mit `calibrate_sheet_of_light` verwendet werden kann.

Parameter

- ▷ **`Width`** (input_control) ..... number ⇝ *real*
  Breite des Körpers.
  Defaultwert: 0.1
- ▷ **`Length`** (input_control) ..... number ⇝ *real*
  Länge des Körpers.
  Defaultwert: 0.15
- ▷ **`HeightMin`** (input_control) ..... number ⇝ *real*
  Minimale Höhe der Grundebene.
  Defaultwert: 0.005
- ▷ **`HeightMax`** (input_control) ..... number ⇝ *real*
  Maximale Höhe der Grundebene.
  Defaultwert: 0.04
- ▷ **`FileName`** (input_control) ..... filename.write ⇝ *string*
  Dateiname für das Modell des Kalibrierobjekts.
  Defaultwert: 'calib_object.dxf'
  Dateiendung: `.dxf`

Ergebnis

`create_sheet_of_light_calib_object` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

3D Metrology

`create_sheet_of_light_model` `( ProfileRegion : : GenParamName, GenParamValue : SheetOfLightModelID )`

Anlegen eines generischen Modells zur Durchführung von 3D-Messungen nach dem Lichtschnittverfahren.

Mit Hilfe von `create_sheet_of_light_model` wird ein generisches Modell zur Durchführung von 3D-Messungen nach dem Lichtschnittverfahren angelegt.

Das Lichtschnittverfahren ermöglicht es, die Oberfläche von lichtstreuenden Körpern zu rekonstruieren. Hierfür werden eine Kamera mit Flächensensor und ein Lichtlinienprojektor (typischerweise ein Laserlinienprojektor) benötigt. Der Linienprojektor und die Kamera haben eine feste Anordnung zueinander. Ihre optischen Achsen bilden einen sogenannten Triangulierungswinkel, dessen Wert typischerweise zwischen 30° und 60° ausgewählt wird. Die projizierte Lichtlinie beschreibt eine Ebene im Raum. Diese Ebene schneidet die Oberfläche des zu messenden Körpers, so dass ein Profil der Oberfläche für die Kamera sichtbar wird. Durch relative Bewegung des zu messenden Körpers unter dem Messsystem (bestehend aus Kamera und Linienprojektor in fester Anordnung) können nacheinander mehrere Profile aufgenommen werden, und somit die gesamte Oberfläche des Körpers rekonstruiert werden. Je nach Anforderung kann der zu messende Körper unter dem Messsystem bewegt werden oder auch das Messsystem über dem Körper. Bitte bemerken Sie, dass die Profile annähernd waagerecht orientiert sein müssen, weil sie spaltenweise verarbeitet werden.

Wenn die geometrischen Eigenschaften des Messsystems bekannt sind, ist es möglich kalibrierte dreidimensionale Koordinaten der rekonstruierten Oberfläche zu berechnen. Die Referenzdokumentation zum Operator `set_sheet_of_light_param` liefert einen Überblick über die hierzu benötigte Information. Sind diese Informationen nicht vorhanden, so wird als Ergebnis der Messung nur ein Disparitätsbild erzeugt, bei welchem die Pixelwerte der subpixel-genauen Position des detektierten Profils entsprechen.

Als Ergebnis liefert der Operator in `SheetOfLightModelID` ein Handle zurück, das für alle weiteren Lichtschnittverfahren-Operationen verwendet wird, beispielsweise um Modelleinstellungen zu ändern, um Profile zu messen, um Kalibriertransformationen anzuwenden oder um auf die Messergebnisse zuzugreifen.

Erforderliche ikonische Eingabeparameter

Der folgende ikonische Parameter muss gesetzt werden, um Messungen durchführen zu können:

`ProfileRegion`: definiert die Region des Profilbildes, die durch den Operator `measure_profile_sheet_of_light` verarbeitet wird. Diese Region sollte rechteckig sein und kann zum Beispiel mit Hilfe von `gen_rectangle1` erzeugt werden. Wenn `ProfileRegion` nicht rechteckig ist wird ihre kleinste umschließende Region benutzt. `ProfileRegion` wird nur von dem Operator `measure_profile_sheet_of_light` berücksichtigt und wird bei der Verarbeitung von Disparitätsbildern ignoriert.

Standardeinstellung des Lichtschnitt-Modells

Die Standardeinstellungen des Lichtschnitt-Modells wurden ausgewählt, um Messungen in einer Grundkonfiguration durchführen zu können. Die folgende Liste gibt einen Überblick über die Standardwerte der Modellparameter (siehe `set_sheet_of_light_param` für eine detaillierte Beschreibung der Modellparameter):

'method' wird auf *'center_of_gravity'* gesetzt

'min_gray': wird auf 100 gesetzt

'num_profiles' wird auf 512 gesetzt

'ambiguity_solving' wird auf *'first'* gesetzt

'score_type' wird auf *'none'* gesetzt

'calibration' wird auf *'none'* gesetzt

Ändern der Modelleinstellung

Wir empfehlen die Modelleinstellungen an die konkrete Messaufgabe anzupassen, um die Qualität der Messung zu erhöhen oder um die Laufzeit zu verkürzen. Eine Änderung der Modelleinstellung ist ebenfalls erforderlich, um kalibrierte Messwerte zu ermitteln.

Mit Hilfe der generischen Parameter `GenParamName` und `GenParamValue` ist es möglich die Standardwerte des Lichtschnitt-Modells zu ändern. Diese Änderungen können ebenfalls mit dem Operator `set_sheet_of_light_param` durchgeführt werden. Bei diesem Operator kann jedoch nur ein einziger Parameter verändert werden, während mit `create_sheet_of_light_model` mehrere Parameter mit einem Aufruf verändert werden können. `set_sheet_of_light_param` liefert eine detaillierte Beschreibung der Modellparameter.

Bitte beachten Sie, dass die folgenden Modellparameter nicht mit dem Operator `create_sheet_of_light_model`, sondern nur mit dem Operator `set_sheet_of_light_param` gesetzt werden können: *'camera_parameter'*, *'camera_pose'*, *'lightplane_pose'*, und *'movement_pose'*.

In jedem Fall können die aktuell eingestellten Modellparameter mit get_sheet_of_light_param ausgelesen werden. Die Namen der Parameter, die für das konkrete Modell gesetzt bzw. abgefragt werden können, liefert query_sheet_of_light_params.

Verwendung des vereinfachten Parameter des Lichtschnitt-Modells

Wenn ein einfacher Aufbau vorliegt oder keine metrische Messung nötig ist, können die Disparitäten mit einem stark vereinfachten Parametersatz in 3D umgewandelt werden.

Durch das Setzen der Kalibrierung mit Hilfe von set_sheet_of_light_param auf den Wert *'offset_scale'* werden die Kameraparameter und Posen des Lichtschnitt-Modells so gesetzt, dass Pixel als Maßstab für eine Überführung in 3D genutzt werden können. Das erlaubt das erzeugen von 3D Modellen und 3D Bildern aus einem unkalibrierten Aufbau.

Die sechs vereinfachten Parameter, die die Transformation von Disparitäten in 3D steuern sind: *'scale_x'*, *'scale_y'*, *'scale_z'*, *'offset_x'*, *'offset_y'*,*'offset_z'*. Die Beschreibung von set_sheet_of_light_param enthält detaillierte Informationen über diese Parameter.

Verwendung des Handles in mehreren Threads

Es ist zu beachten, dass man bei der Verwendung eines Lichtschnitt-Modells SheetOfLightModelID in mehreren Threads sorgfältig umgehen muss. Ein und dasselbe Handle kann nicht gleichzeitig von unterschiedlichen Threads verwendet werden, wenn diese das Handle modifizieren. Somit muss man besonders vorsichtig sein, wenn die Threads solche Operatoren aufrufen, die Änderungen an den Daten des Handles vornehmen. Ein entsprechender Hinweis ist in dem *'Attention'* Abschnitt der Operatoren zu finden. Falls man trotzdem dasselbe Handle in Operatoren verwenden möchte, die gleichzeitig in unterschiedlichen Threads das Handle modifizieren, müssen die Threads synchronisiert werden. Damit wird sichergestellt, dass die Threads keinen gleichzeitigen Zugriff auf das Handle haben. Wenn man nicht sicher ist, ob die Verwendung eines Handles thread-safe ist, kann man im *'Attention'* Abschnitt des entsprechenden Operatoreintrags in der Operator-Referenz nachsehen, ob es eine Warnung gibt, die auf dieses Problem hinweist. Unterschiedliche Handles hingegen können unabhängig und sicher in unterschiedlichen Threads verwendet werden.

Parameter

- ▷ **ProfileRegion** (input_object) ..... region ⇝ *object*
  Region in welcher die zu detektierenden Profile gesucht werden. Wenn diese Region nicht rechteckig ist, wird die kleinste umschließende rechteckige Region verwendet.
- ▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
  Namen der generischen Parameter, die für die Detektion des Profils eingestellt werden können.
  Defaultwert: 'min_gray'
  Werteliste: GenParamName ∈ {'min_gray', 'method', 'ambiguity_solving', 'score_type', 'num_profiles', 'calibration', 'scale', 'scale_x', 'scale_y', 'scale_z', 'offset_x', 'offset_y', 'offset_z'}
- ▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *integer* / real / string
  Wert der generischen Parameter, die für die Detektion des Profils eingestellt werden können.
  Defaultwert: 50
  Wertevorschläge: GenParamValue ∈ {'default', 'center_of_gravity', 'last', 'first', 'brightest', 'none', 'intensity', 'width', 'offset_scale', 50, 100, 150, 180}
- ▷ **SheetOfLightModelID** (output_control) ..... sheet_of_light_model ⇝ *handle*
  Handle für den Zugriff auf das Lichtschnitt-Modell

Beispiel

```
* Create the rectangular region in which the profiles are measured.
gen_rectangle1 (ProfileRegion, 120, 75, 195, 710)
*
* Create a model in order to measure profiles according to
* the sheet-of-light technique. Simultaneously set some
* parameters for the model.
create_sheet_of_light_model (ProfileRegion, ['min_gray','num_profiles', \
                             'ambiguity_solving','score_type'], \
                             [70,290,'first','width'], \
  SheetOfLightModelID)
*
* Measure the profile from successive images
```

```
for Index := 1 to 290 by 1
    read_image (ProfileImage, 'sheet_of_light/connection_rod_'+Index$'.3')
    dev_display (ProfileImage)
    dev_display (ProfileRegion)
    measure_profile_sheet_of_light (ProfileImage, SheetOfLightModelID, [])
endfor
*
* Get the resulting disparity and score images
get_sheet_of_light_result (Disparity, SheetOfLightModelID, 'disparity')
get_sheet_of_light_result (Score, SheetOfLightModelID, 'score')
*
* Close the sheet-of-light handle once the measurement
* has been performed
```

Ergebnis

`create_sheet_of_light_model` liefert den Wert TRUE, wenn die übergebenen Parameter korrekt sind, und das Model angelegt werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`gen_rectangle1`

Nachfolger

`set_sheet_of_light_param`, `measure_profile_sheet_of_light`

Siehe auch

`clear_sheet_of_light_model`, `calibrate_sheet_of_light`

Modul

3D Metrology

deserialize_sheet_of_light_model (: : SerializedItemHandle : SheetOfLightModelID)

Deserialisieren eines Lichtschnitt-Modells.

`deserialize_sheet_of_light_model` deserialisiert ein Lichtschnitt-Modell, welches mit `serialize_sheet_of_light_model` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Modell wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Lichtschnitt-Modell mit dem Handle `SheetOfLightModelID` gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **SheetOfLightModelID** (output_control) ..... sheet_of_light_model ⇝ *handle*
Handle für das deserialisierte Lichtschnitt-Modell.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_sheet_of_light_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_sheet_of_light_model`

Nachfolger

`measure_profile_sheet_of_light`

Alternativen

`create_sheet_of_light_model`

Siehe auch

`serialize_sheet_of_light_model`

Modul

3D Metrology

get_sheet_of_light_param (: : SheetOfLightModelID, GenParamName : GenParamValue)

Abfrage eines Modellparameters.

Der Operator `get_sheet_of_light_param` erlaubt es, die Werte der Modellparameter abzufragen. Der Name des gewünschten Parameters wird dem generischen Parameter `GenParamName` übergeben und der entsprechende Wert wird im Parameter `GenParamValue` zurückgegeben. Die aktuelle Konfiguration können mit Hilfe von `set_sheet_of_light_param` jederzeit verändert werden.

Die gleichzeitige Abfrage mehrerer Parameter ist nicht möglich. Um mehrere Parameter abzufragen, muss der Operator `get_sheet_of_light_param` mehrmals aufgerufen werden.

Die Werte der folgenden Modellparameter kann abgefragt werden:

Messung der Profile:

'method': definiert die Methode anhand welcher die Position des Profils im Bild, d.h. die Disparität, bestimmt wird. Die Werte *'default'* und *'center_of_gravity'* verweisen beide auf dieselbe Methode, bei welcher die Position des Profils spaltenweise und mit Subpixelgenauigkeit ermittelt wird, indem der Schwerpunkt der Grauwerte g_i berechnet wird, die folgende Bedingung erfüllen:

$$g_i \geq 'min_gray'$$

'min_gray': der kleinste Grauwert, der für die Berechnung der Disparität berücksichtigt wird (vgl. *'method'* oben).

'num_profiles': Anzahl der Profile für welche Speicher im Lichtschnitt-Modell reserviert wurde. Standardmäßig wird *'num_profiles'* auf 512 gesetzt. Wird diese Anzahl während einer Messung überschritten, so wird zusätzlicher Speicher automatisch reserviert. Da dieser Prozess jedoch etwas Zeit in Anspruch nimmt, wird empfohlen *'num_profiles'* vor dem Start der Messung auf einen geeigneten Wert zu setzen.

'ambiguity_solving': Dieser Parameter legt fest, welcher gemessene Wert zurückgeliefert wird, wenn die Bestimmung der Position des Profils nicht eindeutig ist.

'first': der erste gemessene Wert wird zurückgeliefert. Diese Methode ist die schnellste.

'last': der letzte gemessene Wert wird zurückgeliefert.

'brightest': die Helligkeit aller gemessenen Werte wird berechnet und die Position, die der höchsten Helligkeit entspricht, wird zurückgeliefert. Die Helligkeit wird wie folgt berechnet:

$$brightness = \frac{1}{n} \sum_{i=0}^{n} g_i \ ,$$

wobei g_i die Grauwerte und n die Anzahl der Pixel darstellen, die zur Berechnung der Position der Profils berücksichtigt werden.

'score_type': Dieser Parameter legt fest, ob Bewertungsmerkmale (*'score'*) des Disparitätsbildes berechnet werden sollen, und wenn ja, nach welchem Kriterium.

'none': es werden keine Bewertungsmerkmale berechnet.

'width': für jeden Pixel des Disparitätsbildes wird die lokale Breite des Profils als Bewertungsmerkmal zurückgeliefert.

'intensity': für jeden Pixel des Disparitätsbildes wird als Bewertungsmerkmal die lokale Intensität des Profils zurückgeliefert. Diese wird wie folgt berechnet:

$$score = \frac{1}{n} \sum_{i=0}^{n} g_i$$

wobei g_i die Grauwerte und n die Anzahl der Pixel darstellen, die zur Berechnung der Position der Profils berücksichtigt werden.

Kalibrierung der Messung:

'calibration': gibt an, ob und wenn ja welche Kalibriertransformationen auf das Disparitätsbild angewendet werden sollen:

'none': es werden keine Kalibriertransformationen angewendet.

'xz': es werden Kalibriertransformationen angewendet, die die geometrischen Eigenschaften des Messsystems berücksichtigen (Kamera und Lichtlinienprojektor). Die Bewegung des Objekts während der Messung wird jedoch nicht berücksichtigt.

'xyz': es werden Kalibriertransformationen angewendet, die sowohl die geometrischen Eigenschaften des Messsystems (Kamera und Lichtlinienprojektor) als auch die Bewegung des Objekts während der Messung berücksichtigen.

'offset_scale': Ein vereinfachter Parametersatz um einen Aufbau zu beschreiben. Dieses Modell kann bereits mit Standardeinstellungen angewandt werden, bietet aber auch die Möglichkeit den Setup mit nur 6 Parameters zu konfigurieren. Drei der Parameter beschreiben eine anisotrope Skalierung: *'scale_x'* beschreibt die Skalierung entlang der Spaltenrichtung des Bildes in die neue x-Achse, *'scale_y'* beschreibt eine lineare Bewegung des Objekts zwischen zwei aufeinander folgenden Bildern und *'scale_z'* beschreibt das Verhältnis der Disparität zur neuen z-Achse. Die übrigen drei Parameter beschreiben die Lage des neuen Zentrums relativ zum neuen Weltkoordinatensystem (*'offset_x'*, *'offset_y'*, *'offset_z'*).

'camera_parameter': die internen Parameter der Kamera, die zur Durchführung der Messung benutzt wurden. Diese Parameter werden benötigt, wenn *'calibration'* auf *'xz'* oder *'xyz'* gesetzt wurde.

'camera_pose': die Pose des Weltkoordinatensystems relativ zum Kamerakoordinatesystem. Diese Pose wird benötigt, wenn *'calibration'* auf *'xz'* oder *'xyz'* gesetzt wurde.

'lightplane_pose': die Pose des Koordinatensystems der Lichtebene relativ zum Weltkoordinatensystem. Mit dem Koordinatensystem der Lichtebene ist ein Koordinatensystem gemeint, dessen Ebene definiert durch z=0 mit der Lichtebene übereinstimmt, die der Lichtlinienprojektor im Raum erzeugt. Diese Pose wird benötigt, wenn *'calibration'* auf *'xz'* oder *'xyz'* gesetzt wurde.

'movement_pose': die Pose, die die Bewegung des Objekts relativ zum Messsystem während der Messung beschreibt. Sie soll im Weltkoordinatensystem formuliert sein und wird benötigt, wenn *'calibration'* auf *'xyz'* gesetzt wurde.

'scale': dieser Parameter legt fest mit welchem Faktor die kalibrierten 3D-Koordinaten X, Y und Z skaliert werden. *'scale'* muss als der Quotient *Zieleinheit/Originaleinheit* eingegeben werden. Die Originaleinheit wird durch das Kalibrierobjekt bestimmt, und ist Meter, wenn Sie eine Standardkalibrierplatte benutzen. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'*, *'xz'* oder *'xyz'* relevant.

Der Standardwert des Modellparameters *'scale'* ist 1.0.

'scale_x': Dieser Wert definiert die Breite eines Pixels im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Der Standardwert für *'scale_x'* ist *1.0*.

'scale_y': Dieser Wert definiert die lineare Bewegung zwischen zwei Profilen. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Der Standardwert für *'scale_y'* ist *10.0*.

'scale_z': Dieser Wert definiert die Höher von Disparitäten im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Der Standardwert für *'scale_z'* ist *1.0*.

'offset_x': Dieser Wert beschreibt die Verschiebung in x im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.
Der Standardwert für *'offset_x'* ist *0.0*.

'offset_y': Dieser Wert beschreibt die Verschiebung in y im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.
Der Standardwert für *'offset_y'* ist *0.0*.

'offset_z': Dieser Wert beschreibt die Verschiebung in z im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.
Der Standardwert für *'offset_z'* ist *0.0*.

Parameter

▷ **SheetOfLightModelID** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des Lichtschnitt-Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
Name des Modellparameters, der abgefragt werden soll.
Defaultwert: 'method'
Werteliste: GenParamName ∈ {'min_gray', 'method', 'ambiguity_solving', 'score_type', 'num_profiles', 'calibration', 'camera_parameter', 'camera_pose', 'lightplane_pose', 'movement_pose', 'scale', 'scale_x', 'scale_y', 'scale_z', 'offset_x', 'offset_y', 'offset_z'}

▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Wert des abgefragten Modellparameters.

Ergebnis

get_sheet_of_light_param liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

query_sheet_of_light_params, set_sheet_of_light_param

Nachfolger

measure_profile_sheet_of_light, set_sheet_of_light_param, apply_sheet_of_light_calibration

Modul

3D Metrology

get_sheet_of_light_result (: ResultValue : SheetOfLightModelID, ResultName :)

Zugriff auf die ikonischen Ergebnisse einer Lichtschnittmessung.

Der Operator get_sheet_of_light_result ermöglicht den Zugriff auf die in einem Lichtschnittmodell gespeicherten ikonischen Messergebnisse.

Die verschiedenen Messergebnisse können, wie unten erläutert, mit dem Parameter ResultName spezifiziert werden:

Nicht-kalibrierte Ergebnisse:

'disparity': das gemessene Disparitätsbild, d.h. die subpixelgenaue Position der detektierten Profile. Die Disparitätswerte können als nicht-kalibrierte Pseudo-Entfernungswerte betrachtet werden.

'score': die Bewertungsmerkmale des gemessenen Disparitätsbildes werden zurückgeliefert. Diese werden entsprechend dem Wert des Modellparameters *'score_type'* berechnet. Wurde *'score_type'* auf *'none'* gesetzt, so wird ein leeres Bild zurückgeliefert. Die Referenzdokumentation des Operators set_sheet_of_light_param liefert Details über die möglichen Werte des Parameters *'score_type'*.

Kalibrierte Ergebnisse:

'x': Die kalibrierten X-Koordinaten der 3D-Punkte der rekonstruierten Oberfläche. Die Koordinaten werden als Pixelwerte zurückgeliefert.

'y': Die kalibrierten Y-Koordinaten der 3D-Punkte der rekonstruierten Oberfläche. Die Koordinaten werden als Pixelwerte zurückgeliefert.

'z': Die kalibrierten Z-Koordinaten der 3D-Punkte der rekonstruierten Oberfläche. Die Koordinaten werden als Pixelwerte zurückgeliefert.

Beachten Sie, dass bei den Bildern die zurückgeliefert werden, wenn `ResultName` auf *'x'*, *'y'* oder *'z'* gesetzt wurde, die Pixelwerte die Semantik von Koordinaten haben. Diese Koordinaten beziehen sich auf das Weltkoordinatensystem, das implizit durch die Festlegung der Pose der Kamera bestimmt wurde. Die Einheit dieser Koordinaten ist vom Wert des Parameters *'scale'* abhängig. Der Operator `set_sheet_of_light_param` liefert weitere Details über die möglichen Werte des Parameters *'scale'*.

`get_sheet_of_light_result` liefert ein leeres Objekt zurück, wenn das angeforderte Ergebnis nicht berechnet wurde und im Lichtschnitt-Modell nicht vorhanden ist.

Parameter

▷ **`ResultValue`** (output_object) ..... singlechannelimage ⇝ *object : real*
Gewünschtes Messergebnis.

▷ **`SheetOfLightModelID`** (input_control) ..... sheet_of_light_model ⇝ *handle*
Handle des zu verwendenden Lichtschnitt-Modells.

▷ **`ResultName`** (input_control) ..... string(-array) ⇝ *string*
Messergebnis, das ausgegeben werden soll.
Defaultwert: 'disparity'
Werteliste: `ResultName` ∈ {'disparity', 'score', 'x', 'y', 'z'}

Ergebnis

`get_sheet_of_light_result` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

3D Metrology

`get_sheet_of_light_result_object_model_3d` (
`: : SheetOfLightModelID : ObjectModel3D )`

Zugriff auf das Ergebnis einer vollständig kalibrierten Lichtschnittmessung als 3D-Objektmodell.

Der Operator `get_sheet_of_light_result_object_model_3d` liefert das Ergebnis einer vollständig kalibrierten Lichtschnittmessung als 3D-Objektmodell. Das Handle des Lichtschnitt-Modells, dessen Messergebnis benötigt wird, muss in `SheetOfLightModelID` übergeben werden. Der Parameter *'calibration'* des Lichtschnitt-Modells muss auf den Wert *'xyz'* oder *'offset_scale'* gesetzt worden sein. Ansonsten können die gemessenen Punktkoordinaten nicht in ein 3D-Objektmodell überführt werden und eine Fehlerbehandlung wird durchgeführt.

Das Handle des 3D-Objektmodells, das das Ergebnis der Lichtschnittmessung beinhaltet, wird in `ObjectModel3D` zurückgegeben. Die 3D-Punkte im 3D-Objektmodell sind nicht vermascht, so dass keine Oberflächenelemente (Facetten) im Modell verfügbar sind. Wenn ein 3D-Objektmodell mit vermaschten 3D-Punkten benötigt wird, muss zunächst der Operator `get_sheet_of_light_result` aufgerufen werden, um die gemessenen *'x'*, *'y'* und *'z'* Koordinaten zu erhalten. Anschließend muss der Operator `xyz_to_object_model_3d` mit passenden Parametern aufgerufen werden. Siehe `xyz_to_object_model_3d` für weitere Informationen über 3D-Objektmodelle.

Die Einheit der aus dem Lichtschnitt-Modell erhaltenen Koordinaten ist vom Wert des Parameters *'scale'* abhängig. Der Operator `set_sheet_of_light_param` liefert weitere Details über die möglichen Werte des Parameters *'scale'*.

`get_sheet_of_light_result_object_model_3d` liefert ein leeres 3D-Objektmodell zurück, wenn das angeforderte Ergebnis nicht gemessen wurde und im Lichtschnitt-Modell nicht vorhanden ist.

Parameter

▷ **`SheetOfLightModelID`** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle für den Zugriff auf das Lichtschnitt-Modell.

▷ **`ObjectModel3D`** (output_control) . . . . . object_model_3d ⇝ *handle*
Handle des resultierenden 3D-Objektmodells.

Ergebnis

`get_sheet_of_light_result_object_model_3d` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`create_sheet_of_light_model`, `measure_profile_sheet_of_light`, `calibrate_sheet_of_light`

Nachfolger

`clear_object_model_3d`

Modul

3D Metrology

`measure_profile_sheet_of_light` (
`ProfileImage : : SheetOfLightModelID, MovementPose : )`

Messung eines 3D-Profils anhand eines Lichtschnittbildes.

Der Operator `measure_profile_sheet_of_light` misst das im Eingabebild `ProfileImage` vorhandene Profil und speichert die resultierenden Disparitätswerte in das Lichtschnitt-Modell. Beachten Sie, dass `ProfileImage` nur in der Region `ProfileRegion` bearbeitet wird, die mit dem Operator `create_sheet_of_light_model` für das Lichtschnitt-Modell gesetzt wurde. Da `ProfileImage` spaltenweise verarbeitet wird, muss das Profil annähernd waagerecht orientiert sein.

Einfluss verschiedener Modell-Parameter

Wenn der Modellparameter *'score_type'* auf die Werte *'intensity'* oder *'width'* gesetzt wurde, werden während der Messung ebenfalls Bewertungsmerkmale (*'score'*) berechnet und im Lichtschnitt-Modell gespeichert. `set_sheet_of_light_param` liefert eine detaillierte Beschreibung dieses Modellparameters.

Wenn der Modellparameter *'calibration'* auf *'xz'*, *'xyz'* oder *'offset_scale'* gesetzt wurde, und alle benötigten Parameter für das Modell gesetzt wurden, wird die Kalibriertransformation automatisch auf die Disparitätswerte angewendet. `set_sheet_of_light_param` liefert eine detaillierte Beschreibung der Modellparameter, die für den Einsatz der Kalibriertansformationen benötigt werden.

Setzen von MovementPose

`MovementPose` beschreibt die Bewegung des Objekts zwischen der vorherigen Profilaufnahme und der Aufnahme des aktuellen Profils.

Wenn der Modellparameter *'calibration'* auf *'none'* oder *'xz'* gesetzt wurde (siehe `set_sheet_of_light_param`), wird die Bewegung des Objekts in der Kalibriertransformation nicht

berücksichtigt. MovementPose wird in diesem Fall ignoriert und kann zum Beispiel mit einem leeren Tupel gesetzt werden.

Wenn *'calibration'* auf *'xyz'* gesetzt wurde, muss diese Pose im Lichtschnitt-Modell vorhanden sein. Sie kann hier in MovementPose oder auch mit Hilfe des Parameters *'movement_pose'* im Operator set_sheet_of_light_param gesetzt werden.

Wenn *'calibration'* auf *'offset_scale'* gesetzt wurde, kann hier in MovementPose eine Bewegung angegeben werden. Es sollte allerdings bedacht werden, dass die Bedeutung der Bewegung nicht zwingend metrisch ist.

Wenn die Bewegung des Objektes während der Aufnahme von zwei sukzessiven Profilen konstant ist, wird empfohlen MovementPose hier mit einem leeren Tupel zu setzen, und die konstante Pose der Bewegung mit dem Operator set_sheet_of_light_param zu setzen. Diese Konfiguration ergibt sich zum Beispiel, wenn das zu messende Objekt auf einem Fließband befördert wird und das Messsystem ortsfest ist.

Wenn die Bewegung des Objektes während der Aufnahme von zwei sukzessiven Profilen nicht konstant ist, zum Beispiel weil das Messsystem mit einem Roboter über das zu messende Objekt gefahren wird, muss MovementPose bei jedem Aufruf von measure_profile_sheet_of_light gesetzt werden.

Bitte beachten Sie, dass MovementPose im Weltkoordinatensystem formuliert sein muss, das implizit durch die Pose der Kamera definiert wurde (siehe den Parameter *'camera_pose'* in set_sheet_of_light_param).

Parameter

- ▷ **ProfileImage** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **SheetOfLightModelID** (input_control) ..... sheet_of_light_model ⇝ *handle*
  Handle des Lichtschnitt-Modells.
- ▷ **MovementPose** (input_control) ..... number-array ⇝ *integer* / real
  Pose, welche die Bewegung der zu messenden Szene zwischen dem vorherigen und dem aktuellen Eingabebild beschreibt.

Ergebnis

measure_profile_sheet_of_light liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SheetOfLightModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Nachfolger

apply_sheet_of_light_calibration, get_sheet_of_light_result

Siehe auch

query_sheet_of_light_params, get_sheet_of_light_param, get_sheet_of_light_result, apply_sheet_of_light_calibration

Modul

3D Metrology

query_sheet_of_light_params (: : SheetOfLightModelID, QueryName : GenParamName)

Liefert für das übergebene Lichtschnitt-Modell die Namen der generischen Parameter bzw. Objekte, die in den anderen Lichtschnitt-Operatoren gesetzt oder abgefragt werden können.

Der Operator query_sheet_of_light_params ermöglicht es, die Namen der generischen Parameter abzurufen, die in den Operatoren create_sheet_of_light_model, set_sheet_of_light_param, get_sheet_of_light_param und get_sheet_of_light_result verwendet werden können. Die Auswahl der **Parametergruppe** erfolgt über den Parameter QueryName.

'create_model_params': create_sheet_of_light_model Parameter für das Erstellen des Lichtschnitt-Modells.

'set_model_params': set_sheet_of_light_param Parameter für die Einstellung des Lichtschnitt-Modells.

'get_model_params': get_sheet_of_light_param Parameter für die Abfrage von Parametern des Lichtschnitt-Modells.

'get_result_objects': get_sheet_of_light_result Parameter für den Zugriff auf die ikonischen Ergebnisse einer Messung.

Die erstellte Parameterliste hängt nicht vom gegenwärtigen Bearbeitungszustand des Modells oder von einzelnen Ergebnissen ab.

Parameter

▷ **SheetOfLightModelID** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des Lichtschnitt-Modells.

▷ **QueryName** (input_control) . . . . . attribute.name ⇝ *string*
Name der Parametergruppe.
Defaultwert: 'create_model_params'
Werteliste: QueryName ∈ {'create_model_params', 'set_model_params', 'get_model_params', 'get_result_objects'}

▷ **GenParamName** (output_control) . . . . . attribute.value-array ⇝ *string*
Liste mit den Namen der jeweils gültigen generischen Parameter.

Ergebnis

query_sheet_of_light_params liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

create_sheet_of_light_model, set_sheet_of_light_param, get_sheet_of_light_param, get_sheet_of_light_result

Modul

3D Metrology

read_sheet_of_light_model (: : FileName : SheetOfLightModelID)

Liest ein Lichtschnitt-Modell aus einer Datei und legt ein neues Modell an.

Der Operator read_sheet_of_light_model liest aus der Datei FileName ein Lichtschnitt-Modell ein und erzeugt aus den eingelesenen Daten ein neues Modell. In SheetOfLightModelID wird ein Handle auf das neue Modell zurückgeliefert. Die Datei FileName wird mit dem Operator write_sheet_of_light_model erzeugt. Die Dateiendung für das Lichtschnitt-Modell ist in HALCON 'solm'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der einzulesenden Datei.
Defaultwert: 'sheet_of_light_model.solm'
Dateiendung: .solm

▷ **SheetOfLightModelID** (output_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle für das neu eingelesene Lichtschnitt-Modell.

Ergebnis

`read_sheet_of_light_model` liefert den Wert TRUE, falls eine Datei mit dem angegebenen Namen gefunden wurde und korrekt eingelesen werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`measure_profile_sheet_of_light`

Alternativen

`create_sheet_of_light_model`

Siehe auch

`write_sheet_of_light_model`

Modul

3D Metrology

reset_sheet_of_light_model (: : SheetOfLightModelID :)

Zurücksetzen eines Lichtschnitt-Modells.

Der Operator `reset_sheet_of_light_model` setzt alle internen Indizes und Ergebnisfelder für ein mit `create_sheet_of_light_model` angelegtes Lichtschnitt-Modell zurück. Alle Parametereinstellungen bleiben erhalten. Das Handle des Lichtschnitt-Modells muss in `SheetOfLightModelID` übergeben werden.

Parameter

▷ **SheetOfLightModelID** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des Lichtschnitt-Modells.

Ergebnis

`reset_sheet_of_light_model` liefert den Wert TRUE, falls das übergebene Handle des Lichtschnitt-Modells gültig ist, und das darüber referenzierte Modell vollständig zurückgesetzt werden kann. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SheetOfLightModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

`clear_sheet_of_light_model`

Modul

3D Metrology

serialize_sheet_of_light_model (
: : SheetOfLightModelID : SerializedItemHandle)

Serialisieren eines Lichtschnitt-Modells.

serialize_sheet_of_light_model serialisiert die Daten eines Lichtschnitt-Modells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_sheet_of_light_model in eine Datei geschrieben werden. Das Lichtschnitt-Modell wird in dem Handle SheetOfLightModelID übergeben. Das serialisierte Modell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_sheet_of_light_model wieder deserialisiert werden.

Parameter

▷ **SheetOfLightModelID** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des zu serialisierenden Lichtschnitt-Modells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_sheet_of_light_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_sheet_of_light_model, set_sheet_of_light_param

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_sheet_of_light_model

Siehe auch

deserialize_sheet_of_light_model

Modul

3D Metrology

set_profile_sheet_of_light (
ProfileDisparityImage : : SheetOfLightModelID, MovementPoses :)

Setzen eines Disparitätsbild als neues Profil.

set_profile_sheet_of_light fügt dem Lichtschnitt-Modell SheetOfLightModelID neue Profile hinzu, die als Zeilen im Disparitätsbild ProfileDisparityImage vorliegen. Für jedes Profil kann eine eigene Pose in MovementPoses angegeben werden. Die Pose gilt dann als relative Bewegung zur vorherigen Zeile. Sollte keine Pose angegeben werden, wird die Standardbewegung angewandt, die mit set_sheet_of_light_param gesetzt werden kann. Wenn genau eine Pose angegeben wird, so wird diese die neue Standardbewegung.

Parameter

▷ **ProfileDisparityImage** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2 / real
Disparitätsbild, das ein oder mehrere Profile enthält.

▷ **SheetOfLightModelID** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des Lichtschnitt-Modells.

▷ **MovementPoses** (input_control) . . . . . number-array ⇝ *integer* / real
Posen, welche die Bewegungen der zu messenden Szene zwischen dem vorherigen und dem momentanen Eingabebild beschreibt.

Ergebnis

`set_profile_sheet_of_light` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SheetOfLightModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Nachfolger

`get_sheet_of_light_result`, `get_sheet_of_light_result_object_model_3d`

Siehe auch

`query_sheet_of_light_params`, `get_sheet_of_light_param`, `get_sheet_of_light_result`, `apply_sheet_of_light_calibration`

Modul

3D Metrology

set_sheet_of_light_param (: : SheetOfLightModelID, GenParamName, GenParamValue :)

Setzen ausgewählter Parameter des Lichtschnitt-Modells.

Der Operator `set_sheet_of_light_param` erlaubt es, ausgewählte Parameter des Lichtschnitt-Modells zu setzen, um dieses an die gestellte Messaufgabe anzupassen. Alle Modellparameter, außer die internen Kameraparameter *'camera_parameter'* und die folgenden Poses *'camera_pose'*, *'lightplane_pose'* und *'movement_pose'* können ebenfalls mit Hilfe des Operators `create_sheet_of_light_model` gesetzt werden. Die aktuelle Konfiguration kann mit Hilfe von `get_sheet_of_light_param` ausgelesen werden.

Es folgt eine Übersicht über die verschiedenen Modellparameter mit den jeweiligen Wertebereichen und Standardwerten:

Messung der Profile:

'method': definiert die Methode anhand welcher die Position des Profils im Bild (Disparität) bestimmt wird. Die Werte *'default'* und *'center_of_gravity'* verweisen beide auf dieselbe Methode, bei welcher die Disparität spaltenweise und mit Subpixelgenauigkeit ermittelt wird, indem der Schwerpunkt der Grauwerte g_i berechnet wird, die die folgende Bedingung erfüllen:

$$g_i \geq 'min_gray'$$

'min_gray': der kleinste Grauwert, der für die Berechnung der Position des Profils berücksichtigt wird (vgl. *'method'* oben).

Wertevorschläge: 20, 50, 100, 128, 200, 220, 250

Default: 100

'num_profiles': Anzahl der Profile für welche Speicher im Lichtschnitt-Modell reserviert wurde. Standardmäßig wird *'num_profiles'* auf 512 gesetzt. Wird diese Anzahl an Profilen während einer Messung überschritten, so wird zusätzlicher Speicher automatisch reserviert. Da dieser Prozess jedoch etwas Zeit in Anspruch nimmt, wird empfohlen *'num_profiles'* vor dem Start der Messung auf einen geeigneten Wert zu setzen.

Wertevorschläge: 1, 2, 50, 100, 512, 1024, 3000

Default: 512

'ambiguity_solving': Dieser Parameter legt fest, welcher gemessene Wert zurückgeliefert wird, wenn die Bestimmung der Position des Profils nicht eindeutig ist.

'first': der erste gemessene Wert wird zurückgeliefert. Diese Methode ist die schnellste.

'last': der letzte gemessene Wert wird zurückgeliefert.

'brightest': die Helligkeit aller potentiellen Disparitätswerte wird berechnet, und die Disparität mit der höchsten Helligkeit wird zurückgeliefert. Die Helligkeit wird wie folgt berechnet:

$$brightness = \frac{1}{n}\sum_{i=0}^{n} g_i \ ,$$

wobei g_i die Grauwerte und n die Anzahl der Pixel darstellen, die zur Berechnung der Disparität berücksichtigt werden.

Default: 'first'

'score_type': Dieser Parameter legt fest, ob Bewertungsmerkmale (*'score'*) des Disparitätsbildes berechnet werden sollen, und wenn ja, nach welchem Kriterium.

'none': es werden keine Bewertungsmerkmale berechnet.

'width': für jedes Pixel des Disparitätsbildes wird die lokale Breite des Profils als Bewertungsmerkmal zurückgeliefert.

'intensity': für jedes Pixel des Disparitätsbildes wird als Bewertungsmerkmal die lokale Intensität des Profils zurückgeliefert. Diese wird wie folgt berechnet:

$$score = \frac{1}{n}\sum_{i=0}^{n} g_i$$

wobei g_i die Grauwerte und n die Anzahl der Pixel darstellen, die zur Berechnung der Disparität berücksichtigt werden.

Default: 'none'

Kalibrierung der Messung:

'calibration': gibt an, ob und wenn ja welche Kalibriertransformationen auf das Disparitätsbild angewendet werden sollen:

'none': es werden keine Kalibriertransformationen angewendet.

'xz': es werden Kalibriertransformationen angewendet, die die geometrischen Eigenschaften des Messsystems (Kamera und Lichtlinienprojektor) berücksichtigen. Die Bewegung des Objekts während der Messung wird jedoch nicht berücksichtigt.

'xyz': es werden Kalibriertransformationen angewendet, die sowohl die geometrischen Eigenschaften des Messsystems (Kamera und Lichtlinienprojektor) als auch die Bewegung des Objekts während der Messung berücksichtigen.

'offset_scale': Ein vereinfachter Parametersatz um einen Aufbau manuell zu beschreiben. Es kann bereits mit Standardeinstellungen angewandt werden, bietet aber auch die Möglichkeit den Setup mit nur 6 Parameters zu konfigurieren. Drei der Parameter beschreiben eine anisotrope Skalierung: (*'scale_x'*) beschreibt die Skalierung entlang der Spaltenrichtung des Bildes in die neue x-Achse, (*'scale_y'*) beschreibt eine lineare Bewegung des Objekts zwischen zwei aufeinander folgenden Bildern und *'scale_z'* beschreibt das Verhältnis der Disparität zur neuen z-Achse. Die übrigen drei Parameter beschreiben die Lage des neuen Zentrums relativ zum neuen Weltkoodinatensystem (*'offset_x'*, *'offset_y'*, *'offset_z'*).

Default: 'none'

'camera_parameter': die internen Parameter der Kamera, die zur Durchführung der Messung benutzt wurden. Diese Parameter werden benötigt, wenn *'calibration'* auf *'xz'* oder *'xyz'* gesetzt wurde. Soll mit `calibrate_sheet_of_light` kalibriert werden, wird dieser Parameter benutzt, um die Startparameter der Kamera zu setzen.

'calibration_object': die Beschreibungsdatei des 3D-Kalibrierkörpers für die Kalibrierung mit `calibrate_sheet_of_light`. Soll mit `calibrate_sheet_of_light` kalibriert werden, so muss hier der Dateinamen des mit `create_sheet_of_light_calib_object` erzeugten Kalibrierkörpers gesetzt werden.

'camera_pose': die Pose die das Kamerakoordinatensystem in das Weltkoordinatensystem transformiert, also die Pose, mit der Punktkoordinaten vom Weltkoordinatensystem in das Kamerakoordinatensystem transformiert werden können. Diese Pose wird benötigt, wenn *'calibration'* auf *'xz'* oder *'xyz'* gesetzt wurde.

Es ist zu beachten, dass das Weltkoordinatensystem implizit durch das Setzen von *'camera_pose'* festgelegt wird.

'lightplane_pose': die Pose die das Koordinatensystem der Lichtebene in das Weltkoordinatensystem transformiert, also die Pose, mit der Punktkoordinaten vom Weltkoordinatensystem in das Koordinatensystem der Lichtebene transformiert werden können. Mit dem Koordinatensystem der Lichtebene ist ein Koordinatensystem gemeint, dessen Ebene definiert durch z=0 mit der Lichtebene übereinstimmt, die der Lichtlinienprojektor im Raum erzeugt. Diese Pose wird benötigt, wenn *'calibration'* auf *'xz'* oder *'xyz'* gesetzt wurde.

'movement_pose': die Pose, die die Bewegung des Objekts relativ zum Messsystem während der Messung beschreibt. Diese Pose soll im Weltkoordinatensystem formuliert sein und wird benötigt, wenn *'calibration'* auf *'xyz'* gesetzt wurde.

'scale': dieser Parameter legt fest mit welchem Faktor die kalibrierten 3D-Koordinaten X, Y und Z skaliert werden. *'scale'* muss als der Quotient *Zieleinheit/Originaleinheit* eingegeben werden. Die Originaleinheit wird durch das Kalibrierobjekt bestimmt, und ist Meter, wenn Sie eine Standardkalibrierplatte benutzen. Der Standardwert des Modellparameters *'scale'* ist 1.0. Wenn die Originaleinheit Meter ist, kann *'scale'* direkt mit der gewünschten Zieleinheit gesetzt werden, zum Beispiel mit *'m'*,*'cm'*,*'mm'*, *'microns'*, oder *'um'*. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'*, *'xz'* oder *'xyz'* relevant.

Wertevorschläge: 'm','cm','mm', 'microns','um','1.0',0.01, 0.001, 1.0e-6

Default: 1.0

'scale_x': Dieser Wert definiert die Breite eines Pixels im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Wertevorschläge: '1.0',0.01, 10.0, 1.0e-6

Der Standardwert für *'scale_x'* ist *1.0*.

'scale_y': Dieser Wert definiert die lineare Bewegung zwischen zwei Profilen. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Wertevorschläge: 1.0,0.01, 10.0, 1.0e-6

Der Standardwert für *'scale_y'* ist *10.0*.

'scale_z': Dieser Wert definiert die Höher von Disparitäten im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Wertevorschläge: 1.0,0.01, 10.0, 1.0e-6

Der Standardwert für *'scale_z'* ist *1.0*.

'offset_x': Dieser Wert beschreibt die Verschiebung in x im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Wertevorschläge: 1.0,0.0, -1.0, 1.0e-6

Der Standardwert für *'offset_x'* ist *0.0*.

'offset_y': Dieser Wert beschreibt die Verschiebung in y im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Wertevorschläge: 1.0,0.0, -1.0, 1.0e-6

Der Standardwert für *'offset_y'* ist *0.0*.

'offset_z': Dieser Wert beschreibt die Verschiebung in z im 3D Raum. Dieser Wert ist nur im Falle der Kalibriertransformationen *'offset_scale'* relevant.

Wertevorschläge: 1.0,0.0, -1.0, 1.0e-6

Der Standardwert für *'offset_z'* ist *0.0*.

Parameter

▷ **SheetOfLightModelID** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des Lichtschnitt-Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
Name des einzustellenden Modellparameters.
Defaultwert: 'method'
Werteliste: GenParamName ∈ {'method', 'ambiguity_solving', 'score_type', 'num_profiles', 'min_gray', 'scale', 'calibration', 'calibration_object', 'camera_parameter', 'camera_pose', 'lightplane_pose', 'movement_pose', 'scale_x', 'scale_y', 'scale_z', 'offset_x', 'offset_y', 'offset_z'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Wert des einzustellenden Modellparameters.
Defaultwert: 'center_of_gravity'
Wertevorschläge: GenParamValue ∈ {'default', 'center_of_gravity', 'last', 'first', 'brightest', 'none', 'intensity', 'width', 'xz', 'xyz', 'offset_scale', 'm', 'cm', 'mm', 'um', 'microns', 1.0, 1e-2, 1e-3, 1e-6}

Ergebnis

set_sheet_of_light_param liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SheetOfLightModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Nachfolger

get_sheet_of_light_param, measure_profile_sheet_of_light, apply_sheet_of_light_calibration

Alternativen

create_sheet_of_light_model

Siehe auch

query_sheet_of_light_params, get_sheet_of_light_param, get_sheet_of_light_result

Modul

3D Metrology

write_sheet_of_light_model (: : SheetOfLightModelID, FileName :)

Schreibt ein Lichtschnitt-Modell in eine Datei.

Der Operator write_sheet_of_light_model schreibt ein Lichtschnitt-Modell in eine Datei mit dem Namen FileName. Das so gesicherte Modell kann mit read_sheet_of_light_model später wieder eingelesen und wiederverwendet werden. Gespeichert werden alle generischen Parameter des Modells (siehe set_sheet_of_light_param) und die Ergebnisse von calibrate_sheet_of_light.

Die Dateiendung für das Lichtschnitt-Modell ist in HALCON 'solm'. Der Handle des zu schreibenden Lichtschnitt-Modell ist in SheetOfLightModelID zu übergeben.

Parameter

▷ **SheetOfLightModelID** (input_control) . . . . . sheet_of_light_model ⇝ *handle*
Handle des zu schreibenden Lichtschnitt-Modell.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der zu schreibenden Datei.
Defaultwert: 'sheet_of_light_model.solm'
Dateiendung: .solm

Ergebnis

write_sheet_of_light_model liefert den Wert TRUE, falls der übergebene Handle gültig ist und das Modell in die Datei mit dem angegebenen Namen geschrieben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_sheet_of_light_model`, `set_sheet_of_light_param`

Siehe auch

`read_sheet_of_light_model`

Modul

3D Metrology

Kapitel 6

Bild

Dieses Kapitel beschreibt Operatoren zum Umgang mit verschiedenen Bildtypen.

Um die verschiedenen Bildtypen, die mit HALCON verarbeitet werden können, zu verstehen, werden in den folgenden Abschnitten die drei Komponenten eines Bildes (Pixel, Kanäle, und Domäne) erklärt.

Pixel

In HALCON können Pixel genutzt werden um unterschiedliche Arten von Informationen zu repräsentieren. Daher werden deren Bedeutung entsprechend verschiedene Pixeltypen unterschieden. Die folgende Tabelle listet diese verschiedenen Pixeltypen und deren zugehörigen Standard-Bildtypen. Zu beachten ist, dass diese Auflistung nicht exklusiv ist (beispielsweise können Grauwertbilder viele weitere Bildtypen aufweisen). Bildtypen können mit `convert_image_type` konvertiert werden.

Pixeltyp	Standard-Bildtyp
Grauwerte	`byte, uint2`
Differenz	`int1, int2`
2D-Histogramm	`int4`
Kantenrichtungen	`direction`
Ableitungen	`real`
Fouriertransformation	`complex`
Hue-Werte	`cyclic`
Vektorfeld	`vector_field`

Es ist zu beachten, dass der Bildtyp `vector_field` zusätzlich als `vector_field_absolute` oder `vector_field_relative` spezifiziert werden kann. Zudem existiert auch der Bildtyp `int8` (64 Bit mit Vorzeichen), welcher nur auf 64-Bit-Systemen verfügbar ist. Weitere Informationen zu den einzelnen Pixeltypen werden im Folgenden gegeben.

Grauwerte Grauwertbilder sind standardmäßig vom Typ `byte` (8 Bit, ohne Vorzeichen) oder `uint2` (16 Bit, ohne Vorzeichen) und werden meist zur Darstellung lokaler Lichtintensitäten auf einem Sensor genutzt.



Exemplarisches Grauwertbild.

Differenzbild Um die Unterschiede zweier Bilder zu zeigen kann pixelweise deren Differenz gebildet werden, wofür sich beispielsweise die Bildtypen `int1` (8 Bit, mit Vorzeichen) oder `int2` (16 Bit, mit Vorzeichen) eignen.



(1) (2) (3)

Vergleich eines Bildes (1) mit einer anderen (z.B. darauffolgend aufgenommenen) Aufnahme (2) durch Subtraktion Letzterer von der Ersteren. Als Ergebnis erhält man ein Differenzbild (3).

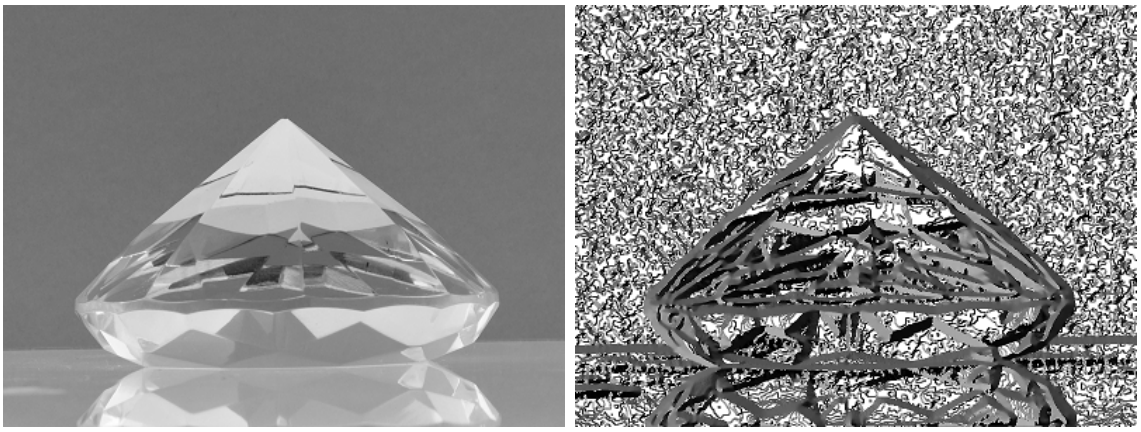
2D-Histogramm Um Bildmerkmale basierend auf dem Vorkommen von Grauwerten in zwei Bildern zu untersuchen kann ein 2D-Histogramm genutzt werden, welches vom Bildtyp `int4` (32 Bit, mit Vorzeichen) ist. Dabei bilden die beiden Achsen (Zeile und Spalte des Histogramms) jeweils die Grauwerte eines Eingabebildes ab. Die entsprechenden Pixel eines Eingabebildes werden dementsprechend in dem 2D-Histogramm eingetragen. Umso häufiger eine bestimmte Kombination an Grauwerten in den Bildern auftritt, desto höher der entsprechende Grauwert im Ausgabebild (siehe auch `histo_2dim`).



(1) (2) (3)

Zwei Kanäle eines Beispielbildes ((1), (2)) und deren zugehöriges 2D-Histogramm (3).

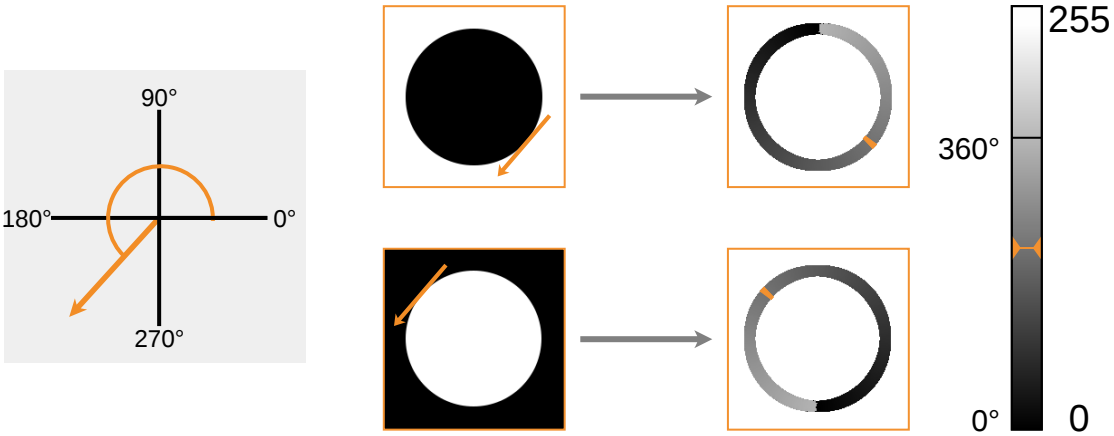
Kantenrichtungen Um die Orientierung der Bildkanten zu visualisieren steht der Bildtyp `direction` (8 Bit, ohne Vorzeichen) zur Verfügung.



(1) (2)

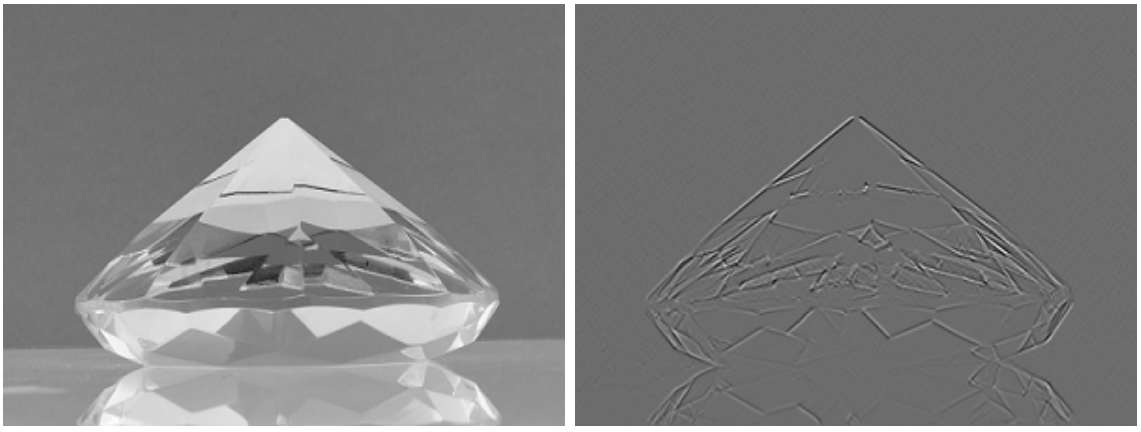
Untersuchung von Bildmerkmalen (1) durch die Visualisierung der Kantenrichtungen (2).

In Bildern des Typs `direction` wird eine Kantenrichtung von x Grad in mathematisch positivem Sinn in Bezug auf die horizontale Achse als x / 2 (also im Bereich von 0 bis 179) im Ergebnisbild abgespeichert. Pixel mit einer Kantenamplitude von 0 wird der Wert 255 (undefinierte Richtung) zugeordnet.



Visualisierung der Kantenrichtungen dunkler und heller Objekte mittels `sobel_dir`: Die Kantenrichtung (im Eingabebild mit tangentialem Pfeil markiert) eines Pixels wird im rechts Ergebnisbild mit einem entsprechenden Grauwert abgebildet. Stellen ohne Kanten werden mit einem Grauwert von 255 markiert.

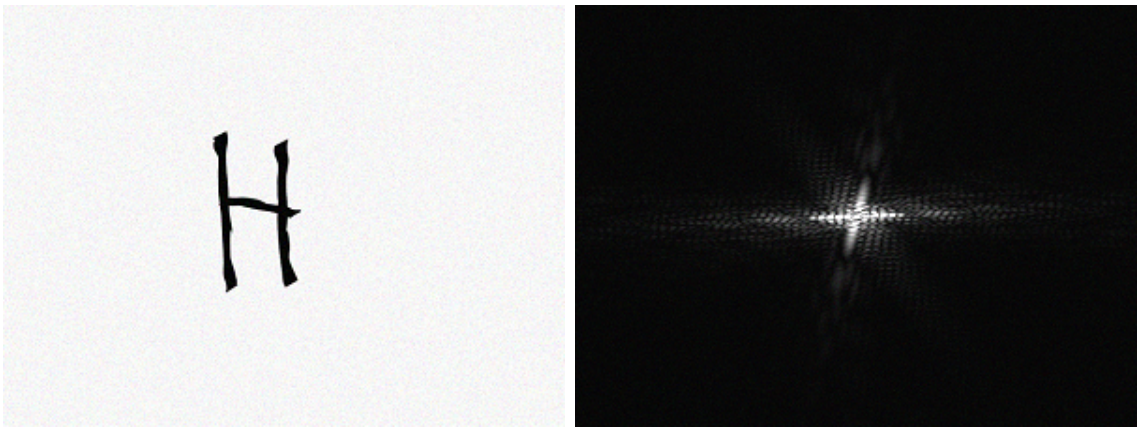
Ableitungen Der Bildtyp `real` (32 Bit, Fließkommawerte) wird für Ableitungsbilder genutzt mit deren Hilfe beispielsweise Kanten extrahiert werden.



(1) (2)

Manche Merkmale innerhalb eines Bildes (1) (z.B. Kanten, Gradienten) können durch das Bilden ihrer Ableitungen (2) untersucht werden.

Fouriertransformation Um den Frequenzbereich eines Bildes zu untersuchen kann die Fouriertransformation genutzt werden. Die Kombination aus Magnitude und Phase der einzelnen Frequenzen wird durch komplexe Zahlen repräsentiert, wofür der Bildtyp `complex` (zwei `real`-Werte pro Pixel) verwendet wird.



(1) (2)

Eine mögliche Vorgehensweise zur Identifikation von Buchstaben (1) ist die Betrachtung des Frequenzbereichs mit Hilfe der Fouriertransformation (2).

Hue-Werte Der Bildtyp `cyclic` (8 Bit, ohne Vorzeichen) wird für Hue-Werte (Farbwerte) genutzt, sodass der Wertebereich zyklisch ist. Dementsprechend gilt hier $255 + 1 = 0$.



(1) (2)

Die Pixel repräsentieren Hue-Werte. Wird ein Pixelwert über 255 erhöht, so wird er an den Anfang des Grauwertbereichs zurückverschoben. Addiert man zu den Pixelwerten eines Bildes des Bildtyps `cyclic` (1) eine Konstante, ergibt sich ein Bild wie in (2) zu sehen.

Vektorfeld Ein spezieller Bildtyp zur Darstellung absoluten/relativen optischen Flusses ist `vector_field` (zusammengesetzt aus zwei Bildern des Bildtyps `real` für die x- und y-Richtung). Der Bildtyp kann weiter spezifiziert werden indem man ihn auf `vector_field_absolute` (als absolute Koordinaten interpretiert) beziehungsweise `vector_field_relative` (als Vektoren interpretiert) setzt.



(1) (2) (3)

Die relative Bewegung von Pixeln (beispielsweise in aufeinander folgenden Bildern (1) und (2) können durch ein Vektorfeld dargestellt werden (3).

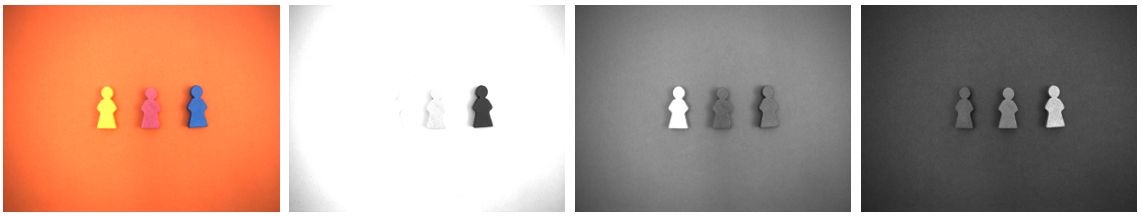
Kanäle

Neben verschiedenen Pixeltypen können in einem Bild auch verschiedene Kanäle genutzt werden um spezifische Informationen zu speichern. Prinzipiell können Kanäle genutzt werden um einem Pixel mehrere Werte zuordnen zu können. In den nächsten Absätzen werden einige typische Fälle zur Nutzung von Bildkanälen kurz erläutert.

Grauwertbilder (Einkanalige Bilder) Häufig wird in einem Bild nur ein Wert pro Pixel gespeichert. In einem Grauwertbild beispielsweise, ist pro Pixel ein lokaler Intensitätswert von Licht auf einem Sensor registriert. Nichtsdestotrotz können viele Arten von Informationen in Pixeln gespeichert werden (z.B. Grauwertableitungen, Distanzen). Bei der Interpretation dieser Daten kann es aber häufig von Vorteil sein diese wie Grauwertbilder zu visualisieren.

RGB-Bild (Dreikanalige Bilder) Für Farbbilder werden Pixelwerte typischerweise in drei Kanälen gespeichert. Diese repräsentieren jeweils die Intensitäten für rotes, grünes oder blaues (RGB) Licht pro Pixel, wodurch durch additives Mischen eine Farbe für jedes Pixel definiert wird.

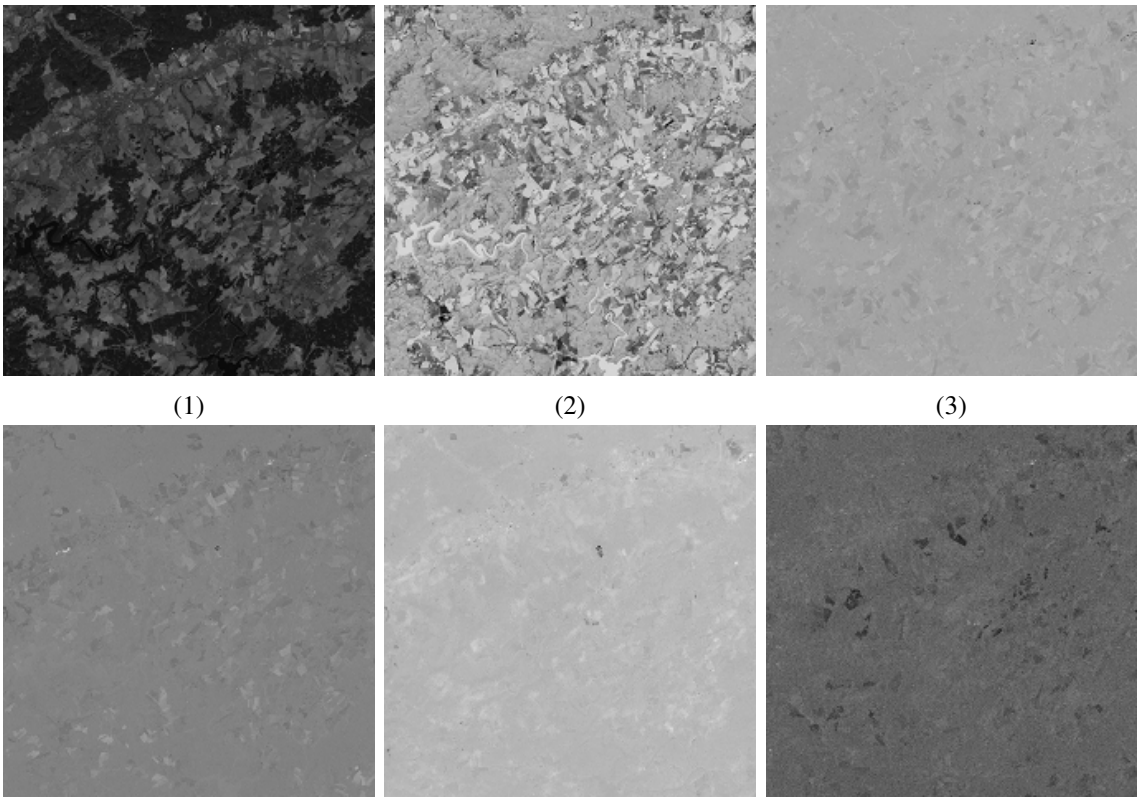
Neben dem RGB-Modell existieren einige weitere Farbmodelle. Weitere Informationen zu diesen Modellen und dem Transformieren zwischen den verschiedenen Farbräumen finden sich in `trans_to_rgb` beziehungsweise `trans_from_rgb`.



(1) (2) (3) (4)

RGB-Bild (1) und dessen drei Kanäle im Einzelnen: lokale Intensitäten unterscheiden sich für den roten (2), grünen (3), und blauen Kanal (4).

Multispektralbild (Mehrkanalige Bilder) Mit speziellen Kameras kann eine Vielzahl spektraler Bänder aufgenommen werden, auch außerhalb des sichtbaren Lichtspektrums. Beispielsweise nehmen satellitengestützte Kameras oft multispektrale Daten auf, welche sie dementsprechend in einer Vielzahl von Kanälen abspeichern.



(1) (2) (3)

(4) (5) (6)

Sechs Kanäle eines Satellitenbildes: die unterschiedlichen Kanäle des multispektralen Bildes können genutzt werden um Merkmale zu extrahieren, die besonders in einem speziellen spektralen Band prominent auftreten.

Mehrkanalige Bilder Neben der Darstellung von Lichtintensitäten kann die Nutzung mehrerer Kanäle in einem Bild auch verschiedene weitere Zwecke erfüllen:

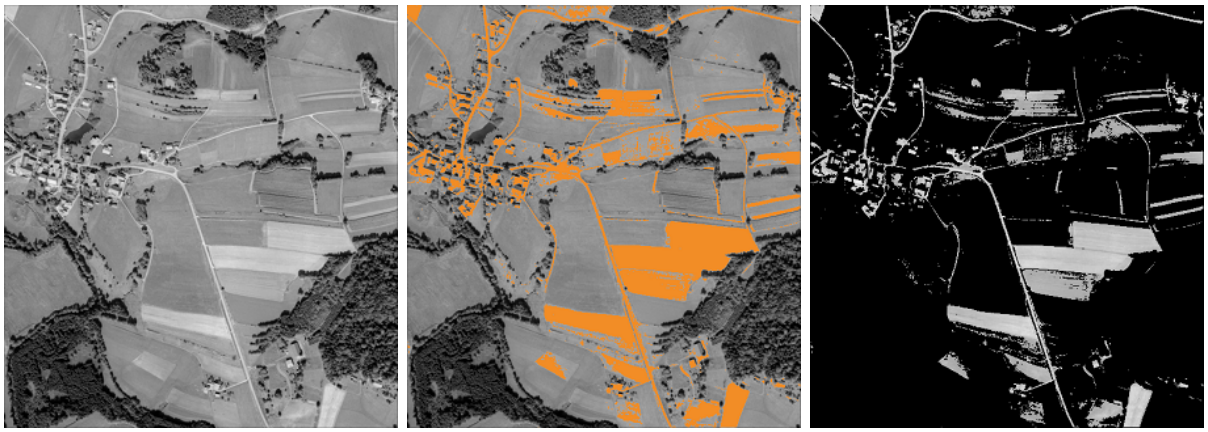
Beispielsweise kann durch die Kombination eines Grauwertbildes mit einem zusätzlichen Kanal, welcher Tiefenwerte enthält, eine Szene als 3D-Plot visualisiert werden.

Statt der Nutzung einer Tiefenkamera zur Aufnahme von Tiefendaten kann in manchen Fällen auch ein mehrkanaliges Bild zur Extraktion von Tiefenwerten genutzt werden. Dazu muss jeder Kanal ein Bild einer Serie mit unterschiedlichen Fokusebenen enthalten.

Abgesehen von der Speicherung räumlicher Informationen einer Szene können die Kanäle eines Bildes auch als Merkmalsraum interpretiert werden um eine Hauptkomponentenanalyse (PCA) durchzuführen.

Domäne

Die Domäne eines Bildes bestimmt die Fläche eines Bildes, die in den folgenden Bearbeitungsschritten berücksichtigt werden soll. Um die Verarbeitung auf die relevanten Bildausschnitte zu reduzieren und/oder die Anwendung zu beschleunigen kann die Domäne eines Bildes entsprechend reduziert werden.



(1) (2) (3)

Die Domäne des Eingabebildes (1) wird reduziert, beispielsweise durch die Anwendung eines Schwellenwerts (2). Bildbereiche, die für weitere Operationen relevant sind (orange), bleiben im Ergebnisbild (3) erhalten, wohingegen überflüssige Bereiche ausgeschlossen werden (schwarz).

6.1 Bildeinzug

close_framegrabber (: : AcqHandle :)

Schließen des angegebenen Bildeinzugsgeräts.

close_framegrabber schließt das durch AcqHandle spezifizierten Bildeinzugsgerät. Dabei wird insbesondere etwaiger Speicherplatz für Datenpuffer freigegeben und das Bildeinzugsgerät wieder für andere Prozesse zugänglich gemacht.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

D.h., info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers verhindern die gleichzeitige Ausführung anderer Operatoren innerhalb der Gruppe, laufen aber mit allen nicht exklusiven Operatoren außerhalb der Gruppe parallel.

Parameter

▷ **AcqHandle** (input_control) . . . . . framegrabber ⇝ *handle*
Handle des zu schließenden Bildeinzugsgeräts.

Ergebnis

close_framegrabber liefert den Wert TRUE, sofern das angegebene Bildeinzugsgerät geschlossen werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- AcqHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

grab_image, grab_image_async

Siehe auch

open_framegrabber

Modul

Foundation

get_framegrabber_callback (: : AcqHandle, CallbackType : CallbackFunction, UserContext)

Abfrage einer Callback-Funktion eines Bildeinzugsgeräts.

Der Operator get_framegrabber_callback liefert eine bestimmte Callback-Funktion für das durch AcqHandle angegebene Bildeinzugsgerät. Falls die Callback-Funktion zuvor via set_framegrabber_callback registriert wurde, enthält CallbackFunction einen Zeiger darauf, andernfalls NULL. Weiterhin wird in UserContext auch der Zeiger auf die benutzerspezifischen Kontextdaten zurückgegeben, der beim Aufruf von set_framegrabber_callback angegeben wurde (oder NULL).

Mit dem Parameter CallbackType können verschiedene Callback-Typen abgefragt werden. Typische Werte sind:

'exception': Der Bildeinzug hat eine Fehlerbehandlung ausgelöst.

'exposure_end': Die Belichtung des nächsten Bildes ist beendet.

'exposure_start': Die Belichtung des nächsten Bildes ist gestartet.

'transfer_end': Ein neues Bild ist verfügbar und kann mit grab_image_async abgeholt werden.

Abhängig von der Funktionalität des darunterliegenden API sind auch weitere Werte für CallbackType möglich. Alle tatsächlich verfügbaren Callback-Typen für ein bestimmtes Bildeinzugsgerät können durch den Aufruf von get_framegrabber_param mit dem Parameter 'available_callback_types' abgefragt werden. Nähere Informationen finden sich in der Dokumentation der verwendeten Bildeinzugsschnittstelle.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

get_framegrabber_callback läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

- ▷ **AcqHandle** (input_control) ..... framegrabber ⇝ *handle*
  Zu benutzendes Bildeinzugsgerät.
- ▷ **CallbackType** (input_control) ..... string ⇝ *string*
  Callback-Typ.
  Defaultwert: 'transfer_end'
  Wertevorschläge: CallbackType ∈ {'exception', 'exposure_end', 'exposure_start', 'transfer_end'}
- ▷ **CallbackFunction** (output_control) ..... pointer ⇝ *integer*
  Zeiger auf die Callback-Funktion.
- ▷ **UserContext** (output_control) ..... pointer ⇝ *integer*
  Zeiger auf benutzerspezifische Kontextdaten.

Ergebnis

Ist das Bildeinzugsgerät geöffnet und wird der angegebene Callback-Typ unterstützt, liefert get_framegrabber_callback den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_framegrabber, set_framegrabber_callback

Nachfolger

grab_image, grab_data, grab_image_start, grab_image_async, grab_data_async, set_framegrabber_param, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, set_framegrabber_callback

Modul

Foundation

get_framegrabber_lut (: : AcqHandle : ImageRed, ImageGreen, ImageBlue)

Abfrage der Look-Up-Tabelle des Bildeinzugsgeräts.

get_framegrabber_lut fragt die Look-Up-Tabelle (LUT) des mit AcqHandle spezifizierten Bildeinzugsgeräts ab. Diese Operation wird nicht von allen Bildeinzugsgeräten unterstützt.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

get_framegrabber_lut läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

- ▷ **AcqHandle** (input_control) ..... framegrabber ⇝ *handle*
  Zu benutzendes Bildeinzugsgerät.
- ▷ **ImageRed** (output_control) ..... integer-array ⇝ *integer*
  Rotanteil der LUT-Einträge.
- ▷ **ImageGreen** (output_control) ..... integer-array ⇝ *integer*
  Grünanteil der LUT-Einträge.
- ▷ **ImageBlue** (output_control) ..... integer-array ⇝ *integer*
  Blauanteil der LUT-Einträge.

Ergebnis

get_framegrabber_lut liefert den Wert TRUE, falls der Bildeinzugsgerät geöffnet ist.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_framegrabber

Nachfolger

set_framegrabber_lut

Siehe auch

set_framegrabber_lut, open_framegrabber

Modul

Foundation

get_framegrabber_param (: : AcqHandle, Param : Value)

Abfrage spezifischer Parameter eines Bildeinzugsgeräts.

get_framegrabber_param liefert spezielle Parameterwerte für das durch AcqHandle angegebene Bildeinzugsgerät. Die unten aufgelisteten Standardparameter sind dabei für jedes beliebige Bildeinzugsgerät abrufbar. Darüberhinaus werden in der Regel noch weitere Parameter von einem spezifischen Bildeinzugsgerät unterstützt. Eine Liste dieser zusätzlichen Parameter ist mit der Abfrage *'parameter'* vermöge info_framegrabber abrufbar.

Standardwerte für Param, vgl. open_framegrabber:

'name' Name der Bildeinzugsschnittstelle.

'horizontal_resolution' Horizontale Auflösung des Bildeinzugsgeräts.

'vertical_resolution' Vertikale Auflösung des Bildeinzugsgeräts.

'image_width' Breite des Bildausschnittes.

'image_height' Höhe des Bildausschnittes.

'start_row' Zeilennummer der oberen linken Ecke des Bildausschnittes.

'start_column' Spaltennummer der oberen linken Ecke des Bildausschnittes.

'field' Selektiertes Halbbild bzw. Vollbild.

'bits_per_channel' Anzahl übertragener Bits pro Pixel und Bildkanal.

'color_space' Farbraum des resultierenden Bildes.

'generic' Generischer Wert mit gerätespezifischer Bedeutung.

'external_trigger' Trigger durch externes Signal (*'true'* / *'false'*).

'camera_type' Verwendeter Kameratyp.

'device' Gerätename des Bildeinzugsgeräts.

'port' Port, an den das Bildeinzugsgerät angeschlossen ist.

'line_in' Kameraeingang (falls Multiplexer verfügbar).

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

get_framegrabber_param läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

▷ **AcqHandle** (input_control) ..... framegrabber ⇝ *handle*
Zu benutzendes Bildeinzugsgerät.

▷ **Param** (input_control) ..... string(-array) ⇝ *string*
Abzufragender Parameter.
Defaultwert: 'revision'
Wertevorschläge: Param ∈ {'bits_per_channel', 'camera_type', 'color_space', 'continuous_grabbing', 'device', 'external_trigger', 'field', 'generic', 'grab_timeout', 'horizontal_resolution', 'image_available', 'image_height', 'image_width', 'line_in', 'name', 'port', 'revision', 'start_column', 'start_row', 'vertical_resolution', 'volatile'}

▷ **Value** (output_control) ..... string(-array) ⇝ *string* / real / integer
Parameterwert.

Ergebnis

Ist das Bildeinzugsgerät geöffnet und wird der angegebene Parameter unterstützt, liefert get_framegrabber_param den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_framegrabber, set_framegrabber_param

Nachfolger

grab_image, grab_data, grab_image_start, grab_image_async, grab_data_async, set_framegrabber_param, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Foundation

grab_data (: Image, Region, Contours : AcqHandle : Data)

Synchroner Einzug von Bildern und vorverarbeiteten Bilddaten vom angegebenen Bildeinzugsgerät.

grab_data zieht über das durch AcqHandle spezifizierte Bildeinzugsgerät Bilder und vorverarbeitete Bilddaten ein. Der gewünschte Betriebsmodus des Bildeinzugsgeräts, ein passender Bildausschnitt sowie weitere Schnittstellen-spezifische Einstellungen können mittels der Operatoren open_framegrabber und set_framegrabber_param vorgenommen werden. Abhängig von der aktuellen Konfiguration des Bildeinzugsgeräts können die vorverarbeiteten Bilddaten in Form von Bildern (Image), Bildregionen (Region), XLD-Konturen (Contours) und Steuerdaten (Data) zurückgeliefert werden.

Der Abbruch des Bildeinzugs kann mit dem Operator set_framegrabber_param erfolgen, sofern die jeweilige Bildeinzugsschnittstelle den dazu notwendigen Parameter 'do_abort_grab' unterstützt. Es ist zu beachten, dass, als Ausnahme von der Beschreibung der gleichzeitigen Verwendung mehrerer Threads (siehe unten), 'do_abort_grab' auch aus einem anderen Thread aufgerufen werden kann.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden die Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

grab_data läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

- ▷ **Image** (output_object) ..... image(-array) ⇝ *object :* byte / real / uint2
  Eingezogene Bilddaten.
- ▷ **Region** (output_object) ..... region(-array) ⇝ *object*
  Vorverarbeitete Bildregionen.
- ▷ **Contours** (output_object) ..... xld_cont(-array) ⇝ *object*
  Vorverarbeitete XLD-Konturen.
- ▷ **AcqHandle** (input_control) ..... framegrabber ⇝ *handle*
  Zu benutzendes Bildeinzugsgerät.
- ▷ **Data** (output_control) ..... string(-array) ⇝ *string* / real / integer / handle
  Vorverarbeitete Steuerdaten.

Beispiel

```
* Select a suitable image acquisition interface name AcqName
info_framegrabber(AcqName,'port',Information,Values)
* Open image acquisition device using the default settings, see
* documentation of the actually used interface for more details
open_framegrabber(AcqName,1,1,0,0,0,0,'default',-1,'default',-1.0, \
                  'default','default','default',-1,-1,AcqHandle)
```

```
* Grab and segment image
grab_data (Image, Region, Contours, AcqHandle, Data)
* Process Region...
close_framegrabber(AcqHandle)
```

Ergebnis

Ist das Bildeinzugsgerät geöffnet und unterstützt es den Bildeinzug via `grab_data`, liefert `grab_data` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_framegrabber`, `grab_image_start`, `set_framegrabber_param`

Nachfolger

`grab_data_async`, `grab_image_start`, `grab_image`, `grab_image_async`, `set_framegrabber_param`, `close_framegrabber`

Siehe auch

`open_framegrabber`, `info_framegrabber`, `set_framegrabber_param`

Modul

Foundation

grab_data_async (: Image, Region, Contours : AcqHandle, MaxDelay : Data)

Asynchroner Einzug von Bildern und vorverarbeiteten Bilddaten vom angegebenen Bildeinzugsgerät.

`grab_data_async` fordert ein asynchron eingezogenes Bild und vorverarbeitete Bilddaten über das durch `AcqHandle` spezifizierte Bildeinzugsgerät an. Standardmäßig startet `grab_data_async` den nächsten asynchronen Einzug, bevor der Operator sich zurückmeldet. Nähere Informationen zu dem Verhalten eines bestimmten Bildeinzugsgeräts befinden sich in der Dokumentation für die verwendete Bildeinzugsschnittstelle im Verzeichnis *'doc/html/reference/acquisition'*. Der gewünschte Betriebsmodus des Bildeinzugsgeräts, ein passender Bildausschnitt sowie weitere Schnittstellen-spezifische Einstellungen können mittels der Operatoren `open_framegrabber` und `set_framegrabber_param` vorgenommen werden. Abhängig von der aktuellen Konfiguration des Bildeinzugsgeräts können die vorverarbeiteten Bilddaten in Form von Bildern (`Image`), Bildregionen (`Region`), XLD-Konturen (`Contours`) und Steuerdaten (`Data`) zurückgeliefert werden.

Der Abschluss des neu gestarteten Bildeinzuges erfolgt mittels `grab_data_async` oder `grab_image_async`. Sind mehr als `MaxDelay` ms seit dem Start des Einzuges vergangen, wird falls nötig ein neues Bild eingezogen. Ein negativer Wert für `MaxDelay` deaktiviert diesen Mechanismus.

Der Abbruch des Bildeinzugs kann mit dem Operator `set_framegrabber_param` erfolgen, sofern die jeweilige Bildeinzugsschnittstelle den dazu notwendigen Parameter `'do_abort_grab'` unterstützt. Es ist zu beachten, dass, als Ausnahme von der Beschreibung der gleichzeitigen Verwendung mehrerer Threads (siehe unten), `'do_abort_grab'` auch aus einem anderen Thread aufgerufen werden kann.

Falls nach `grab_data_async` der Operator `grab_image` oder `grab_data` aufgerufen wird, wird der mit `grab_data_async` begonnene asynchrone Bildeinzug abgebrochen und ein neuer, synchroner Bildeinzug gestartet.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in `open_framegrabber` aufgelistet) eine eigene Gruppe innerhalb der `info_framegrabber`, `open_framegrabber`, `close_framegrabber` und `close_all_framegrabbers` exklusiv ausgeführt werden.

`grab_data_async` läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

- ▷ **Image** (output_object) . . . . . image(-array) ⇝ *object :* byte / real / uint2
  Eingezogene Bilddaten.
- ▷ **Region** (output_object) . . . . . region(-array) ⇝ *object*
  Vorverarbeitete Bildregionen.
- ▷ **Contours** (output_object) . . . . . xld_cont(-array) ⇝ *object*
  Vorverarbeitete XLD-Konturen.
- ▷ **AcqHandle** (input_control) . . . . . framegrabber ⇝ *handle*
  Zu benutzendes Bildeinzugsgerät.
- ▷ **MaxDelay** (input_control) . . . . . number ⇝ *real*
  Maximal erlaubte Verzögerung zwischen Start des Einzugs und Auslesen des Bildes [ms].
  Defaultwert: -1.0
  Wertevorschläge: `MaxDelay` ∈ {-1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9}
- ▷ **Data** (output_control) . . . . . string(-array) ⇝ *string* / real / integer / handle
  Vorverarbeitete Steuerdaten.

Beispiel

```
* Select a suitable image acquisition interface name AcqName
open_framegrabber(AcqName,1,1,0,0,0,0,'default',-1,'default',-1.0, \
                  'default','default','default',-1,-1,AcqHandle)
* Grab image, segment it, and start next grab
grab_data_async (Image1, Region1, Contours1, AcqHandle, -1.0, Data1)
* Process data 1...
* Finish asynchronous grab, segment this image, and start next grab
grab_data_async (Image2, Region2, Contours2, AcqHandle, -1.0, Data2)
* Process data 2...
close_framegrabber(AcqHandle)
```

Ergebnis

Ist das Bildeinzugsgerät geöffnet und unterstützt es den Bildeinzug via `grab_data_async`, liefert `grab_data_async` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_framegrabber`, `grab_image_start`, `set_framegrabber_param`

Nachfolger

`grab_image_async`, `set_framegrabber_param`, `close_framegrabber`

Siehe auch

`open_framegrabber`, `info_framegrabber`, `set_framegrabber_param`

Modul

Foundation

grab_image (: Image : AcqHandle :)

Synchroner Einzug eines Bildes vom angegebenen Bildeinzugsgerät.

`grab_image` zieht über das durch `AcqHandle` spezifizierte Bildeinzugsgerät ein Bild ein. Der gewünschte Betriebsmodus des Bildeinzugsgeräts, ein passender Bildausschnitt sowie weitere Schnittstellen-spezifische Einstellungen können mittels der Operatoren `open_framegrabber` und `set_framegrabber_param` vorgenommen werden.

Der Abbruch des Bildeinzugs kann mit dem Operator set_framegrabber_param erfolgen, sofern die jeweilige Bildeinzugsschnittstelle den dazu notwendigen Parameter 'do_abort_grab' unterstützt. Es ist zu beachten, dass, als Ausnahme von der Beschreibung der gleichzeitigen Verwendung mehrerer Threads (siehe unten), 'do_abort_grab' auch aus einem anderen Thread aufgerufen werden kann.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

grab_image läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

- ▷ **Image** (output_object) ..... image ⇝ *object :* byte / uint2
  Eingezogenes Bild.
- ▷ **AcqHandle** (input_control) ..... framegrabber ⇝ *handle*
  Zu benutzendes Bildeinzugsgerät.

Beispiel

```
* Select a suitable image acquisition interface name AcqName
info_framegrabber(AcqName,'port',Information,Values)
* Open image acquisition device using the default settings, see
* documentation of the actually used interface for more details
open_framegrabber(AcqName,1,1,0,0,0,0,'default',-1,'default',-1.0, \
                  'default','default','default',-1,-1,AcqHandle)
grab_image(Image,AcqHandle)
close_framegrabber(AcqHandle)
```

Ergebnis

Wenn der Bildeinzug erfolgreich verlaufen ist, liefert grab_image den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_framegrabber, set_framegrabber_param

Nachfolger

grab_image_start, grab_image_async, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Foundation

grab_image_async (: Image : AcqHandle, MaxDelay :)

Asynchroner Einzug eines Bildes vom angegebenen Bildeinzugsgerät.

grab_image_async fordert ein asynchron eingezogenes Bild über das durch AcqHandle spezifizierte Bildeinzugsgerät an. Standardmäßig startet grab_image_async den nächsten asynchronen Einzug, bevor der Operator sich zurückmeldet. Nähere Informationen zu dem Verhalten eines bestimmten Bildeinzugsgeräts befinden sich in der Dokumentation für die verwendete Bildeinzugsschnittstelle im Verzeichnis *'doc/html/reference/acquisition'*. Der gewünschte Betriebsmodus des Bildeinzugsgeräts, ein passender Bildausschnitt sowie weitere Schnittstellen-spezifische Einstellungen können mittels der Operatoren open_framegrabber und set_framegrabber_param vorgenommen werden.

Der Abschluss des neu gestarteten Bildeinzuges erfolgt mittels grab_image_async oder grab_data_async. Sind mehr als MaxDelay ms seit dem Start des Einzuges vergangen, wird falls nötig ein neues Bild eingezogen. Ein negativer Wert für MaxDelay deaktiviert diesen Mechanismus.

Der Abbruch des Bildeinzugs kann mit dem Operator set_framegrabber_param erfolgen, sofern die jeweilige Bildeinzugsschnittstelle den dazu notwendigen Parameter 'do_abort_grab' unterstützt. Es ist zu beachten, dass, als Ausnahme von der Beschreibung der gleichzeitigen Verwendung mehrerer Threads (siehe unten), 'do_abort_grab' auch aus einem anderen Thread aufgerufen werden kann.

Falls nach grab_image_async der Operator grab_image oder grab_data aufgerufen wird, wird der mit grab_image_async begonnene asynchrone Bildeinzug abgebrochen und ein neuer, synchroner Bildeinzug gestartet.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

grab_image_async läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

- ▷ **Image** (output_object) ..... image ⇝ *object :* byte / int2
  Eingezogenes Bild.
- ▷ **AcqHandle** (input_control) ..... framegrabber ⇝ *handle*
  Zu benutzendes Bildeinzugsgerät.
- ▷ **MaxDelay** (input_control) ..... number ⇝ *real*
  Maximal erlaubte Verzögerung zwischen Start des Einzugs und Auslesen des Bildes [ms].
  Defaultwert: -1.0
  Wertevorschläge: MaxDelay ∈ {-1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9}

Beispiel

```
* Select a suitable image acquisition interface named AcqName
open_framegrabber('AcqName', 1, 1, 0, 0, 0, 0, 'default' ,-1 ,'default' ,-1.0, \
                  'default', 'default', 'default', -1, -1, AcqHandle)
* Grab image + start next grab
grab_image_async(Image1, AcqHandle, -1.0)
* Process Image1 ...
* Finish asynchronous grab + start next grab
grab_image_async(Image2, AcqHandle, -1.0)
* Process Image2 ...
close_framegrabber(AcqHandle)
```

Ergebnis

Ist das Bildeinzugsgerät geöffnet und unterstützt es den asynchronen Bildeinzug, liefert grab_image_async den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

grab_image_start, open_framegrabber, set_framegrabber_param

Nachfolger

grab_data_async, set_framegrabber_param, close_framegrabber

Siehe auch

grab_image_start, open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Foundation

grab_image_start (: : AcqHandle, MaxDelay :)

Start eines asynchronen Bildeinzugs vom angegebenen Bildeinzugsgerät.

grab_image_start startet den asynchronen Bildeinzug über das durch AcqHandle spezifizierten Bildeinzugsgerät. Der gewünschte Betriebsmodus des Bildeinzugsgeräts sowie ein passender Bildausschnitt kann mittels open_framegrabber eingestellt werden. Weitere Schnittstellen-spezifische Einstellungen können gegebenenfalls mit set_framegrabber_param vorgenommen werden.

Der Abschluss des Bildeinzuges erfolgt dann mittels grab_image_async bzw. grab_data_async. Der Parameter MaxDelay ist obsolet und hat keinen Einfluss auf das neue eingezogene Bild. Um festzustellen, ob ein eingezogenes Bild zu alt ist, kann dagegen der Parameter MaxDelay des Operators grab_image_async bzw. grab_data_async verwendet werden.

Beachten Sie, dass der Operator grab_image_start nur in Verbindung mit grab_image_async oder grab_data_async sinnvoll ist. Falls stattdessen grab_image oder grab_data aufgerufen werden, wird der mit grab_image_start begonnene asynchrone Bildeinzug abgebrochen und ein neuer, synchroner Bildeinzug gestartet.

Der Abbruch des Bildeinzugs kann mit dem Operator set_framegrabber_param erfolgen, sofern die jeweilige Bildeinzugsschnittstelle den dazu notwendigen Parameter 'do_abort_grab' unterstützt. Es ist zu beachten, dass, als Ausnahme von der Beschreibung der gleichzeitigen Verwendung mehrerer Threads (siehe unten), 'do_abort_grab' auch aus einem anderen Thread aufgerufen werden kann.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

grab_image_start läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

▷ **AcqHandle** (input_control) ..... framegrabber ⇝ *handle*
Zu benutzendes Bildeinzugsgerät.

▷ **MaxDelay** (input_control) ..... number ⇝ *real*
Dieser Parameter ist obsolet und hat keinen Einfluss auf das neue eingezogene Bild.
Defaultwert: -1.0
Wertevorschläge: MaxDelay ∈ {-1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9}

Beispiel

```
* Select a suitable image acquisition interface named AcqName.
open_framegrabber('AcqName', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', \
                  -1.0, 'default', 'default', 'default', -1, -1, AcqHandle)
* Start asynchronous grabbing.
grab_image_start(AcqHandle, -1)
* Run acquisition loop.
while (true)
    * Get image, start next grab.
    grab_image_async(Image, AcqHandle, -1.0)
    * Next: Do something with the grabbed image.
endwhile
close_framegrabber(AcqHandle)
```

Ergebnis

Ist das Bildeinzugsgerät geöffnet und unterstützt es den asynchronen Bildeinzug, liefert grab_image_start den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_framegrabber, set_framegrabber_param

Nachfolger

grab_image_async, grab_data_async, set_framegrabber_param, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Foundation

info_framegrabber (: : Name, Query : Information, ValueList)

Ausgabe von Informationen zur angegebenen Bildeinzugsschnittstelle.

info_framegrabber liefert Informationen zur angegebenen Bildeinzugsschnittstelle Name. Über den Parameter Query wird die gewünschte Information ausgewählt. In Information wird dann eine textuelle Beschreibung, in ValueList gegebenenfalls eine Liste von zulässigen Werten zurückgegeben. Derzeit werden folgende Abfragen unterstützt:

'bits_per_channel': Auflistung aller unterstützten Werte für den Parameter BitsPerChannel, vgl. open_framegrabber.

'camera_type': Beschreibung und Auflistung aller unterstützten Werte für den Parameter CameraType, vgl. open_framegrabber.

'color_space': Auflistung aller unterstützten Werte für den Parameter ColorSpace, vgl. open_framegrabber.

'defaults': Schnittstellen-spezifische Defaultwerte in ValueList, vgl. open_framegrabber.

'device': Auflistung aller unterstützten Werte für den Parameter Device, vgl. open_framegrabber.

'external_trigger': Auflistung aller unterstützten Werte für den Parameter ExternalTrigger, vgl. open_framegrabber.

'field': Auflistung aller unterstützten Werte für den Parameter Field, vgl. open_framegrabber.

'general': Allgemeine Angaben in Information.

'generic': Generischer Wert mit gerätespezifischer Bedeutung, vgl. open_framegrabber.

'horizontal_resolution': Auflistung aller unterstützten Werte für den Parameter HorizontalResolution, vgl. open_framegrabber.

'image_height': Auflistung aller unterstützten Werte für den Parameter ImageHeight, vgl. open_framegrabber.

'image_width': Auflistung aller unterstützten Werte für den Parameter ImageWidth, vgl. open_framegrabber.

'info_boards': Informationen über die tatsächlich im Rechner installierten Bildeinzugsgeräte. Diese Daten werden z.B für den Auto-detect-Mechanismus des Image Acquisition Assistant in HDevelop verwendet.

'line_in': Auflistung aller unterstützten Werte für den Parameter LineIn, vgl. open_framegrabber.

'parameters': Auflistung aller Schnittstellen-spezifischen Parameter, die über set_framegrabber_param bzw. get_framegrabber_param zugänglich sind.

'parameters_readonly': Auflistung aller Schnittstellen-spezifischen Parameter, die nur über get_framegrabber_param zugänglich sind.

'parameters_writeonly': Auflistung aller Schnittstellen-spezifischen Parameter, die nur über set_framegrabber_param zugänglich sind.

'port': Auflistung aller unterstützten Werte für den Parameter Port, vgl. open_framegrabber.

'revision': Aktuelle Versionsnummer der Bildeinzugsschnittstelle.

'start_column': Auflistung aller unterstützten Werte für den Parameter `StartColumn`, vgl. `open_framegrabber`.

'start_row': Auflistung aller unterstützten Werte für den Parameter `StartRow`, vgl. `open_framegrabber`.

'vertical_resolution': Auflistung aller unterstützten Werte für den Parameter `VerticalResolution`, vgl. `open_framegrabber`.

Weitere Informationen zu spezifischen Bildeinzugsschnittstellen findet sich in der Dokumentation aller Bildeinzugsschnittstellen im Verzeichnis *'doc/html/reference/acquisition'*.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in `open_framegrabber` aufgelistet) eine eigene Gruppe innerhalb der `info_framegrabber`, `open_framegrabber`, `close_framegrabber` und `close_all_framegrabbers` exklusiv ausgeführt werden.

D.h., `info_framegrabber`, `open_framegrabber`, `close_framegrabber` und `close_all_framegrabbers` verhindern die gleichzeitige Ausführung anderer Operatoren innerhalb der Gruppe, laufen aber mit allen nicht exklusiven Operatoren außerhalb der Gruppe parallel.

Auf Windows Systemen können Fehler-Dialogboxen vom Betriebssystems erscheinen, wenn abhängende Module des Interfaces nicht gefunden werden können. Dies ist beispielsweise der Fall, wenn das entsprechende SDK nicht installiert wurde. Das Erscheinen der Dialogboxen kann über Windows Error Mode kontrolliert werden. Informationen dazu finden sich in der Beschreibung von SetErrorMode in der MSDN Dokumentation.

Parameter

▷ **`Name`** (input_control) ..... string ⇝ *string*
Name der HALCON Bildeinzugsschnittstelle, d.h. Name der zugehörigen DLL (Windows) bzw. 'shared library' (Linux/macOS).
Defaultwert: 'File'
Wertevorschläge: `Name` ∈ {'ABS', 'ADLINK', 'AlkUSB3', 'Andor', 'BitFlow', 'Crevis', 'DahengCAM', 'DirectFile', 'DirectShow', 'Ensenso-NxLib', 'File', 'FocalSpecLCI', 'GenICamTL', 'GigEVision2', 'GingaDG', 'Ginga++', 'GStreamer', 'heliCamC3', 'KeyenceVJ', 'LinX', 'LPS36', 'LuCam', 'MatrixVisionAcquire', 'MILLite', 'MultiCam', 'O3D3xx', 'Opteon', 'PhoXi', 'PixeLINK', 'pylon', 'SaperaLT', 'Sentech', 'SICK-3DCamera', 'SICK-ScanningRuler', 'SiliconSoftware', 'Slink', 'TWAIN', 'uEye', 'USB3Vision', 'Video4Linux2', 'VRmUsbCam'}

▷ **`Query`** (input_control) ..... string ⇝ *string*
Name der gewählten Anfrage.
Defaultwert: 'info_boards'
Werteliste: `Query` ∈ {'defaults', 'general', 'info_boards', 'parameters', 'parameters_readonly', 'parameters_writeonly', 'revision', 'bits_per_channel', 'camera_type', 'color_space', 'device', 'external_trigger', 'field', 'generic', 'horizontal_resolution', 'image_height', 'image_width', 'port', 'start_column', 'start_row', 'vertical_resolution'}

▷ **`Information`** (output_control) ..... string ⇝ *string*
Textuelle Information (abhängig von `Query`).

▷ **`ValueList`** (output_control) ..... string-array ⇝ *string* / integer / real
Werteliste (abhängig von `Query`).

Beispiel

```
* Select a suitable image acquisition interface name AcqName
info_framegrabber(AcqName,'port',Information,Values)
* Open image acquisition device using the default settings, see
* documentation of the actually used interface for more details
open_framegrabber(AcqName,1,1,0,0,0,0,'default',-1,'default',-1.0, \
                  'default','default','default',-1,-1,AcqHandle)
grab_image(Image,AcqHandle)
close_framegrabber(AcqHandle)
```

Ergebnis

Sind die Parameterwerte korrekt und ist die angegebene Bildeinzugsschnittstelle verfügbar, liefert `info_framegrabber` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_framegrabber`

Nachfolger

`open_framegrabber`

Siehe auch

`open_framegrabber`

Modul

Foundation

```
open_framegrabber ( : : Name, HorizontalResolution,
    VerticalResolution, ImageWidth, ImageHeight, StartRow,
    StartColumn, Field, BitsPerChannel, ColorSpace, Generic,
    ExternalTrigger, CameraType, Device, Port, LineIn : AcqHandle )
```

Öffnen und Konfigurieren eines Bildeinzugsgeräts.

`open_framegrabber` öffnet und konfiguriert das ausgewählte Bildeinzugsgerät. Dabei wird insbesondere auch die Verbindung zum Bildeinzugsgerät getestet, das Bildeinzugsgerät für andere Prozesse gesperrt und gegebenenfalls Speicher für die Datenpuffer reserviert. Der eigentliche Bildeinzug erfolgt dann mittels `grab_image`, `grab_data`, `grab_image_async` oder `grab_data_async`. Wird das Bildeinzugsgerät nicht mehr benötigt, sollte er mittels `close_framegrabber` wieder geschlossen und so für andere Prozesse freigegeben werden. Einige Bildeinzugsgeräte erlauben das gleichzeitige Öffnen mehrerer Instanzen.

Für alle Parameter kann explizit der gerätespezifische Default-Wert verwendet werden (vgl. Parameterbeschreibung unten). Nähere Informationen zu einem bestimmten Bildeinzugsgerät erhält man mit Hilfe von `info_framegrabber`. Die ausführliche Dokumentation zu allen gerätespezifischen Parametern findet sich in der Dokumentation für die verwendete Bildeinzugsschnittstelle im Verzeichnis *'doc/html/reference/acquisition'*.

Die Bedeutung der Parameter von `open_framegrabber` ist im einzelnen:

`HorizontalResolution`, **`VerticalResolution`** Gewünschte Auflösung des Bildeinzugsgeräts.

`ImageWidth`, **`ImageHeight`** Größe des Bildausschnittes, der von `grab_image` etc. geliefert werden soll.

`StartRow`, **`StartColumn`** Linke obere Ecke des gewünschten Bildausschnittes.

`Field` Gewünschtes Halbbild (*'first'*, *'second'* oder *'next'*) bzw. Einstellung eines Vollbildes.

`BitsPerChannel` Die Anzahl an Bits, die das Bildeinzugsgerät pro Pixel und Bildkanal überträgt (typischerweise *5*, *8*, *10*, *12* oder *16*).

`ColorSpace` Festlegung des Ausgabe-Farbformats der eingezogenen Bilder (typischerweise *'gray'* oder *'raw'* für einkanalige bzw. *'rgb'* oder *'yuv'* für dreikanalige Bilder).

`Generic` Generischer Parameter mit gerätespezifischer Bedeutung. Diese ist mittels `info_framegrabber` abfragbar.

`ExternalTrigger` Aktivierung des externen Triggers des Bildeinzugsgeräts (soweit unterstützt).

`CameraType` Nähere Spezifikation des gewünschten Bildeinzugsgeräts (typischerweise die Angabe des analogen Videoformats oder der Name der zu verwendenden Kamerakonfigurationsdatei). Die Bedeutung dieses Parameters ist gerätespezifisch und kann mittels `info_framegrabber` abgefragt werden.

`Device` Gerätename des zu verwendenden Bildeinzugsgeräts.

`Port` Port, an den das Bildeinzugsgerät angeschlossen ist.

`LineIn` Selektion des Kameraeingangs (falls ein Multiplexer für den Port verfügbar ist).

Der Operator open_framegrabber liefert mit AcqHandle einen Handle auf das geöffneten Bildeinzugsgerät zurück.

Achtung

Bedingt durch die Vielzahl der unterstützten Bildeinzugsgeräte, umfasst open_framegrabber eine große Anzahl von Parametern. Für ein bestimmten Bildeinzugsgerät werden deshalb nicht immer alle Parameter benötigt.

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren also info_framegrabber, open_framegrabber, get_framegrabber_param, set_framegrabber_param, get_framegrabber_callback, set_framegrabber_callback, get_framegrabber_lut, set_framegrabber_lut, grab_image_start, grab_image, grab_image_async, grab_data, grab_data_async, close_framegrabber und close_all_framegrabbers eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

D.h., info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers verhindern die gleichzeitige Ausführung anderer Operatoren innerhalb der Gruppe, laufen aber mit allen nicht exklusiven Operatoren außerhalb der Gruppe parallel.

Auf Windows Systemen können Fehler-Dialogboxen vom Betriebssystems erscheinen, wenn abhängende Module des Interfaces nicht gefunden werden können. Dies ist beispielsweise der Fall, wenn das entsprechende SDK nicht installiert wurde. Das Erscheinen der Dialogboxen kann über Windows Error Mode kontrolliert werden. Informationen dazu finden sich in der Beschreibung von SetErrorMode in der MSDN Dokumentation.

Parameter

▷ **Name** (input_control) ..... string ⇝ *string*
Name der HALCON-Bildeinzugsschnittstelle, d.h. Name der zugehörigen DLL (Windows) bzw. 'shared library' (Linux/macOS).
Defaultwert: 'File'
Wertevorschläge: Name ∈ {'ABS', 'ADLINK', 'AlkUSB3', 'Andor', 'BitFlow', 'Crevis', 'DahengCAM', 'DirectFile', 'DirectShow', 'Ensenso-NxLib', 'File', 'FocalSpecLCI', 'GenICamTL', 'GigEVision2', 'GingaDG', 'Ginga++', 'GStreamer', 'heliCamC3', 'KeyenceVJ', 'LinX', 'LPS36', 'LuCam', 'MatrixVisionAcquire', 'MILLite', 'MultiCam', 'O3D3xx', 'Opteon', 'PhoXi', 'PixeLINK', 'pylon', 'SaperaLT', 'Sentech', 'SICK-3DCamera', 'SICK-ScanningRuler', 'SiliconSoftware', 'Slink', 'TWAIN', 'uEye', 'USB3Vision', 'Video4Linux2', 'VRmUsbCam'}

▷ **HorizontalResolution** (input_control) ..... extent.x ⇝ *integer*
Gewünschte horizontale Auflösung des Bildeinzugsgeräts (absoluter Wert oder *1* für Vollauflösung, *2* für halbe Auflösung bzw. *4* für viertel Auflösung).
Defaultwert: 1
Wertevorschläge: HorizontalResolution ∈ {1, 2, 4, 1600, 1280, 768, 640, 384, 320, 192, 160, -1}

▷ **VerticalResolution** (input_control) ..... extent.y ⇝ *integer*
Gewünschte vertikale Auflösung des Bildeinzugsgeräts (absoluter Wert oder *1* für Vollauflösung, *2* für halbe Auflösung bzw. *4* für viertel Auflösung).
Defaultwert: 1
Wertevorschläge: VerticalResolution ∈ {1, 2, 4, 1200, 1024, 576, 480, 288, 240, 144, 120, -1}

▷ **ImageWidth** (input_control) ..... rectangle.extent.x ⇝ *integer*
Breite des gewünschten Bildausschnittes (absoluter Wert oder *0* für HorizontalResolution - 2*StartColumn).
Defaultwert: 0
Wertevorschläge: ImageWidth ∈ {0, -1}

▷ **ImageHeight** (input_control) ..... rectangle.extent.y ⇝ *integer*
Höhe des gewünschten Bildausschnittes (absoluter Wert oder *0* für VerticalResolution - 2*StartRow).
Defaultwert: 0
Wertevorschläge: ImageHeight ∈ {0, -1}

▷ **StartRow** (input_control) ..... rectangle.origin.y ⇝ *integer*
Zeilennummer der oberen linken Ecke des gewünschten Bildausschnittes (bzw. Höhe eines Rahmens, falls ImageHeight = *0*).
Defaultwert: 0
Wertevorschläge: StartRow ∈ {0, -1}

▷ **StartColumn** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltennummer der oberen linken Ecke des gewünschten Bildausschnittes (bzw. Breite eines Rahmens, falls ImageWidth = *0*).
Defaultwert: 0
Wertevorschläge: StartColumn ∈ {0, -1}

▷ **Field** (input_control) . . . . . string ⇝ *string*
Gewünschtes Halbbild bzw. Vollbild.
Defaultwert: 'default'
Wertevorschläge: Field ∈ {'first', 'second', 'next', 'interlaced', 'progressive', 'default'}

▷ **BitsPerChannel** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl übertragener Bits pro Pixel und Bildkanal (-1: gerätespezifischer Defaultwert).
Defaultwert: -1
Wertevorschläge: BitsPerChannel ∈ {5, 8, 10, 12, 14, 16, -1}

▷ **ColorSpace** (input_control) . . . . . string(-array) ⇝ *string*
Festlegung des Ausgabe-Farbformats der eingezogenen Bilder, typischerweise *'gray'* oder *'raw'* für einkanalige bzw. *'rgb'* oder *'yuv'* für dreikanalige Bilder (*'default'*: gerätespezifischer Defaultwert).
Defaultwert: 'default'
Wertevorschläge: ColorSpace ∈ {'gray', 'raw', 'rgb', 'yuv', 'default'}

▷ **Generic** (input_control) . . . . . string(-array) ⇝ *real* / string / integer
Generischer Parameter mit gerätespezifischer Bedeutung.
Defaultwert: -1

▷ **ExternalTrigger** (input_control) . . . . . string ⇝ *string*
Trigger durch externes Signal.
Defaultwert: 'default'
Werteliste: ExternalTrigger ∈ {'true', 'false', 'default'}

▷ **CameraType** (input_control) . . . . . string(-array) ⇝ *string*
Verwendeter Kameratyp (*'default'*: gerätespezifischer Defaultwert).
Defaultwert: 'default'
Wertevorschläge: CameraType ∈ {'ntsc', 'pal', 'auto', 'default'}

▷ **Device** (input_control) . . . . . string(-array) ⇝ *string*
Device, an das das Bildeinzugsgerät angeschlossen ist (*'default'*: gerätespezifischer Defaultwert).
Defaultwert: 'default'
Wertevorschläge: Device ∈ {'-1', '0', '1', '2', '3', 'default'}

▷ **Port** (input_control) . . . . . integer(-array) ⇝ *integer*
Port, an den das Bildeinzugsgerät angeschlossen ist (*-1*: gerätespezifischer Defaultwert).
Defaultwert: -1
Wertevorschläge: Port ∈ {0, 1, 2, 3, -1}

▷ **LineIn** (input_control) . . . . . integer(-array) ⇝ *integer*
Kameraeingang (*-1*: gerätespezifischer Defaultwert).
Defaultwert: -1
Wertevorschläge: LineIn ∈ {1, 2, 3, 4, -1}

▷ **AcqHandle** (output_control) . . . . . framegrabber ⇝ *handle*
Handle für das neu geöffnete Bildeinzugsgerät.

Beispiel

```
* Select a suitable image acquisition interface name AcqName
info_framegrabber(AcqName,'port',Information,Values)
* Open image acquisition device using the default settings, see
* documentation of the actually used interface for more details
open_framegrabber(AcqName,1,1,0,0,0,0,'default',-1,'default',-1.0, \
                  'default','default','default',-1,-1,AcqHandle)
grab_image(Image,AcqHandle)
close_framegrabber(AcqHandle)
```

Ergebnis

Sind die Parameterwerte korrekt und konnte das gewünschte Bildeinzugsgerät geöffnet werden, liefert open_framegrabber den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`info_framegrabber`

Nachfolger

`grab_image`, `grab_data`, `grab_image_start`, `grab_image_async`, `grab_data_async`, `set_framegrabber_param`, `set_framegrabber_callback`

Siehe auch

`info_framegrabber`, `close_framegrabber`, `grab_image`

Modul

Foundation

`set_framegrabber_callback` `( : : AcqHandle, CallbackType, CallbackFunction, UserContext : )`

Registrieren einer Callback-Funktion für ein Bildeinzugsgerät.

Der Operator `set_framegrabber_callback` registriert eine Callback-Funktion für das durch `AcqHandle` angegebene Bildeinzugsgerät. Der Parameter `CallbackFunction` enthält einen Zeiger auf die zu registrierende Funktion, während `CallbackType` angibt, mit welcher Funktion des darunterliegenden API sie verbunden werden soll.

Typische Werte für `CallbackType` sind:

'exception': Der Bildeinzug hat eine Fehlerbehandlung ausgelöst.

'exposure_end': Die Belichtung des nächsten Bildes ist beendet.

'exposure_start': Die Belichtung des nächsten Bildes ist gestartet.

'transfer_end': Ein neues Bild ist verfügbar und kann mit `grab_image_async` abgeholt werden.

Abhängig von der Funktionalität des darunterliegenden API sind auch weitere Werte für `CallbackType` möglich. Alle tatsächlich verfügbaren Callback-Typen für ein bestimmtes Bildeinzugsgerät können durch den Aufruf von `get_framegrabber_param` mit dem Parameter `'available_callback_types'` abgefragt werden. Nähere Informationen finden sich in der Dokumentation der verwendeten Bildeinzugsschnittstelle.

Sobald der Callback registriert worden ist, wird bei jedem Eintreten des entsprechenden Ereignisses (z.B. die Benachrichtigung, dass die Belichtung abgeschlossen ist) die angegebene Callback-Funktion `CallbackFunction` aufgerufen. Ist `CallbackFunction` gleich NULL, wird der entsprechende Callback gelöscht.

Die Signatur der Callback-Funktion lautet folgendermaßen:

```
Herror HAcqCallback(void *AcqHandle, void *Context, void *UserContext)
```

Der erste Parameter der Callback-Funktion enthält das in `AcqHandle` übergebene Handle für das Bildeinzugsgerät, der zweite einen Zeiger auf schnittstellenabhängige Kontextdaten und der dritte Parameter ist der Zeiger auf die benutzerspezifischen Kontextdaten, der in `UserContext` angegeben wird.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in `open_framegrabber` aufgelistet) eine eigene Gruppe innerhalb der `info_framegrabber`, `open_framegrabber`, `close_framegrabber` und `close_all_framegrabbers` exklusiv ausgeführt werden.

`set_framegrabber_callback` läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

▷ **AcqHandle** (input_control) . . . . . framegrabber ⇝ *handle*
Zu benutzendes Bildeinzugsgerät.

▷ **CallbackType** (input_control) . . . . . string ⇝ *string*
Callback-Typ.
Defaultwert: 'transfer_end'
Wertevorschläge: `CallbackType` ∈ {'exception', 'exposure_end', 'exposure_start', 'transfer_end'}

▷ **CallbackFunction** (input_control) . . . . . pointer ⇝ *integer*
Zu setzender Zeiger auf die Callback-Funktion.

▷ **UserContext** (input_control) . . . . . pointer ⇝ *integer*
Zeiger auf benutzerspezifische Kontextdaten.

Ergebnis

Ist das Bildeinzugsgerät geöffnet und konnte der angegebene Callback erfolgreich gesetzt werden, liefert `set_framegrabber_callback` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- AcqHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_framegrabber`, `set_framegrabber_param`

Nachfolger

`grab_image`, `grab_data`, `grab_image_start`, `grab_image_async`, `grab_data_async`, `get_framegrabber_param`

Siehe auch

`open_framegrabber`, `get_framegrabber_callback`

Modul

Foundation

set_framegrabber_lut (: : AcqHandle, ImageRed, ImageGreen, ImageBlue :)

Setzen der Look-Up-Tabelle des Bildeinzugsgeräts.

`set_framegrabber_lut` setzt die Look-Up-Tabelle (LUT) des durch `AcqHandle` spezifizierten Bildeinzugsgerät. Diese Operation wird nicht von allen Bildeinzugsgeräten unterstützt.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in `open_framegrabber` aufgelistet) eine eigene Gruppe innerhalb der `info_framegrabber`, `open_framegrabber`, `close_framegrabber` und `close_all_framegrabbers` exklusiv ausgeführt werden.

`set_framegrabber_lut` läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

- ▷ **AcqHandle** (input_control) . . . . . framegrabber ⇝ *handle*
  Zu benutzendes Bildeinzugsgerät.
- ▷ **ImageRed** (input_control) . . . . . integer-array ⇝ *integer*
  Rotanteil der LUT-Einträge.
- ▷ **ImageGreen** (input_control) . . . . . integer-array ⇝ *integer*
  Grünanteil der LUT-Einträge.
- ▷ **ImageBlue** (input_control) . . . . . integer-array ⇝ *integer*
  Blauanteil der LUT-Einträge.

Ergebnis

`set_framegrabber_lut` liefert den Wert TRUE, sofern die übergebene LUT korrekt ist und das Bildeinzugsgerät geöffnet ist.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- AcqHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_framegrabber`, `get_framegrabber_lut`

Nachfolger

`grab_image`, `grab_data`, `grab_image_start`, `grab_image_async`, `grab_data_async`

Siehe auch

`get_framegrabber_lut`, `open_framegrabber`

Modul

Foundation

set_framegrabber_param (: : AcqHandle, Param, Value :)

Setzen spezifischer Parameter eines Bildeinzugsgeräts.

`set_framegrabber_param` setzt spezifische Parameter für das durch `AcqHandle` angegebene Bildeinzugsgerät. Nähere Informationen zu einem bestimmten Bildeinzugsgerät erhält man mit Hilfe von `info_framegrabber`. Die ausführliche Dokumentation zu allen gerätespezifischen Parametern findet sich in der Dokumentation für die verwendete Bildeinzugsschnittstelle im Verzeichnis *'doc/html/reference/acquisition'*.

Achtung

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in `open_framegrabber` aufgelistet) eine eigene Gruppe innerhalb der `info_framegrabber`, `open_framegrabber`, `close_framegrabber` und `close_all_framegrabbers` exklusiv ausgeführt werden.

`set_framegrabber_param` läuft mit allen nicht exklusiven Operatoren innerhalb und außerhalb der Gruppe parallel.

Parameter

▷ **AcqHandle** (input_control) . . . . . framegrabber ⇝ *handle*
Zu benutzendes Bildeinzugsgerät.

▷ **Param** (input_control) . . . . . string(-array) ⇝ *string*
Parametername.
Wertevorschläge: `Param` ∈ {'color_space', 'continuous_grabbing', 'external_trigger', 'grab_timeout', 'image_height', 'image_width', 'port', 'start_column', 'start_row', 'volatile'}

▷ **Value** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
Zu setzender Parameterwert.

Ergebnis

Ist das Bildeinzugsgerät geöffnet und wird der angegebene Parameter bzw. Parameterwert unterstützt, liefert `set_framegrabber_param` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- AcqHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_framegrabber`

Nachfolger

`grab_image`, `grab_data`, `grab_image_start`, `grab_image_async`, `grab_data_async`, `get_framegrabber_param`

Siehe auch

`open_framegrabber`, `info_framegrabber`, `get_framegrabber_param`

Modul

Foundation

6.2 Domain

add_channels (Regions, Image : GrayRegions : :)

Regionen mit Grauwerten versehen.

`add_channels` erzeugt ein Array an Bildern `GrayRegions`, mit einem Element für jede Eingaberegion in `Regions`. Diese Bilder entsprechen dem Eingabebild `Image` mit neuem Definitionsbereich, nämlich dem als der Durchschnitt des Definitionsbereiches des Eingabebildes mit der jeweiligen Region. Der neue Definitionsbereich kann also eine Teilmenge der Region sein. Die Größe der Bild-Matrix wird nicht verändert und alle Kanäle von `Image` werden übernommen.

Parameter

▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
Eingaberegionen (ohne Pixelwerte).

▷ **Image** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Eingabebild mit Pixelwerten für Regionen.

▷ **GrayRegions** (output_object) . . . . . image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Ausgabebild(er) mit Regionen und Pixelwerten (ein Bild pro Eingaberegion).
Parameteranzahl: Regions == GrayRegions

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `gen_circle`, `draw_region`

Nachfolger

`threshold`, `regiongrowing`, `get_domain`

Alternativen

`change_domain`, `reduce_domain`

Siehe auch

`full_domain`, `get_domain`, `intersection`

Modul

Foundation

change_domain (Image, NewDomain : ImageNew : :)

Verändert den Definitionsbereich eines Bildes.

`change_domain` verwendet die angegebene Region als neuen Definitionsbereich. Im Gegensatz zu `reduce_domain` wird hier nicht die Schnittmenge mit dem bisherigen Definitionsbereich gebildet, d.h. die Größe der Matrix wird nicht verändert. Dies impliziert insbesondere, dass die Region nicht größer sein darf als die Bildmatrix, andernfalls kann die Verwendung von derart inkonsistenten Bildobjekten bei nachfolgenden Operationen zu Fehlern und Systemabstürzen führen.

Achtung

Aus Laufzeitgründen wird die übergebene Region nicht auf Konsistenz (d.h. ob sie zur Bildmatrix passt) überprüft. Falsche Regionen führen bei nachfolgenden Operationen zu Systemabstürzen.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild.

▷ **NewDomain** (input_object) ..... region ⇝ *object*

Neuer Definitionsbereich.

▷ **ImageNew** (output_object) ..... image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Bild mit neuem Definitionsbereich.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`get_domain`

Alternativen

`reduce_domain`

Siehe auch

`full_domain`, `get_domain`, `intersection`

Modul

Foundation

full_domain (Image : ImageFull : :)

Definitionsbereich eines Bildes maximal erweitern.

full_domain trägt ein Rechteck mit der Kantenlänge des Bildes als neuen Definitionsbereich ein. Dies bedeutet, dass alle Pixel der Matrix bei weiteren Operationen mit einbezogen werden. Man erhält somit den gleichen Definitionsbereich, den man nach dem Einlesen oder Erzeugen eines Bildes hat. Die Größe der Matrix wird nicht verändert.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild.

▷ **ImageFull** (output_object) . . . . . image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Bild mit maximalem Definitionsbereich.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

get_domain

Alternativen

change_domain, reduce_domain

Siehe auch

get_domain, gen_rectangle1

Modul

Foundation

get_domain (Image : Domain : :)

Zugriff auf den Definitionsbereich eines Bildes.

get_domain liefert zu allen Bildern in der Eingabe die Definitionsbereiche als Region zurück.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebilder.

▷ **Domain** (output_object) . . . . . region(-array) ⇝ *object*

Definitionsbereiche der Eingabebilder.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.

Nachfolger

change_domain, reduce_domain, full_domain

Siehe auch

change_domain, reduce_domain, full_domain

Modul

Foundation

rectangle1_domain (Image : ImageReduced : Row1, Column1, Row2, Column2 :)

Verkleinert den Definitionsbereich eines Bildes auf ein Rechteck.

rectangle1_domain verkleinert den Definitionsbereich der Eingabebilder auf das angegebene Rechteck. Die Größe der Matrix wird nicht verändert.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild.

▷ **ImageReduced** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Bild mit reduziertem Definitionsbereich.

▷ **Row1** (input_control) ..... rectangle.origin.y ⇝ *integer*

Zeilenindex der linken oberen Ecke des Bildausschnittes.
Defaultwert: 100
Wertevorschläge: Row1 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: $0 \leq$ Row1 ≤ 1024

▷ **Column1** (input_control) ..... rectangle.origin.x ⇝ *integer*

Spaltenindex der linken oberen Ecke des Bildausschnittes.
Defaultwert: 100
Wertevorschläge: Column1 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: $0 \leq$ Column1 ≤ 1024

▷ **Row2** (input_control) ..... rectangle.origin.y ⇝ *integer*

Zeilenindex der rechten unteren Ecke des Bildausschnittes.
Defaultwert: 200
Wertevorschläge: Row2 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: $0 \leq$ Row2 ≤ 1024

▷ **Column2** (input_control) ..... rectangle.origin.x ⇝ *integer*

Spaltenindex der rechten unteren Ecke des Bildausschnittes.
Defaultwert: 200
Wertevorschläge: Column2 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: $0 \leq$ Column2 ≤ 1024

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

get_domain

Alternativen

change_domain, reduce_domain, add_channels

Siehe auch

full_domain, get_domain, intersection

Modul

Foundation

reduce_domain (Image, Region : ImageReduced : :)

Verkleinert den Definitionsbereich eines Bildes.

reduce_domain verkleinert den Definitionsbereich der Eingabebilder um die angegebene Region. Der neue Definitionsbereich berechnet sich als der Durchschnitt des alten Definitionsbereiches mit der Region. Der neue Definitionsbereich kann also eine Teilmenge der Region sein. Die Größe der Matrix wird nicht verändert.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild.

▷ **Region** (input_object) ..... region ⇝ *object*

Neuer Definitionsbereich.

▷ **ImageReduced** (output_object) ..... image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Bild mit reduziertem Definitionsbereich.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

get_domain

Alternativen

change_domain, rectangle1_domain, add_channels

Siehe auch

full_domain, get_domain, intersection

Modul

Foundation

6.3 Format

change_format (Image : ImagePart : Width, Height :)

Ändern der Bildgröße.

change_format vergrößert bzw. verkleinert die Eingabebilder auf die angegebene Höhe bzw. Breite. Bei einer Verkleinerung werden am „rechten“ bzw. „unteren“ Bildrand Teile abgeschnitten. Bei einer Vergrößerung werden die zusätzlichen Bereiche mit 0 besetzt. Der Definitionsbereich des Ausgabebildes entspricht dem des Eingabebildes, eventuell beschnitten auf die neue Bildgröße. Es wird kein Zooming durchgeführt.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild.

▷ **ImagePart** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Bild mit neuem Format.

▷ **Width** (input_control) ..... extent.x ⇝ *integer*

Breite des neuen Bildes.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {32, 64, 128, 256, 512, 768, 1024}

▷ **Height** (input_control) ..... extent.y ⇝ *integer*

Höhe des neuen Bildes.
Defaultwert: 512
Wertevorschläge: `Height` ∈ {32, 64, 128, 256, 512, 525, 1024}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`disp_image`

Alternativen

`crop_part`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Foundation

`crop_domain ( Image : ImagePart : : )`

Ausschneiden der gültigen Pixel.

`crop_domain` schneidet aus den Eingabebildern einen rechteckigen Ausschnitt aus. Dieses Rechteck ist das kleinste umschließende Rechteck um den Definitionsbereich des Bildes. Die neue Bildmatrix hat die Größe des Rechtecks.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real

Eingabebild.

▷ **ImagePart** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real

Bildausschnitt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`disp_image`

Alternativen

`crop_part`, `crop_rectangle1`, `change_format`, `reduce_domain`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Foundation

crop_domain_rel (Image : ImagePart : Top, Left, Bottom, Right :)

Ausschneiden eines Bildausschnittes um den Definitionsbereich.

`crop_domain_rel` schneidet aus den Eingabebildern einen rechteckigen Ausschnitt aus. Der Ausschnitt wird durch das umschließende Rechteck des Definitionsbereiches bestimmt. Das Rechteck kann durch die Steuerparameter oben (`Top`), links (`Left`), unten (`Bottom`) oder rechts (`Right`) verändert werden. Positive Werte bewirken eine Verkleinerung, negative Zahlen bewirken eine Vergrößerung. Sind alle Parameter gleich Null, dann bleibt das Rechteck unverändert.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Eingabebild.

▷ **ImagePart** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Bildausschnitt.

▷ **Top** (input_control) ..... integer ⇝ *integer*
Anzahl Zeilen die „oben“ abgeschnitten werden.
Defaultwert: -1
Wertevorschläge: `Top` ∈ {-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

▷ **Left** (input_control) ..... integer ⇝ *integer*
Anzahl Spalten die „links“ abgeschnitten werden.
Defaultwert: -1
Wertevorschläge: `Left` ∈ {-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

▷ **Bottom** (input_control) ..... integer ⇝ *integer*
Anzahl Zeilen die „unten“ abgeschnitten werden.
Defaultwert: -1
Wertevorschläge: `Bottom` ∈ {-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

▷ **Right** (input_control) ..... integer ⇝ *integer*
Anzahl Zeilen die „rechts“ abgeschnitten werden.
Defaultwert: -1
Wertevorschläge: `Right` ∈ {-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

Ergebnis

`crop_domain_rel` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

reduce_domain, threshold, connection, regiongrowing, pouring

Alternativen

crop_domain, crop_rectangle1

Siehe auch

smallest_rectangle1, intersection, gen_rectangle1, clip_region

Modul

Foundation

crop_part (Image : ImagePart : Row, Column, Width, Height :)

Ausschneiden eines oder mehrerer rechteckiger Bildausschnitte.

crop_part schneidet aus den Eingabebildern einen oder mehrere rechteckige Ausschnitte aus. Die Ausschnitte werden durch die Koordinaten der linken oberen Ecken und den Größen der Rechtecke beschrieben. Die linken oberen Ecken der Ausschnitte müssen innerhalb des Bildes liegen. Rechts und unten dürfen die Rechtecke aus dem Bild herausragen, allerdings wird der Definitionsbereich des entsprechenden Ausgabebildes so gesetzt, dass nur der Teil enthalten ist, der aus dem Eingabebild stammt. Liegen die rechteckigen Ausschnitte komplett im Eingabebild, hat jedes resultierende Bild die Größe des entsprechenden Rechtecks.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real

Eingabebild.

▷ **ImagePart** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real

Bildausschnitt.

▷ **Row** (input_control) ..... rectangle.origin.y(-array) ⇝ *integer*

Zeilenindex der linken oberen Ecke des Bildausschnittes.
Defaultwert: 100
Wertevorschläge: Row ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: 0 ≤ Row ≤ 1024

▷ **Column** (input_control) ..... rectangle.origin.x(-array) ⇝ *integer*

Spaltenindex der linken oberen Ecke des Bildausschnittes.
Defaultwert: 100
Wertevorschläge: Column ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: 0 ≤ Column ≤ 1024

▷ **Width** (input_control) ..... rectangle.extent.x(-array) ⇝ *integer*

Breite des neuen Bildes.
Defaultwert: 128
Wertevorschläge: Width ∈ {32, 64, 128, 256, 512, 768}
Typischer Wertebereich: 0 ≤ Width ≤ 1024

▷ **Height** (input_control) ..... rectangle.extent.y(-array) ⇝ *integer*

Höhe des neuen Bildes.
Defaultwert: 128
Wertevorschläge: Height ∈ {32, 64, 128, 256, 512, 525}
Typischer Wertebereich: 0 ≤ Height ≤ 1024

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

disp_image

Alternativen

crop_rectangle1, crop_domain, change_format, reduce_domain

Siehe auch

zoom_image_size, zoom_image_factor

Modul

Foundation

crop_rectangle1 (Image : ImagePart : Row1, Column1, Row2, Column2 :)

Ausschneiden eines oder mehrerer rechteckiger Bildausschnitte.

crop_rectangle1 schneidet aus den Eingabebildern einen oder mehrere rechteckige Ausschnitte aus. Die Ausschnitte werden durch die Koordinaten der linken oberen Ecken und die Koordinaten der rechten unteren Ecken beschrieben. Die linken oberen Ecken der Ausschnitte müssen innerhalb des Bildes liegen. Rechts und unten dürfen die Rechtecke aus dem Bild herausragen, allerdings wird der Definitionsbereich des entsprechenden Ausgabebildes so gesetzt, dass nur der Teil enthalten ist, der aus dem Eingabebild stammt. Liegen die rechteckigen Ausschnitte komplett im Eingabebild, hat jedes resultierende Bild die Größe des entsprechenden Rechtecks.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real

Eingabebild.

▷ **ImagePart** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real

Bildausschnitt.

▷ **Row1** (input_control) ..... rectangle.origin.y(-array) ⇝ *integer*

Zeilenindex der linken oberen Ecke des Bildausschnittes.
Defaultwert: 100
Wertevorschläge: Row1 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: $0 \leq$ Row1 ≤ 1024

▷ **Column1** (input_control) ..... rectangle.origin.x(-array) ⇝ *integer*

Spaltenindex der linken oberen Ecke des Bildausschnittes.
Defaultwert: 100
Wertevorschläge: Column1 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: $0 \leq$ Column1 ≤ 1024

▷ **Row2** (input_control) ..... rectangle.corner.y(-array) ⇝ *integer*

Zeilenindex der rechten unteren Ecke des Bildausschnittes.
Defaultwert: 200
Wertevorschläge: Row2 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: $0 \leq$ Row2 ≤ 1024

▷ **Column2** (input_control) ..... rectangle.corner.x(-array) ⇝ *integer*

Spaltenindex der rechten unteren Ecke des Bildausschnittes.
Defaultwert: 200
Wertevorschläge: Column2 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich: $0 \leq$ Column2 ≤ 1024

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`disp_image`

Alternativen

`crop_part`, `crop_domain`, `change_format`, `reduce_domain`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Foundation

crop_rectangle2 (Image : ImagePart : Row, Column, Phi, Length1, Length2, AlignToAxis, Interpolation :)

Ausschneiden eines oder mehrerer beliebig orientierter rechteckiger Bildausschnitte.

`crop_rectangle2` schneidet aus den Eingabebildern einen oder mehrere rechteckige Ausschnitte aus und gibt diese in `ImagePart` zurück. Die Bildausschnitte werden jeweils über den Schwerpunkt (`Row`, `Column`), die Orientierung `Phi` und die halben Kantenlängen `Length1` und `Length2` beschrieben. `Phi` bestimmt im mathematisch positiven Sinn den Winkel in Bogenmaß zwischen der horizontalen Achse und der ersten Kante (mit Länge `Length1`). Die Parameter der Rechtecke verwenden Pixel-zentrierte, subpixel-genaue Koordinaten, siehe Transformationen / 2D-Transformationen.

Der Parameter `AlignToAxis` bestimmt die Ausrichtung der Ausgabebilder. Mögliche Werte:

- *'true'*: Der Bildausschnitt wird so rotiert, sodass er an den Koordinatenachsen ausgerichtet ist. Für diese Transformation muss eine geeignete Interpolation durchgeführt werden. Eine detaillierte Beschreibung der Interpolationsmodi findet sich in der Beschreibung von `affine_trans_image`. Die Rechtecke dürfen aus dem Eingabebild herausragen. Die Grauwerte von Pixeln, welche außerhalb des Eingabebildes liegen, werden auf 0 gesetzt.
- *'false'*: Das kleinstmögliche umschließende Rechteck des Eingabe-Rechtecks wird ausgeschnitten und der Definitionsbereich entsprechend gesetzt. Die Rechtecke dürfen aus dem Eingabebild herausragen. Jedoch entspricht dann das Ausgabebild dem Teil des Rechtecks, der innerhalb des Eingabebildes liegt. Für dieses Verfahren wird keine Interpolation benötigt, sodass der Parameter `Interpolation` keinen Einfluss auf das Endergebnis hat.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Eingabebild(er).

▷ **ImagePart** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Bildausschnitt(e).

▷ **Row** (input_control) . . . . . rectangle2.center.y(-array) ⇝ *real* / integer
Zeilenindex des Zentrums des Bildausschnittes.
Defaultwert: 300.0
Wertevorschläge: `Row` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
(lin)
Empfohlene Schrittweite: 10.0

▷ **Column** (input_control) . . . . . rectangle2.center.x(-array) ⇝ *real* / integer
Spaltenindex des Zentrums des Bildausschnittes.
Defaultwert: 200.0
Wertevorschläge: `Column` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
(lin)
Empfohlene Schrittweite: 10.0

▷ **Phi** (input_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real* / integer
Orientierung des Rechtecks (Bogenmaß).
Defaultwert: 0.0
Wertevorschläge: `Phi` ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
Typischer Wertebereich: -1.178097 ≤ `Phi` ≤ 1.178097 (lin)

▷ **Length1** (input_control) . . . . . rectangle2.hwidth(-array) ⇝ *real* / integer
Länge der ersten Halbachse.
Defaultwert: 100.0
Wertevorschläge: `Length1` ∈ {3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0}
(lin)
Empfohlene Schrittweite: 10.0

▷ **Length2** (input_control) . . . . . rectangle2.hheight(-array) ⇝ *real* / integer
Länge der zweiten Halbachse.
Defaultwert: 20.0
Wertevorschläge: `Length2` ∈ {1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0}
(lin)
Empfohlene Schrittweite: 10.0

▷ **AlignToAxis** (input_control) . . . . . string ⇝ *string*
Gibt an, ob der Bildausschnitt an den Koordinatenachsen ausgerichtet wird.
Defaultwert: 'true'
Werteliste: `AlignToAxis` ∈ {'true', 'false'}

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Interpolationsverfahren für die Transformation.
Defaultwert: 'constant'
Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`disp_image`

Alternativen

`crop_domain`, `crop_rectangle1`, `crop_part`, `change_format`, `reduce_domain`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Foundation

tile_channels (Image : TiledImage : NumColumns, TileOrder :)

Zusammenfügen von mehreren Bildern zu einem großen Bild.

`tile_channels` fügt ein aus mehreren Kanälen bestehendes Bild durch Kachelung zu einem großen einkanaligen Bild zusammen. Das Eingabebild `Image` besteht aus *Num* Bildern gleicher Größe, die in den einzelnen Kanälen gespeichert werden. Das Ausgabebild `TiledImage` besteht aus einem einzigen Kanal, in dem die *Num* Eingabekanäle in `NumColumns` Spalten gekachelt worden sind. Insbesondere kann `tile_channels` also keine Farbbilder zusammenfügen. Hierzu kann `tile_images` verwendet werden. Der Parameter `TileOrder` legt die Richtung fest, in der die Eingabekanäle in die Ausgabe kopiert werden, falls dies nicht schon durch `NumColumns` festgelegt wird (d.h., falls `NumColumns` != 1 und `NumColumns` != *Num*). Falls `TileOrder` = *'horizontal'*, werden die Bilder in horizontaler Richtung kopiert, d.h. der zweite Kanal des Eingabebildes liegt rechts neben dem ersten Kanal. Falls `TileOrder` = *'vertical'*, werden die Bilder in vertikaler Richtung kopiert, d.h. der zweite Kanal des Eingabebildes liegt unter dem ersten Kanal. Der Definitionsbereich (die Region) des Ausgabebildes `TiledImage` wird berechnet, indem die Region des Eingabebildes `Image` an die entsprechenden Stellen im Ausgabebild kopiert wird. Falls *Num* kein Vielfaches von `NumColumns` ist, hat das Ausgabebild undefinierte Grauwerte in der unteren linken Ecke des Bildes. Die Ausgaberegion ist entsprechend gesetzt.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Eingabebild.

▷ **TiledImage** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Gekacheltes Ausgabebild.

▷ **NumColumns** (input_control) ..... integer ⇝ *integer*
Anzahl der Spalten, die für das Ausgabebild verwendet werden
Defaultwert: 1
Wertevorschläge: `NumColumns` ∈ {1, 2, 3, 4, 5, 6, 7}
Restriktion: NumColumns >= 1

▷ **TileOrder** (input_control) ..... string ⇝ *string*
Reihenfolge der Eingabebilder im Ausgabebild.
Defaultwert: 'vertical'
Werteliste: `TileOrder` ∈ {'horizontal', 'vertical'}

Beispiel

```
* Grab 5 single-channel images and stack them vertically.
gen_rectangle1 (Image, 0, 0, Height-1, Width-1)
for I := 1 to 5 by 1
    grab_image_async (ImageGrabbed, AcqHandle, -1)
    append_channel (Image, ImageGrabbed, Image)
endfor
tile_channels (Image, TiledImage, 1, 'vertical')
```

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `tile_channels` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`append_channel`

Alternativen

`tile_images`, `tile_images_offset`

Siehe auch

`change_format`, `crop_part`, `crop_rectangle1`

Modul

Foundation

tile_images (Images : TiledImage : NumColumns, TileOrder :)

Zusammenfügen von mehreren Bildobjekten zu einem großen Bild.

`tile_images` fügt mehrere Bildobjekte, die die gleiche Anzahl von Kanälen besitzen müssen, durch Kachelung zu einem großen Bild zusammen. Das Eingabeobjekt `Images` besteht aus *Num* Bildern, die unterschiedlich groß sein können. Das Ausgabebild `TiledImage` besteht aus einem Bildobjekt, das genauso viele Kanäle besitzt, wie die Eingabebilder. Im Ausgabebild sind die *Num* Eingabekanäle in `NumColumns` Spalten gekachelt. Jede Kachel besitzt dieselbe Größe, die sich aus der maximalen Breite und Höhe der Eingabebilder bestimmt. Falls

ein Eingabebild kleiner als die Kachelgröße ist, wird es in die Mitte der entsprechenden Kachel kopiert. Der Parameter `TileOrder` legt die Richtung fest, in der die Eingabekanäle in die Ausgabe kopiert werden, falls dies nicht schon durch `NumColumns` festgelegt wird (d.h., falls `NumColumns` != 1 und `NumColumns` != *Num*). Falls `TileOrder` = *'horizontal'*, werden die Bilder in horizontaler Richtung kopiert, d.h. das zweite Eingabebild liegt rechts neben dem ersten Eingabebild. Falls `TileOrder` = *'vertical'*, werden die Bilder in vertikaler Richtung kopiert, d.h. das zweite Eingabebild liegt unter dem ersten Eingabebild. Der Definitionsbereich (die Region) des Ausgabebildes `TiledImage` wird berechnet, indem die Regionen des Eingabebildes `Images` an die entsprechenden Stellen im Ausgabebild kopiert werden. Falls *Num* kein Vielfaches von `NumColumns` ist, hat das Ausgabebild undefinierte Grauwerte in der unteren linken Ecke des Bildes. Die Ausgaberegion ist entsprechend gesetzt.

Parameter

▷ **`Images`** (input_object) ..... (multichannel-)image-array ⇝ *object* : byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Eingabebilder.

▷ **`TiledImage`** (output_object) ..... (multichannel-)image ⇝ *object* : byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Gekacheltes Ausgabebild.

▷ **`NumColumns`** (input_control) ..... integer ⇝ *integer*
Anzahl der Spalten, die für das Ausgabebild verwendet werden
Defaultwert: 1
Wertevorschläge: `NumColumns` ∈ {1, 2, 3, 4, 5, 6, 7}
Restriktion: NumColumns >= 1

▷ **`TileOrder`** (input_control) ..... string ⇝ *string*
Reihenfolge der Eingabebilder im Ausgabebild.
Defaultwert: 'vertical'
Werteliste: `TileOrder` ∈ {'horizontal', 'vertical'}

Beispiel

```
* Grab 5 (multi-channel) images and stack them vertically.
gen_empty_obj (Images)
for I := 1 to 5 by 1
    grab_image_async (ImageGrabbed, AcqHandle, -1)
    concat_obj (Images, ImageGrabbed, Images)
endfor
tile_images (Images, TiledImage, 1, 'vertical')
```

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `tile_images` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`append_channel`

Alternativen

`tile_channels`, `tile_images_offset`

Siehe auch

`change_format`, `crop_part`, `crop_rectangle1`

Modul

Foundation

tile_images_offset (Images : TiledImage : OffsetRow, OffsetCol, Row1, Col1, Row2, Col2, Width, Height :)

Zusammenfügen von mehreren Bildobjekten zu einem großen Bild mit expliziter Positionsangabe.

tile_images_offset fügt mehrere Bildobjekte, die die gleiche Anzahl von Kanälen besitzen müssen, durch Kachelung zu einem großen Bild zusammen. Das Eingabeobjekt Images besteht aus *Num* Bildern, die unterschiedlich groß sein können. Das Ausgabebild TiledImage besteht aus einem Bildobjekt, das genauso viele Kanäle besitzt, wie die Eingabebilder. Die Größe des Ausgabebildes wird durch die Parameter Width und Height bestimmt. Die Position der oberen linken Ecke der Eingabebilder im Ausgabebild wird durch die Parameter OffsetRow und OffsetCol bestimmt. Die beiden Parameter müssen genau *Num* Werte enthalten. Optional kann jedes Eingabebild auf ein beliebiges Rechteck, das kleiner als das Eingabebild ist, zugeschnitten werden. Hierzu sind die Parameter Row1, Col1, Row2 und Col2 entsprechend zu setzen. Wird irgendeiner dieser vier Parameter auf *-1* gesetzt, wird das entsprechende Eingabebild nicht beschnitten. In jedem Fall müssen alle vier Parameter *Num* Werte enthalten. Falls die Eingabebilder beschnitten werden, beziehen sich die Positionsparameter OffsetRow und OffsetCol auf die obere linke Ecke des beschnittenen Bildes. Falls sich die Eingabebilder im Ausgabebild überlappen (unter der Beachtung des jeweiligen Definitionsbereiches), so überschreibt ein Bild mit höherem Index in Images die Bilddaten eines vorhergehenden Bildes. Der Definitionsbereich (die Region) des Ausgabebildes TiledImage wird berechnet, indem die Regionen des Eingabebildes Images an die entsprechenden Stellen im Ausgabebild kopiert werden.

Achtung

Falls die Bilder alle gleich groß sind und sich genau aneinanderfügen, ist der Operator tile_images normalerweise etwas schneller.

Parameter

- ▷ **Images** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
  Eingabebilder.
- ▷ **TiledImage** (output_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
  Gekacheltes Ausgabebild.
- ▷ **OffsetRow** (input_control) ..... point.y(-array) ⇝ *integer*
  Zeilenkoordinate der linken oberen Ecke der Eingabebilder im Ausgabebild.
  Defaultwert: 0
  Wertevorschläge: OffsetRow ∈ {0, 50, 100, 150, 200, 250}
- ▷ **OffsetCol** (input_control) ..... point.x(-array) ⇝ *integer*
  Spaltenkoordinate der linken oberen Ecke der Eingabebilder im Ausgabebild.
  Defaultwert: 0
  Wertevorschläge: OffsetCol ∈ {0, 50, 100, 150, 200, 250}
- ▷ **Row1** (input_control) ..... rectangle.origin.y(-array) ⇝ *integer*
  Zeilenkoordinate der oberen linken Ecke des zu kopierenden Teils des jeweiligen Bildes.
  Defaultwert: -1
  Wertevorschläge: Row1 ∈ {-1, 0, 10, 20, 50, 100, 200, 300, 500}
- ▷ **Col1** (input_control) ..... rectangle.origin.x(-array) ⇝ *integer*
  Spaltenkoordinate der oberen linken Ecke des zu kopierenden Teils des jeweiligen Bildes.
  Defaultwert: -1
  Wertevorschläge: Col1 ∈ {-1, 0, 10, 20, 50, 100, 200, 300, 500}
- ▷ **Row2** (input_control) ..... rectangle.corner.y(-array) ⇝ *integer*
  Zeilenkoordinate der rechten unteren Ecke des zu kopierenden Teils des jeweiligen Bildes.
  Defaultwert: -1
  Wertevorschläge: Row2 ∈ {-1, 0, 10, 20, 50, 100, 200, 300, 500}
- ▷ **Col2** (input_control) ..... rectangle.corner.x(-array) ⇝ *integer*
  Spaltenkoordinate der rechten unteren Ecke des zu kopierenden Teils des jeweiligen Bildes.
  Defaultwert: -1
  Wertevorschläge: Col2 ∈ {-1, 0, 10, 20, 50, 100, 200, 300, 500}
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Ausgabebildes.
  Defaultwert: 512
  Wertevorschläge: Width ∈ {32, 64, 128, 256, 512, 768, 1024, 2048, 4096}

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Ausgabebildes.
Defaultwert: 512
Wertevorschläge: `Height` ∈ {32, 64, 128, 256, 512, 525, 1024, 2048, 4096}

Beispiel

```
* Example 1
* Grab 2 (multi-channel) NTSC images, crop the bottom 5 lines off
* of each image, the right 5 columns off of the first image, and
* the left five lines off of the second image, and put the cropped
* images side-by-side.
gen_empty_obj (Images)
for I := 1 to 2 by 1
    grab_image_async (ImageGrabbed, AcqHandle, -1)
    concat_obj (Images, ImageGrabbed, Images)
endfor
tile_images_offset (Images, TiledImage, [0,635], [0,0], [0,0], \
                    [0,5], [474,474], [634,639], 635, 950)

* Example 2
* Enlarge image by 15 rows and columns on all sides
EnlargeColsBy := 15
EnlargeRowsBy := 15
get_image_pointer1 (Image, Pointer, Type, WidthImage, HeightImage)
tile_images_offset (Image, EnlargedImage, EnlargeRowsBy, EnlargeColsBy, \
                    -1, -1, -1, -1, WidthImage + EnlargeColsBy*2, \
                    HeightImage + EnlargeRowsBy*2)
```

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `tile_images_offset` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`append_channel`

Alternativen

`tile_channels`, `tile_images`

Siehe auch

`change_format`, `crop_part`, `crop_rectangle1`

Modul

Foundation

6.4 Generierung

copy_image (Image : DupImage : :)

Physikalisches Kopieren eines Bildes.

copy_image kopiert das Eingabebild in eine neues Bild mit gleichem Definitionsbereich wie das Eingabebild. Im Gegensatz zu HALCON-Operatoren wie copy_obj wird hierbei eine physikalische Kopie der Kanäle angelegt. Diese kann z.B. dazu verwendet werden, um die Grauwerte zu modifizieren (siehe get_image_pointer1).

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Bild, das kopiert werden soll.

▷ **DupImage** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Kopiertes Bild.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_image, gen_image_const

Nachfolger

set_grayval, get_image_pointer1

Alternativen

set_grayval, paint_gray, gen_image_const, gen_image_proto

Siehe auch

get_image_pointer1

Modul

Foundation

gen_image1 (: Image : Type, Width, Height, PixelPointer :)

Erzeugen eines Bildes aus einem Zeiger auf die Pixel.

gen_image1 erzeugt ein Bild in der Größe Width × Height. Die Pixel in PixelPointer sind zeilensequentiell abgelegt. Der Typ der übergebenen Pixel (PixelPointer) muss mit Type übereinstimmen (siehe gen_image_const für eine detailliertere Beschreibung der Pixeltypen). Der Speicher für das neue Bild wird von HALCON neu angelegt. Der Speicherplatz auf den PixelPointer kann also nach dem Aufruf freigegeben werden. Wie ein Zeigerwert übergeben wird hängt von der benutzten Operatorsignatur und der Programmierumgebung ab. Es ist sicherzustellen, dass die tatsächliche Speicheradresse unter der die Bilddaten gespeichert sind übergeben wird und nicht die Adresse der Zeigervariablen. Es ist darauf zu achten, dass 64-Bit-Zeiger auf 64-Bit-Architekturen nicht beschnitten werden.

Achtung

gen_image1 überprüft nicht, ob PixelPointer einen gültigen Wert hat oder nicht. Es muss also darauf geachtet werden, dass der Wert gültig ist, da es sonst zum Programmabsturz kommen kann!

Parameter

▷ **Image** (output_object) ..... image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real

Erzeugtes Bild mit neuer Bildmatrix.

▷ **Type** (input_control) ..... string ⇝ *string*

Pixeltyp.
Defaultwert: 'byte'
Werteliste: Type ∈ {'byte', 'direction', 'cyclic', 'int1', 'int2', 'uint2', 'int4', 'real'}

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Width` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width >= 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert: 512
Wertevorschläge: `Height` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Height` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height >= 1

▷ **PixelPointer** (input_control) ..... pointer ⇝ *integer*
Zeiger auf den ersten Grauwert.

Beispiel

```
void NewImage(Hobject *new)
{
  unsigned char  image[768*525];
  int            r,c;
  for (r=0; r<525; r++)
    for (c=0; c<768; c++)
      image[r*768+c] = c % 255;
  gen_image1(new,"byte",768,525,(Hlong)image);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image1` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_image_const`, `get_image_pointer1`

Alternativen

`gen_image3`, `gen_image1_extern`, `gen_image_const`, `get_image_pointer1`

Siehe auch

`reduce_domain`, `paint_gray`, `paint_region`, `set_grayval`

Modul

Foundation

gen_image1_extern (: Image : Type, Width, Height, PixelPointer, ClearProc :)

Erzeugen eines Bildes aus einem Zeiger auf Pixel (mit Speicherverwaltung).

`gen_image1_extern` erzeugt ein Bild in der Größe `Width` × `Height`. Die Pixel in `PixelPointer` sind zeilensequentiell abgelegt. Der Typ der übergebenen Pixel (`PixelPointer`) muss mit `Type` übereinstimmen

(siehe gen_image_const für eine detailliertere Beschreibung der Pixeltypen). Wie ein Zeigerwert übergeben wird hängt von der benutzten Operatorsignatur und der Programmierumgebung ab. Es ist sicherzustellen, dass die tatsächliche Speicheradresse unter der die Bilddaten gespeichert sind übergeben wird und nicht die Adresse der Zeigervariablen. Es ist darauf zu achten, dass 64-Bit-Zeiger auf 64-Bit-Architekturen nicht beschnitten werden.

Der Speicher für das neue Bild wird von HALCON im Gegensatz zu gen_image1 nicht neu angelegt und damit auch nicht kopiert. Der Speicherplatz auf den PixelPointer zeigt, muss also beim Löschen des Objektes Image freigegeben werden. Dies geschieht durch die, vom Aufrufer zur Verfügung gestellte, Prozedur ClearProc. Diese Prozedur muss folgende Signatur haben

void ClearProc(void* ptr);

und wird beim Löschen von Image mit der __cdecl Aufrufkonvention aufgerufen. Falls der Speicher nicht freigegeben werden soll (im Fall von Framegrabbern oder statischem Speicher), kann eine Prozedur "ohne Rumpf"oder der NULL-Pointer übergeben werden. Analog zum Parameter PixelPointer muss der Zeiger auf die Prozedur abhängig von der benutzten Operatorsignatur und der Programmierumgebung übergeben werden.

Achtung

gen_image1_extern überprüft nicht, ob in PixelPointer genug Speicher für ein Bild der Größe Width × Height allokiert wurde.

Ebenso überprüft gen_image1_extern nicht, ob PixelPointer einen gültigen Wert hat oder nicht. Es muss also darauf geachtet werden, dass der Wert gültig ist, da es sonst zum Programmabsturz kommen kann!

Parameter

- ▷ **Image** (output_object) . . . . . image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
  Erzeugtes HALCON-Bild.
- ▷ **Type** (input_control) . . . . . string ⇝ *string*
  Pixeltyp.
  Defaultwert: 'byte'
  Werteliste: Type ∈ {'int1', 'int2', 'uint2', 'int4', 'byte', 'real', 'direction', 'cyclic'}
- ▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
  Breite des Bildes.
  Defaultwert: 512
  Wertevorschläge: Width ∈ {128, 256, 512, 1024}
  Typischer Wertebereich: 1 ≤ Width ≤ 512 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: Width >= 1
- ▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
  Höhe des Bildes.
  Defaultwert: 512
  Wertevorschläge: Height ∈ {128, 256, 512, 1024}
  Typischer Wertebereich: 1 ≤ Height ≤ 512 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: Height >= 1
- ▷ **PixelPointer** (input_control) . . . . . pointer ⇝ *integer*
  Zeiger auf den ersten Grauwert.
- ▷ **ClearProc** (input_control) . . . . . pointer ⇝ *integer*
  Zeiger auf die Prozedur, die den Speicher des Bildes beim Löschen des Objektes wieder freigibt.
  Defaultwert: 0

Beispiel

```
void NewImage(Hobject *new)
{
  unsigned char  *image;
  int            r,c;
  image = malloc(640*480);
  for (r=0; r<480; r++)
    for (c=0; c<640; c++)
```

```
      image[r*640+c] = c % 255;
  gen_image1_extern(new,"byte",640,480,(Hlong)image,(Hlong)free);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_image1_extern den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

gen_image1, gen_image_const, get_image_pointer1, gen_image3_extern

Siehe auch

reduce_domain, paint_gray, paint_region, set_grayval

Modul

Foundation

gen_image1_rect (: Image : PixelPointer, Width, Height, VerticalPitch, HorizontalBitPitch, BitsPerPixel, DoCopy, ClearProc :)

Erzeugen eines Bildes mit rechteckiger Domäne aus einem Zeiger auf Pixel (mit Speicherverwaltung).

Der Operator gen_image1_rect erzeugt ein Bild in der Größe (VerticalPitch/(HorizontalBitPitch/8)) * Height. Die Pixel in PixelPointer werden zeilensequentiell abgelegt. Wie ein Zeigerwert übergeben wird hängt von der benutzten Operatorsignatur und der Programmierumgebung ab. Es ist sicherzustellen, dass die tatsächliche Speicheradresse unter der die Bilddaten gespeichert sind übergeben wird und nicht die Adresse der Zeigervariablen. Es ist darauf zu achten, dass 64-Bit-Zeiger auf 64-Bit-Architekturen nicht beschnitten werden. VerticalPitch entspricht dem Abstand (in Bytes) des m-ten Bildpunktes in Zeile n und des m-ten Bildpunktes in Zeile n+1 innerhalb des Speichers und ist für alle Zeilen des 'Eingabebildes' identisch. Die Breite des Ausgabebildes ist VerticalPitch / (HorizontalBitPitch / 8), seine Höhe ist Height. Die Domäne des Ausgabebildes Image ist ein Rechteck der Größe Width * Height. Der Parameter HorizontalBitPitch ist der horizontale Abstand (in Bits) benachbarter Pixel. BitsPerPixel ist die Anzahl der verwendeten Bits pro Pixel.

Ist DoCopy auf *'true'* gesetzt, werden die von PixelPointer referenzierten Bilddaten kopiert und Speicher für das neue Bild wird von HALCON neu angelegt. Andernfalls muss der Speicherbereich auf den PixelPointer zeigt wieder freigegeben werden. Dies geschieht durch die vom Aufrufer zur Verfügung gestellte Prozedur ClearProc. Diese Prozedur muss folgende Signatur haben

void ClearProc(void* ptr);

und wird beim Löschen von Image mit der __cdecl Aufrufkonvention aufgerufen. Falls der Speicher nicht freigegeben werden soll (im Fall von Framegrabbern oder statischem Speicher) kann eine Prozedur "ohne Rumpf" oder der NULL-Pointer übergeben werden. Analog zum Parameter PixelPointer muss der Zeiger auf die Prozedur abhängig von der benutzten Operatorsignatur und der Programmierumgebung übergeben werden. Ist DoCopy gleich *'true'* so ist ClearProc irrelevant. Der Operator gen_image1_rect ist symmetrisch zu get_image_pointer1_rect.

Achtung

gen_image1_rect überprüft nicht, ob PixelPointer einen gültigen Wert hat oder nicht. Es muss also darauf geachtet werden, dass der Wert gültig ist, da es sonst zum Programmabsturz kommen kann!

Parameter

▷ **Image** (output_object) ..... image ⇝ *object :* byte / uint2 / int4
Erzeugtes HALCON-Bild.

▷ **PixelPointer** (input_control) ..... pointer ⇝ *integer*
Zeiger auf das erste Pixel.

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Width` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width >= 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert: 512
Wertevorschläge: `Height` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Height` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height >= 1

▷ **VerticalPitch** (input_control) ..... integer ⇝ *integer*
Abstand (in Bytes) zwischen Pixel m in Zeile n und Pixel m in Zeile n+1 des 'Eingabebildes'.
Restriktion: VerticalPitch >= Width * HorizontalBitPitch / 8

▷ **HorizontalBitPitch** (input_control) ..... integer ⇝ *integer*
Abstand benachbarter Pixel in Bits.
Defaultwert: 8
Werteliste: `HorizontalBitPitch` ∈ {8, 16, 32}

▷ **BitsPerPixel** (input_control) ..... integer ⇝ *integer*
Anzahl verwendeter Bits pro Pixel.
Defaultwert: 8
Werteliste: `BitsPerPixel` ∈ {8, 9, 10, 11, 12, 13, 14, 15, 16, 32}
Restriktion: BitsPerPixel <= HorizontalBitPitch

▷ **DoCopy** (input_control) ..... string ⇝ *string*
Bilddaten kopieren.
Defaultwert: 'false'
Wertevorschläge: `DoCopy` ∈ {'true', 'false'}

▷ **ClearProc** (input_control) ..... pointer ⇝ *integer*
Zeiger auf die Prozedur, die den Speicher des Bildes beim Löschen des Objektes wieder freigibt.
Defaultwert: 0

Beispiel

```
void NewImage(Hobject *new)
{
  unsigned char       *image;
  int                 r,c;

  image = malloc(640*480);
  for (r=0; r<480; r++)
    for (c=0; c<640; c++)
      image[r*640+c] = c % 255;
  gen_image1_rect(new,(Hlong)image,400,480,640,8,8,'false',(long)free);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image1_rect` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`get_image_pointer1_rect`

Alternativen

`gen_image1`, `gen_image1_extern`, `gen_image_const`

Siehe auch

`get_image_pointer1_rect`

Modul

Foundation

gen_image3 (: ImageRGB : Type, Width, Height, PixelPointerRed, PixelPointerGreen, PixelPointerBlue :)

Erzeugen eines Bildes aus drei Zeigern auf die Pixel (rot/grün/blau).

`gen_image3` erzeugt ein dreikanaliges Bild in der Größe `Width` × `Height`. Die Pixel in `PixelPointerRed`, `PixelPointerGreen` und `PixelPointerBlue` sind zeilensequentiell abgelegt. Der Typ der übergebenen Pixel (`PixelPointerRed` etc.) muss mit `Type` übereinstimmen (siehe `gen_image_const` für eine detailliertere Beschreibung der Pixeltypen). Der Speicher für das neue Bild wird von HALCON neu angelegt. Er kann also nach dem Aufruf freigegeben werden. Wie ein Zeigerwert übergeben wird hängt von der benutzten Operatorsignatur und der Programmierumgebung ab. Es ist sicherzustellen, dass die tatsächliche Speicheradresse unter der die Bilddaten gespeichert sind übergeben wird und nicht die Adresse der Zeigervariablen. Es ist darauf zu achten, dass 64-Bit-Zeiger auf 64-Bit-Architekturen nicht beschnitten werden.

Achtung

`gen_image3_extern` überprüft nicht, ob die Pixel in `PixelPointerRed`, `PixelPointerGreen` und `PixelPointerBlue` einen gültigen Wert haben oder nicht. Es muss also darauf geachtet werden, dass diese Werte gültig sind, da es sonst zum Programmabsturz kommen kann!

Parameter

▷ **ImageRGB** (output_object) . . . . . image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Erzeugtes Bild mit neuer Bildmatrix.

▷ **Type** (input_control) . . . . . string ⇝ *string*
Pixeltyp.
Defaultwert: 'byte'
Werteliste: `Type` ∈ {'byte', 'direction', 'cyclic', 'int1', 'int2', 'uint2', 'int4', 'real'}

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Width` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert: 512
Wertevorschläge: `Height` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Height` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **PixelPointerRed** (input_control) . . . . . pointer ⇝ *integer*
Zeiger auf den ersten Rot-Wert (Kanal 1).

▷ **PixelPointerGreen** (input_control) . . . . . pointer ⇝ *integer*
Zeiger auf den ersten Grün-Wert (Kanal 2).

▷ **PixelPointerBlue** (input_control) . . . . . pointer ⇝ *integer*
Zeiger auf den ersten Blau-Wert (Kanal 3).

Beispiel

```
void NewRGBImage(Hobject *new)
{
  unsigned char  red[768*525];
  unsigned char  green[768*525];
  unsigned char  blue[768*525];
  int            r,c;
  for (r=0; r<525; r++)
    for (c=0; c<768; c++)
    {
      red[r*768+c]   = c % 255;
      green[r*768+c] = (767 - c) % 255;
      blue[r*768+c]  = r % 255;
    }
    gen_image3(new,"byte",768,525,(Hlong)red,(long)green,(long)blue);
}

main()
{
  Hobject  rgb;
  open_window(0,0,768,525,0,"","",&WindowHandle);
  NewRGBImage(&rgb);
  disp_color(rgb,WindowHandle);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image3` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_image_const`, `get_image_pointer1`

Nachfolger

`disp_color`

Alternativen

`gen_image1`, `compose3`, `gen_image_const`

Siehe auch

`reduce_domain`, `paint_gray`, `paint_region`, `set_grayval`, `get_image_pointer1`, `decompose3`

Modul

Foundation

gen_image3_extern (: Image : Type, Width, Height, PointerRed, PointerGreen, PointerBlue, ClearProc :)

Erzeugen eines dreikanaligen Bildes aus drei Zeigern auf Pixel (mit Speicherverwaltung).

gen_image3_extern erzeugt ein Dreikanalbild in der Größe Width × Height. Die Pixel in PointerRed, PointerGreen und PointerBlue sind zeilensequentiell abgelegt. Der Typ der übergebenen Pixel muss mit Type übereinstimmen (siehe gen_image_const für eine detailliertere Beschreibung der Pixeltypen). Wie ein Zeigerwert übergeben wird hängt von der benutzten Operatorsignatur und der Programmierumgebung ab. Es ist sicherzustellen, dass die tatsächliche Speicheradresse unter der die Bilddaten gespeichert sind übergeben wird und nicht die Adresse der Zeigervariablen. Es ist darauf zu achten, dass 64-Bit-Zeiger auf 64-Bit-Architekturen nicht beschnitten werden.

Der Speicher für das neue Bild wird von HALCON im Gegensatz zu gen_image3 nicht neu angelegt und damit auch nicht kopiert. Der Speicherplatz auf den (PointerRed, PointerGreen, PointerBlue) zeigt, muss also beim Löschen des Objektes Image freigegeben werden. Dies geschieht durch die, vom Aufrufer zur Verfügung gestellte, Prozedur ClearProc. Diese Prozedur muss folgende Signatur haben

void ClearProc(void* ptr);

und wird beim Löschen von Image mit der __cdecl Aufrufkonvention aufgerufen. Falls der Speicher nicht freigegeben werden soll (im Fall von Framegrabbern oder statischem Speicher), kann eine Prozedur "ohne Rumpf"oder der NULL-Pointer übergeben werden. Analog zu den Parametern PointerRed, PointerGreen und PointerBlue muss der Zeiger auf die Prozedur abhängig von der benutzten Operatorsignatur und der Programmierumgebung übergeben werden.

Achtung

gen_image3_extern überprüft nicht, ob in PointerRed, PointerGreen und PointerBlue genug Speicher für ein Bild der Größe Width × Height allokiert wurde.

Ebenso überprüft gen_image3_extern nicht, ob die Pixel in PointerRed, PointerGreen und PointerBlue einen gültigen Wert haben oder nicht. Es muss also darauf geachtet werden, dass diese Werte gültig sind, da es sonst zum Programmabsturz kommen kann!

Parameter

- ▷ **Image** (output_object) . . . . . image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
  Erzeugtes HALCON-Bild.
- ▷ **Type** (input_control) . . . . . string ⇝ *string*
  Pixeltyp.
  Defaultwert: 'byte'
  Werteliste: Type ∈ {'int1', 'int2', 'uint2', 'int4', 'byte', 'real', 'direction', 'cyclic'}
- ▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
  Breite des Bildes.
  Defaultwert: 512
  Wertevorschläge: Width ∈ {128, 256, 512, 1024}
  Typischer Wertebereich: 1 ≤ Width ≤ 512 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: Width >= 1
- ▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
  Höhe des Bildes.
  Defaultwert: 512
  Wertevorschläge: Height ∈ {128, 256, 512, 1024}
  Typischer Wertebereich: 1 ≤ Height ≤ 512 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: Height >= 1
- ▷ **PointerRed** (input_control) . . . . . pointer ⇝ *integer*
  Zeiger auf den ersten Grauwert des ersten Kanals.
- ▷ **PointerGreen** (input_control) . . . . . pointer ⇝ *integer*
  Zeiger auf den ersten Grauwert des zweiten Kanals.
- ▷ **PointerBlue** (input_control) . . . . . pointer ⇝ *integer*
  Zeiger auf den ersten Grauwert des dritten Kanals.
- ▷ **ClearProc** (input_control) . . . . . pointer ⇝ *integer*
  Zeiger auf die Prozedur, die den Speicher des Bildes beim Löschen des Objektes wieder freigibt.
  Defaultwert: 0

Beispiel

```
void NewImage(Hobject *new)
{
  unsigned char  *image_red;
  unsigned char  *image_green;
  unsigned char  *image_blue;
  int            r,c;
  image_red = malloc(640*480);
  image_green = malloc(640*480);
  image_blue = malloc(640*480);
  for (r=0; r<480; r++)
    for (c=0; c<640; c++)
    {
      image_red[r*640+c] = c % 255;
      image_green[r*640+c] = (c+64) % 255;
      image_blue[r*640+c] = (c+128) % 255;
    }
  gen_image3_extern(new,"byte",640,480,(Hlong)image_red,\
(Hlong)image_green,(Hlong)image_blue,(Hlong)free);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image3_extern` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`gen_image3`, `gen_image_const`, `get_image_pointer3`, `gen_image1_extern`

Siehe auch

`reduce_domain`, `paint_gray`, `paint_region`, `set_grayval`

Modul

Foundation

gen_image_const (: Image : Type, Width, Height :)

Erzeugen eines Bildes mit konstantem Grauwert.

`gen_image_const` erzeugt ein Bild in der angegebenen Größe. Die Breite und Höhe des Bildes wird mit `Width` und `Height` festgelegt. HALCON unterstützt folgende Bildtypen:

'byte' 1 Byte pro Pixel, ohne Vorzeichen
Wertebereich: (0..255)

'int1' 1 Byte pro Pixel, mit Vorzeichen
Wertebereich: (-128..127)

'uint2' 2 Byte pro Pixel, ohne Vorzeichen
Wertebereich: (0..65535)

'int2' 2 Byte pro Pixel, mit Vorzeichen
Wertebereich: (-32768..32767)

'int4' 4 Byte pro Pixel, mit Vorzeichen
Wertebereich: (-2147483648..2147483647)

'int8' 8 Byte pro Pixel, mit Vorzeichen (nur auf 64-Bit-Systemen verfügbar)
Wertebereich: (-9223372036854775808..9223372036854775807)

'real' 4 Byte pro Pixel, Fließkomma
Wertebereich: (-3.4e38..3.4e38)
Genauigkeit: 6 Stellen

'complex' zwei Matrizen vom Typ *'real'*

'vector_field_relative' zwei Matrizen vom Typ *'real'*
Interpretation: Vektoren

'vector_field_absolute' zwei Matrizen vom Typ *'real'*
Interpretation: Absolute Koordinaten

'direction' 1 Byte pro Pixel, ohne Vorzeichen
Wertebereich: (0..179)
Interpretation: Halber Winkel
Achtung: Die Werte 180..254 werden auf den Wert 255 gesetzt, der als undefinierter Winkel interpretiert wird.

'cyclic' 1 Byte pro Pixel, ohne Vorzeichen, zyklische Arithmetik
Wertebereich: (0..255)

Ob das erzeugte Bild mit 0 initialisiert wird oder nicht, wird mit `set_system('init_new_image',<'true'/'false'>)` festgelegt.

Parameter

▷ **`Image`** (output_object) ..... image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Erzeugtes Bild mit neuer Bildmatrix.

▷ **`Type`** (input_control) ..... string ⇝ *string*
Pixeltyp.
Defaultwert: 'byte'
Werteliste: `Type` ∈ {'byte', 'direction', 'cyclic', 'int1', 'int2', 'uint2', 'int4', 'int8', 'real', 'complex', 'vector_field_absolute', 'vector_field_relative'}

▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Width` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width >= 1

▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert: 512
Wertevorschläge: `Height` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Height` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height >= 1

Beispiel

```
gen_image_const(&New,"byte",width,height);
get_image_pointer1(New,(Hlong*)&pointer,type,&width,&height);
for (row=0; row<height-1; row++)
{
```

```
  for (col=0; col<width-1; col++)
  {
    pointer[row*width+col] = (row + col) % 256;
  }
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert`gen_image_const` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`paint_region`, `reduce_domain`, `get_image_pointer1`, `copy_obj`

Alternativen

`gen_image1`, `gen_image3`

Siehe auch

`reduce_domain`, `paint_gray`, `paint_region`, `set_grayval`, `get_image_pointer1`

Modul

Foundation

gen_image_gray_ramp (: ImageGrayRamp : Alpha, Beta, Mean, Row, Column, Width, Height :)

Erzeugen einer Grauwertrampe.

`gen_image_gray_ramp` erzeugt eine Grauwertrampe nach folgender Gleichung:

$$\texttt{ImageGrayRamp}(r,c) = \texttt{Alpha}(r - \texttt{Row}) + \texttt{Beta}(c - \texttt{Column}) + \texttt{Mean}$$

Die Größe des Bildes wird mit `Width` und `Height` bestimmt. Die Grauwerte sind vom Type *'byte'*. Grauwerte außerhalb des gültigen Bereichs werden beschnitten.

Parameter

▷ **ImageGrayRamp** (output_object) ..... image ⇝ *object :* byte
Erzeugtes Bild mit neuer Bildmatrix.

▷ **Alpha** (input_control) ..... real ⇝ *real*
Steigung in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: `Alpha` ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Beta** (input_control) ..... real ⇝ *real*
Steigung in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: `Beta` ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Mean** (input_control) ..... real ⇝ *real*
Mittlerer Grauwert.
Defaultwert: 128
Wertevorschläge: `Mean` ∈ {0.0, 20.0, 40.0, 60.0, 80.0, 100.0, 120.0, 140.0, 160.0, 180.0, 200.0, 220.0, 255.0}
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Bezugspunktes.
Defaultwert: 256
Wertevorschläge: `Row` ∈ {128, 256, 512, 1024}
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Bezugspunktes.
Defaultwert: 256
Wertevorschläge: `Column` ∈ {128, 256, 512, 1024}
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Width` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width >= 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert: 512
Wertevorschläge: `Height` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Height` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height >= 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert`gen_image_gray_ramp` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`moments_gray_plane`

Nachfolger

`paint_region`, `reduce_domain`, `get_image_pointer1`, `copy_obj`

Alternativen

`gen_image1`

Siehe auch

`reduce_domain`, `paint_gray`, `gen_image_const`

Modul

Foundation

gen_image_interleaved (: ImageRGB : PixelPointer, ColorFormat, OriginalWidth, OriginalHeight, Alignment, Type, ImageWidth, ImageHeight, StartRow, StartColumn, BitsPerChannel, BitShift :)

Erzeugen eines dreikanaligen Bildes aus einem Zeiger auf die verschränkten Pixel.

gen_image_interleaved erzeugt ein dreikanaliges Bild aus einem Eingabebild, dessen Pixel in PixelPointer zeilensequentiell abgelegt sind. Die Größe des Eingabebildes muss in OriginalWidth und OriginalHeight übergeben werden, das Format der verschränkten Pixel in ColorFormat.

Das Ausgabebild bekommt die Größe ImageWidth × ImageHeight. In Verbindung mit den Koordinaten der linken, oberen Ecke StartRow und StartColumn lässt sich jeder beliebige Ausschnitt innerhalb des Eingabebildes erzeugen. Wird an ImageWidth, ImageHeight, StartRow und StartColumn jeweils *0* übergeben, so erhält das Ausgabebild die gleichen Abmessungen wie das Eingabebild.

Es ist zu beachten, dass der Typ Type (siehe gen_image_const) des Ausgabebildes ImageRGB so gewählt sein muss, dass sich die Farbwerte aus dem Eingabebild in ihrem vollen Umfang darstellen lassen. D.h. gen_image_interleaved erlaubt es nicht, aus einem Eingabebild mit *'rgb48'* Pixeln ein byte Bild zu erzeugen.

Wird bei den Formaten *'rgb48'*, *'bgr48'*, *'rgbx64'* und *'bgr64'* nicht der volle Umfang von 16 Bit pro Pixel und Kanal ausgeschöpft, so sollte in BitsPerChannel die tatsächlich verwendete Anzahl von Bits übergeben werden. Außerdem können die Pixelwerte aus dem Eingabebild noch um BitShift Bits nach rechts geshiftet werden.

Bei vierkanaligen Eingabebildern in den Formaten *'rgbx'*, *'bgrx'*, *'rgbx64'* und *'bgrx64'* wird der vierte Kanal ignoriert.

Der Speicher für das neue Bild wird von HALCON neu angelegt. Er kann also nach dem Aufruf freigegeben werden. Wie ein Zeigerwert übergeben wird hängt von der benutzten Operatorsignatur und der Programmierumgebung ab. Es ist sicherzustellen, dass die tatsächliche Speicheradresse unter der die Bilddaten gespeichert sind übergeben wird und nicht die Adresse der Zeigervariablen. Es ist darauf zu achten, dass 64-Bit-Zeiger auf 64-Bit-Architekturen nicht beschnitten werden.

Mögliche Werte für ColorFormat:

'rgb555': 16 Bit RGB-Tripel (5 Bit pro Pixel und Kanal), wobei das Auffüllbit (X) das erste Bit ist. Das Bitmuster ist XRRRRRGG|GGGBBBBB.

'bgr555': 16 Bit BGR-Tripel (5 Bit pro Pixel und Kanal), wobei das Auffüllbit (X) das erste Bit ist. Das Bitmuster ist XBBBBBGG|GGGRRRRR.

'rgb5551': 16 Bit RGB-Tripel (5 Bit pro Pixel und Kanal), wobei das Auffüllbit (X) das letzte Bit ist. Das Bitmuster ist RRRRRGGG|GGBBBBBX.

'bgr5551': 16 Bit BGR-Tripel (5 Bit pro Pixel und Kanal), wobei das Auffüllbit (X) das letzte Bit ist. Das Bitmuster ist BBBBBGGG|GGRRRRRX.

'rgb565': 16 Bit RGB-Tripel (5 Bit pro Pixel und Kanal, 6 Bit für den Grünkanal). Das Bitmuster ist RRRRRGGG|GGGBBBBB.

'bgr565': 16 Bit BGR-Tripel (5 Bit pro Pixel und Kanal, 6 Bit für den Grünkanal). Das Bitmuster ist BBBBBGGG|GGGRRRRR.

'rgb': 24 Bit RGB-Tripel (8 Bit pro Pixel und Kanal)

'bgr': 24 Bit BGR-Tripel (8 Bit pro Pixel und Kanal)

'rgbx': 32 Bit RGBX-Quadrupel (8 Bit pro Pixel und Kanal)

'bgrx': 32 Bit BGRX-Quadrupel (8 Bit pro Pixel und Kanal)

'rgb48': 48 Bit RGB-Tripel (16 Bit pro Pixel und Kanal)

'bgr48': 48 Bit BGR-Tripel (16 Bit pro Pixel und Kanal)

'rgbx64': 64 Bit RGBX-Quadrupel (16 Bit pro Pixel und Kanal)

'bgrx64': 64 Bit BGRX-Quadrupel (16 Bit pro Pixel und Kanal)

Die Parameterwerte *'rgb555'*, *'bgr555'*, *'rgb565'*, *'bgr565'*, *'rgb5551'* und *'bgr5551'* können zudem das Suffix 'le' (Little Endian: Niederwertiges Byte wird zuerst erwartet) oder - was der Standard ist, falls das Suffix weggelassen wird - 'be' (Big Endian: Höherwertiges Byte wird zuerst erwartet) erhalten. Zum Beispiel ist das Bitmuster für ColorFormat = *'rgb555'* XRRRRRGG|GGGBBBBB, während das Bitmuster für ColorFormat = *'rgb555le'* GGGBBBBB|XRRRRRGG ist.

Achtung

gen_image_interleaved überprüft nicht, ob PixelPointer einen gültigen Wert hat oder nicht. Es muss also darauf geachtet werden, dass der Wert gültig ist, da es sonst zum Programmabsturz kommen kann!

Parameter

▷ **ImageRGB** (output_object) . . . . . image ⇝ *object :* byte / uint2
Erzeugtes Bild mit neuer Bildmatrix.

▷ **PixelPointer** (input_control) . . . . . pointer ⇝ *integer*
Zeiger auf die verschränkten Pixel.

▷ **ColorFormat** (input_control) . . . . . string ⇝ *string*
Format der Eingabepixel.
Defaultwert: 'rgb'
Werteliste: `ColorFormat` ∈ {'rgb', 'bgr', 'rgbx', 'bgrx', 'rgb48', 'bgr48', 'rgbx64', 'bgrx64', 'rgb555', 'bgr555', 'rgb565', 'bgr565', 'rgb555le', 'bgr555le', 'rgb565le', 'bgr565le', 'rgb555be', 'bgr555be', 'rgb565be', 'bgr565be', 'rgb5551', 'bgr5551', 'rgb5551le', 'bgr5551le', 'rgb5551be', 'bgr5551be'}

▷ **OriginalWidth** (input_control) . . . . . extent.x ⇝ *integer*
Breite des Eingabebildes.
Defaultwert: 512
Wertevorschläge: `OriginalWidth` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ OriginalWidth (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **OriginalHeight** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des Eingabebildes.
Defaultwert: 512
Wertevorschläge: `OriginalHeight` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ OriginalHeight (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Alignment** (input_control) . . . . . integer ⇝ *integer*
Reserviert.

▷ **Type** (input_control) . . . . . string ⇝ *string*
Pixeltyp des Ausgabebildes.
Defaultwert: 'byte'
Werteliste: `Type` ∈ {'byte', 'uint2'}

▷ **ImageWidth** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
Breite des Ausgabebildes.
Defaultwert: 0
Wertevorschläge: `ImageWidth` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 0 ≤ ImageWidth (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **ImageHeight** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
Höhe des Ausgabebildes.
Defaultwert: 0
Wertevorschläge: `ImageHeight` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 0 ≤ ImageHeight (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **StartRow** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilennummer der oberen linken Ecke des gewünschten Bildausschnittes.
Defaultwert: 0
Wertevorschläge: `StartRow` ∈ {-1, 0}

▷ **StartColumn** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltennummer der oberen linken Ecke des gewünschten Bildausschnittes.
Defaultwert: 0
Wertevorschläge: `StartColumn` ∈ {-1, 0}

▷ **BitsPerChannel** (input_control) . . . . . integer ⇝ *integer*
Anzahl benutzter Bits pro Pixel und Kanal des Ausgabebildes (-1: Alle Bits werden benutzt).
Defaultwert: -1
Wertevorschläge: `BitsPerChannel` ∈ {5, 8, 10, 12, 16, -1}

▷ **BitShift** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Bits, um die die Farbwerte der Eingabepixel nach rechts geshiftet werden (nur für uint2 Bilder).
Defaultwert: 0
Wertevorschläge: `BitShift` ∈ {0, 2, 4, 6}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image_interleaved` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`disp_color`

Siehe auch

`reduce_domain`, `paint_gray`, `paint_region`, `set_grayval`, `gen_image_const`

Modul

Foundation

gen_image_proto (Image : ImageCleared : Grayval :)

Erzeugen eines Bildes mit einem konstanten Grauwert.

`gen_image_proto` erzeugt ein Ausgabebild `ImageCleared` mit dem konstanten Grauwert `Grayval`. Ist das Eingabebild vom Typ 'direction', werden Grauwerte im Ausgabebild, die nicht innerhalb des für 'direction'-Bilder erlaubten Wertebereichs liegen, auf den Wert 255 gesetzt, um sie als ungültig zu kennzeichnen. `ImageCleared` hat dieselbe Größe und denselben Pixeltyp wie das Eingabebild `Image`.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Eingabebild.

▷ **ImageCleared** (output_object) . . . . . image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Mit konstantem Wert belegtes Bild.

▷ **Grayval** (input_control) . . . . . number ⇝ *real* / integer
Mit diesem Grauwert wird das Ausgabebild erzeugt.
Defaultwert: 0
Wertevorschläge: `Grayval` ∈ {0, 1, 2, 5, 10, 16, 32, 64, 128, 253, 254, 255}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image_proto` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`set_grayval`, `paint_gray`, `gen_image_const`, `copy_image`

Siehe auch

`get_image_pointer1`

Modul

Foundation

gen_image_surface_first_order (: ImageSurface : Type, Alpha, Beta, Gamma, Row, Column, Width, Height :)

Erzeugen einer Grauwertfläche mit einem Polynom erster Ordnung.

gen_image_surface_first_order erzeugt eine geneigte Grauwertfläche nach folgender Gleichung:

$$\texttt{ImageSurface}(r,c) = \texttt{Alpha}(r - \texttt{Row}) + \texttt{Beta}(c - \texttt{Column}) + \texttt{Gamma}$$

Die Größe des Bildes wird mit Width und Height bestimmt. Die Parameter Row und Column definieren den Referenzpunkt der zu erzeugenden Grauwertfläche. Die Grauwerte sind vom Type Type (siehe gen_image_const für eine detailliertere Beschreibung der Pixeltypen). Grauwerte außerhalb des gültigen Bereichs werden beschnitten.

Parameter

- ▷ **ImageSurface** (output_object) ..... image ⇝ *object :* byte / uint2 / real
  Erzeugtes Bild mit neuer Bildmatrix.
- ▷ **Type** (input_control) ..... string ⇝ *string*
  Pixeltyp.
  Defaultwert: 'byte'
  Werteliste: Type ∈ {'byte', 'uint2', 'real'}
- ▷ **Alpha** (input_control) ..... number ⇝ *real*
  Koeffizient erster Ordnung in vertikaler Richtung.
  Defaultwert: 1.0
  Wertevorschläge: Alpha ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  Minimale Schrittweite: 0.000001
  Empfohlene Schrittweite: -0.005
- ▷ **Beta** (input_control) ..... number ⇝ *real*
  Koeffizient erster Ordnung in horizontaler Richtung.
  Defaultwert: 1.0
  Wertevorschläge: Beta ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  Minimale Schrittweite: 0.000001
  Empfohlene Schrittweite: -0.005
- ▷ **Gamma** (input_control) ..... number ⇝ *real*
  Koeffizient nullter Ordnung.
  Defaultwert: 1.0
  Wertevorschläge: Gamma ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  Minimale Schrittweite: 0.000001
  Empfohlene Schrittweite: -0.005
- ▷ **Row** (input_control) ..... number ⇝ *real*
  Zeilenkoordinate des Referenzpunktes der Fläche.
  Defaultwert: 256.0
  Wertevorschläge: Row ∈ {0.0, 128.0, 256.0, 512.0}
  Minimale Schrittweite: 0.000001
  Empfohlene Schrittweite: -0.005
- ▷ **Column** (input_control) ..... number ⇝ *real*
  Spaltenkoordinate des Referenzpunktes der Fläche.
  Defaultwert: 256.0
  Wertevorschläge: Column ∈ {0.0, 128.0, 256.0, 512.0}
  Minimale Schrittweite: 0.000001
  Empfohlene Schrittweite: -0.005
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Bildes.
  Defaultwert: 512
  Wertevorschläge: Width ∈ {128, 256, 512, 1024}
  Typischer Wertebereich: 1 ≤ Width ≤ 512 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: Width >= 1

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ Height ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height >= 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefertgen_image_surface_first_order den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fit_surface_first_order

Nachfolger

sub_image

Siehe auch

gen_image_gray_ramp, gen_image_surface_second_order, gen_image_const

Modul

Foundation

gen_image_surface_second_order (: ImageSurface : Type, Alpha, Beta, Gamma, Delta, Epsilon, Zeta, Row, Column, Width, Height :)

Erzeugen einer Grauwertfläche mit einem Polynom zweiter Ordnung.

gen_image_surface_second_order erzeugt eine gekrümmte Grauwertfläche nach folgender Gleichung:

$$\begin{aligned} \texttt{ImageSurface}(r,c) &= \texttt{Alpha}(r-\texttt{Row})^2 \\ &+ \texttt{Beta}(c-\texttt{Column})^2 \\ &+ \texttt{Gamma}(r-\texttt{Row})*(c-\texttt{Column}) \\ &+ \texttt{Delta}(r-\texttt{Row}) \\ &+ \texttt{Epsilon}(c-\texttt{Column}) \\ &+ \texttt{Zeta} \end{aligned}$$

Die Größe des Bildes wird mit Width und Height bestimmt. Die Parameter Row und Column definieren den Referenzpunkt der zu erzeugenden Grauwertfläche. Die Grauwerte sind vom Typ Type (siehe gen_image_const für eine detailliertere Beschreibung der Pixeltypen). Grauwerte außerhalb des gültigen Bereichs werden beschnitten.

Parameter

▷ **ImageSurface** (output_object) . . . . . image ⇝ *object :* byte / uint2 / real
Erzeugtes Bild mit neuer Bildmatrix.

▷ **Type** (input_control) . . . . . string ⇝ *string*
Pixeltyp.
Defaultwert: 'byte'
Werteliste: Type ∈ {'byte', 'uint2', 'real'}

▷ **Alpha** (input_control) . . . . . number ⇝ *real*
Koeffizient zweiter Ordnung in vertikaler Richtung.
Defaultwert: 1.0
Wertevorschläge: `Alpha` ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Beta** (input_control) . . . . . number ⇝ *real*
Koeffizient zweiter Ordnung in horizontaler Richtung.
Defaultwert: 1.0
Wertevorschläge: `Beta` ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Gamma** (input_control) . . . . . number ⇝ *real*
Gemischter Koeffizient zweiter Ordnung.
Defaultwert: 1.0
Wertevorschläge: `Gamma` ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Delta** (input_control) . . . . . number ⇝ *real*
Koeffizient erster Ordnung in vertikaler Richtung.
Defaultwert: 1.0
Wertevorschläge: `Delta` ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Epsilon** (input_control) . . . . . number ⇝ *real*
Koeffizient erster Ordnung in horizontaler Richtung.
Defaultwert: 1.0
Wertevorschläge: `Epsilon` ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Zeta** (input_control) . . . . . number ⇝ *real*
Koeffizient nullter Ordnung.
Defaultwert: 1.0
Wertevorschläge: `Zeta` ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Row** (input_control) . . . . . number ⇝ *real*
Zeilenkoordinate des Referenzpunktes der Fläche.
Defaultwert: 256.0
Wertevorschläge: `Row` ∈ {0.0, 128.0, 256.0, 512.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Column** (input_control) . . . . . number ⇝ *real*
Spaltenkoordinate des Referenzpunktes der Fläche.
Defaultwert: 256.0
Wertevorschläge: `Column` ∈ {0.0, 128.0, 256.0, 512.0}
Minimale Schrittweite: 0.000001
Empfohlene Schrittweite: -0.005

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ `Width` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width >= 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ Height ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height >= 1

Beispiel

```
* Adjust an inhomogeneous illumination
* using gen_image_surface_second_order
read_image (Image, 'cap_illumination/cap_illumination_01')
get_image_size (Image, Width, Height)
gen_circle (Circle, 495, 630, 350.5)
difference (Image, Circle, RegionDifference)
fit_surface_second_order (RegionDifference, Image, 'regression', 5, 2, \
                          Alpha, Beta, Gamma, Delta, Epsilon, Zeta)
area_center (RegionDifference, Area, Row, Column)
gen_image_surface_second_order (ImageSurface, 'byte', Alpha, Beta, \
                                Gamma, Delta, Epsilon, Zeta, Row, Column, \
                                Width, Height)
sub_image (Image, ImageSurface, ImageSub, 1, 128)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert`gen_image_surface_second_order` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fit_surface_second_order`

Nachfolger

`sub_image`

Siehe auch

`gen_image_gray_ramp`, `gen_image_surface_first_order`, `gen_image_const`

Modul

Foundation

interleave_channels (
MultichannelImage : InterleavedImage : PixelFormat, RowBytes,
Alpha :)

Erzeugen eines verschachtelten Bildes aus einem Mehrkanalbild.

Der Operator `interleave_channels` erzeugt das verschachtelte Bild `InterleavedImage` aus dem drei- oder vierkanaligen `MultichannelImage`. Das ist hilfreich um ein Farbbild für die Darstellung vorzubereiten, z.B. um es in ein .NET Bitmap zu konvertieren.

Pixelformat

Das Format der Pixel von `InterleavedImage` wird mit `PixelFormat` festgelegt. Mögliche Formate sind *'rgb'*, *'rgba'*, *'argb'*, *'bgr'*, *'bgra'* und *'abgr'*, wobei 'a' für den vierten ('Alpha') Kanal steht. Im Gegensatz zu `gen_image_interleaved` wird dabei LittleEndian als Byteordnung angenommen.

Alphakanal

Der Alphakanal wird auf folgende Art und Weise bestimmt:

Kanäle	Domäne	
3	nicht vorhanden/voll	Der Wert von `Alpha` wird für das gesamte Bild verwendet.
3	eingeschränkt	Innerhalb der Domäne: Alpha wird auf `Alpha` gesetzt. Außerhalb der Domäne: Alpha wird au
4		Alpha wird auf den entsprechenden Wert des vierten Kanals gesetzt, Domäne und `Alpha` werd

Zeilenausrichtung

Die Anzahl der Bytes zwischen dem Anfang von zwei Zeilen im Ausgabebild kann mit `RowBytes` bestimmt werden. So können zum Beispiel Ausrichtungskriterien erfüllt werden. Werden keine Füllbytes benötigt, kann `RowBytes` auf *'match'* gesetzt werden. `RowBytes` muss mindestens so groß sein wie die Anzahl der Bytes in `PixelFormat` mal der Breite von `MultichannelImage`.

Parameter

- ▷ **`MultichannelImage`** (input_object) ..... multichannel-image ⇝ *object :* byte
  Mehrkanaliges Eingabebild.
- ▷ **`InterleavedImage`** (output_object) ..... singlechannelimage ⇝ *object :* byte
  Verschränktes Ausgabebild.
- ▷ **`PixelFormat`** (input_control) ..... string ⇝ *string*
  Zielformat für InterleavedImage.
  Defaultwert: 'rgba'
  Werteliste: `PixelFormat` ∈ {'rgb', 'bgr', 'rgba', 'abgr', 'argb', 'bgra'}
- ▷ **`RowBytes`** (input_control) ..... integer ⇝ *string* / integer
  Anzahl der Bytes in einer Zeile von InterleavedImage.
  Defaultwert: 'match'
  Wertevorschläge: `RowBytes` ∈ {'match'}
- ▷ **`Alpha`** (input_control) ..... integer ⇝ *integer*
  Alphawert für dreikanalige Eingabebilder.
  Defaultwert: 255
  Wertevorschläge: `Alpha` ∈ {255, 0}

Beispiel

```
HTuple type, width, height;
HImage patras = new HImage("patras");

HImage interleaved = patras.InterleaveChannels("argb", "match", 255);
IntPtr ptr = interleaved.GetImagePointer1(out type, out width, out height);

Image img = new Bitmap(width/4, height, width,
                       PixelFormat.Format32bppArgb, ptr);

pictureBox.Image = img;
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Siehe auch

`gen_image_interleaved`

Modul

Foundation

region_to_bin (Region : BinImage : ForegroundGray, BackgroundGray, Width, Height :)

Umwandeln von Regionen in ein binäres byte-Bild.

region_to_bin erzeugt ein *'byte'*-Bild und stellt darin die Eingaberegionen mit dem durch den Eingabe-Steuerparameter ForegroundGray vorgegebenen Grauwert dar. Regionen, die über die Größe des angegebenen Bildes hinausgehen, werden entsprechend beschnitten. Der Hintergrund wird auf BackgroundGray gesetzt.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Enthält die darzustellenden Regionen.
- ▷ **BinImage** (output_object) ..... image ⇝ *object :* byte
  Ergebnisbild der Größe Width × Height mit eingetragenen Regionen.
- ▷ **ForegroundGray** (input_control) ..... integer ⇝ *integer*
  Grauwert für die Darstellung der Regionen.
  Defaultwert: 255
  Wertevorschläge: ForegroundGray ∈ {0, 1, 50, 100, 128, 150, 200, 254, 255}
  Typischer Wertebereich: 0 ≤ ForegroundGray ≤ 255 (lin)
  Empfohlene Schrittweite: 1
- ▷ **BackgroundGray** (input_control) ..... integer ⇝ *integer*
  Grauwert für die Darstellung des Hintergrunds.
  Defaultwert: 0
  Wertevorschläge: BackgroundGray ∈ {0, 1, 50, 100, 128, 150, 200, 254, 255}
  Typischer Wertebereich: 0 ≤ BackgroundGray ≤ 255 (lin)
  Empfohlene Schrittweite: 1
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des zu erstellenden Bildes.
  Defaultwert: 512
  Wertevorschläge: Width ∈ {256, 512, 1024}
  Typischer Wertebereich: 1 ≤ Width ≤ 1024 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 16
  Restriktion: Width >= 1
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des zu erstellenden Bildes.
  Defaultwert: 512
  Wertevorschläge: Height ∈ {256, 512, 1024}
  Typischer Wertebereich: 1 ≤ Height ≤ 1024 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 16
  Restriktion: Height >= 1

Komplexität

$O(2 * \texttt{Height} * \texttt{Width})$.

Ergebnis

region_to_bin liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, connection, regiongrowing, pouring

Nachfolger

`get_grayval`

Alternativen

`region_to_label`, `paint_region`, `set_grayval`

Siehe auch

`gen_image_proto`, `paint_gray`

Modul

Foundation

region_to_label (Region : ImageLabel : Type, Width, Height :)

Eintragen von Regionen in ein (Label-)Bild.

`region_to_label` trägt die Regionen mit ihrem Index $(1..n)$ in eine Bildmatrix ein. Die erste Region erhält den „Grauwert“ 1, die zweite den „Grauwert“ 2 usw. Es werden also nur positive Zahlen verwendet. Bei *'byte'*-Bildern wird der Index modulo 256 genommen.

Regionen, die über die Größe des angegebenen Bildes hinausgehen, werden entsprechend beschnitten. Falls sich Regionen überlappen, so gibt die Reihenfolge vor, welche Region in dem Bild eingetragen wird: Es wird in der Reihenfolge der Regionen in die Matrix gezeichnet. Die zuletzt gezeichnete Region bleibt folglich im Ergebnisbild markiert. Die Regionen können gegebenenfalls mit `expand_region` aufbereitet werden.

Der Hintergrund (keine Regionenpunkte) wird auf 0 gesetzt. Hiermit kann getestet werden, in welchen Bildbereichen keine Regionen vorhanden sind.

Parameter

- ▷ **Region** (input_object) . . . . . region(-array) ⇝ *object*
  Enthält die einzutragenden Regionen.
- ▷ **ImageLabel** (output_object) . . . . . image ⇝ *object :* byte / int2 / int4
  Ergebnisbild der Größe Width × Height mit eingetragenen Regionen.
- ▷ **Type** (input_control) . . . . . string ⇝ *string*
  Pixeltyp
  Defaultwert: 'int2'
  Werteliste: `Type` ∈ {'byte', 'int2', 'int4', 'int8'}
- ▷ **Width** (input_control) . . . . . extent.y ⇝ *integer*
  Breite des zu erstellenden Bildes.
  Defaultwert: 512
  Wertevorschläge: `Width` ∈ {64, 128, 256, 512, 1024}
  Typischer Wertebereich: 1 ≤ `Width` ≤ 1024 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 16
  Restriktion: Width >= 1
- ▷ **Height** (input_control) . . . . . extent.x ⇝ *integer*
  Höhe des zu erstellenden Bildes.
  Defaultwert: 512
  Wertevorschläge: `Height` ∈ {64, 128, 256, 512, 1024}
  Typischer Wertebereich: 1 ≤ `Height` ≤ 1024 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 16
  Restriktion: Height >= 1

Komplexität

$O(2 * \texttt{Height} * \texttt{Width})$.

Ergebnis

`region_to_label` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `expand_region`

Nachfolger

`get_grayval`, `get_image_pointer1`

Alternativen

`region_to_bin`, `paint_region`

Siehe auch

`label_to_region`

Modul

Foundation

region_to_mean (Regions, Image : ImageMean : :)

Einfärben von Regionen mit ihrem mittleren Grauwert.

`region_to_mean` liefert ein Ergebnisbild, in das die Eingaberegionen `Regions` mit ihrem mittleren Grauwert innerhalb des zugrundegelegten Bildes `Image` eingezeichnet wurden. Diese Routine ist insbesondere zur Visualisierung von Segmentierungsergebnissen nützlich.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegionen.
- ▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
  Zugrundeliegendes Grauwertbild.
- ▷ **ImageMean** (output_object) ..... image ⇝ *object :* byte / uint2
  Ausgabebild mit eingefärbten Regionen.

Beispiel

```
read_image(Image,'fabrik')
regiongrowing(Image,Regions,3,3,6,100)
region_to_mean(Regions,Image,Disp)
dev_display(Disp)
dev_set_draw('margin')
dev_display(Regions)
```

Ergebnis

`region_to_mean` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

regiongrowing, connection

Nachfolger

disp_image

Alternativen

paint_region, intensity

Modul

Foundation

6.5 Kanal

access_channel (MultiChannelImage : Image : Channel :)

Zugriff auf einen Kanal eines mehrkanaligen Bildes.

access_channel greift auf einen Kanal des (mehrkanaligen) Eingabebildes zu. Das Ergebnis ist ein einkanaliges Bild. Der Definitionsbereich wird von der Eingabe übernommen. Die Kanäle sind von 1 bis n durchnummeriert. Die Anzahl der Kanäle lässt sich mit count_channels bestimmen.

Parameter

▷ **MultiChannelImage** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

▷ **Image** (output_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ein Kanal von MultiChannelImage.

▷ **Channel** (input_control) ..... channel ⇝ *integer*

Index des Kanals der zugegriffen werden soll.
Defaultwert: 1
Wertevorschläge: Channel ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
Typischer Wertebereich: 1 ≤ Channel

Beispiel

```
read_image(&Color,"patras");    /* read color image */
access_channel(Color,&Red,1);   /* extract red channel */
disp_image(Red,WindowHandle);
```

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

count_channels

Nachfolger

disp_image

Alternativen

decompose2, decompose3, decompose4, decompose5

Siehe auch

count_channels

Modul

Foundation

append_channel (MultiChannelImage, Image : ImageExtended : :)

Erweitern des Bildes um zusätzliche Matrizen (Kanäle).

append_channel hängt die Matrizen des Bildes Image an die Matrizen von MultiChannelImage an. Hierdurch erhält man ein Bild das so viele Matrizen (Kanäle) enthält, wie MultiChannelImage und Image zusammen. Der Definitionsbereich des Ausgabebildes berechnet sich als der Durchschnitt aus den Definitionsbereichen der beiden Eingabebilder. MultiChannelImage darf auch nur eine Region sein, die dann als Definitionsbereich eines Bildes ohne Kanäle betrachtet wird. Beim Erstellen des Mehrkanalbildes werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **MultiChannelImage** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Neu anzufügendes Bild.

▷ **ImageExtended** (output_object) ..... multichannel-image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Um Image erweitertes Bild.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

disp_image

Alternativen

compose2, compose3, compose4, compose5

Modul

Foundation

channels_to_image (Images : MultiChannelImage : :)

Umwandlung von Einkanalbildern in ein Mehrkanalbild

channels_to_image erzeugt aus mehreren einkanaligen Bildern ein Mehrkanalbild. Der neue Definitionsbereich ergibt sich als Durchschnitt der Definitionsbereiche der Eingabebilder. Beim Erstellen des Mehrkanalbildes werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **Images** (input_object) ..... singlechannelimage-array ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Einkanalbilder die zu einem Mehrkanalbild zusammengefasst werden sollen.

▷ **MultiChannelImage** (output_object) ..... multichannel-image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanalbild.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`count_channels`, `disp_image`

Siehe auch

`image_to_channels`

Modul

Foundation

compose2 (Image1, Image2 : MultiChannelImage : :)

Umwandlung von 2 Bildern in ein zweikanaliges Bild.

`compose2` erzeugt aus 2 einkanaligen Bildern ein 2-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder. Beim Erstellen des Mehrkanalbildes werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **Image1** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 1.

▷ **Image2** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 2.

▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`disp_image`

Alternativen

`append_channel`

Siehe auch

`decompose2`

Modul

Foundation

compose3 (Image1, Image2, Image3 : MultiChannelImage : :)

Umwandlung von 3 Bildern in ein dreikanaliges Bild.

compose3 erzeugt aus 3 einkanaligen Bildern ein 3-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder. Beim Erstellen des Mehrkanalbildes werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **Image1** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 1.

▷ **Image2** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 2.

▷ **Image3** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 3.

▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

disp_image

Alternativen

append_channel

Siehe auch

decompose3

Modul

Foundation

compose4 (Image1, Image2, Image3, Image4 : MultiChannelImage : :)

Umwandlung von 4 Bildern in ein vierkanaliges Bild.

compose4 erzeugt aus 4 einkanaligen Bildern ein 4-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder. Beim Erstellen des Mehrkanalbildes werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **Image1** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 1.

▷ **Image2** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 2.

▷ **Image3** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 3.

▷ **Image4** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 4.

▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`disp_image`

Alternativen

`append_channel`

Siehe auch

`decompose4`

Modul

Foundation

compose5 (Image1, Image2, Image3, Image4, Image5 : MultiChannelImage : :)

Umwandlung von 5 Bildern in ein fünfkanaliges Bild.

`compose5` erzeugt aus 5 einkanaligen Bildern ein 5-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder. Beim Erstellen des Mehrkanalbildes werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **Image1** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 1.

▷ **Image2** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 2.

▷ **Image3** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 3.

▷ **Image4** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 4.

▷ **Image5** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 5.

▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`disp_image`

Alternativen

`append_channel`

Siehe auch

`decompose5`

Modul

Foundation

compose6 (Image1, Image2, Image3, Image4, Image5, Image6 : MultiChannelImage : :)

Umwandlung von 6 Bildern in ein sechskanaliges Bild.

`compose6` erzeugt aus 6 einkanaligen Bildern ein 6-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder. Beim Erstellen des Mehrkanalbildes werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **Image1** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 1.

▷ **Image2** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 2.

▷ **Image3** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 3.

▷ **Image4** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 4.

▷ **Image5** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 5.

▷ **Image6** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 6.

▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`disp_image`

Alternativen

`append_channel`

Siehe auch

`decompose6`

Modul

Foundation

`compose7` `( Image1, Image2, Image3, Image4, Image5, Image6, Image7 : MultiChannelImage : : )`

Umwandlung von 7 Bildern in ein siebenkanaliges Bild.

`compose7` erzeugt aus 7 einkanaligen Bildern ein 7-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder. Beim Erstellen des Mehrkanalbildes werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **`Image1`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 1.

▷ **`Image2`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 2.

▷ **`Image3`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 3.

▷ **`Image4`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 4.

▷ **`Image5`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 5.

▷ **`Image6`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 6.

▷ **`Image7`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Eingabebild 7.

▷ **`MultiChannelImage`** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

disp_image

Alternativen

append_channel

Siehe auch

decompose7

Modul

Foundation

count_channels (MultiChannelImage : : : Channels)

Anzahl Kanäle eine Bildes.

count_channels berechnet zu allen Eingabebildern die Anzahl der Kanäle.

Parameter

▷ **MultiChannelImage** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ein- oder Mehrkanaliges Bild.

▷ **Channels** (output_control) ..... integer(-array) ⇝ *integer*

Anzahl Kanäle.

Beispiel

```
read_image(&Color,"patras");
count_channels(Color,&num_channels);
for (i=1; i<=num_channels; i++)
{
  access_channel(Color,&Channel,i);
  disp_image(Channel,WindowHandle);
  clear_obj(Channel);
}
```

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

access_channel, append_channel, disp_image

Siehe auch

append_channel, access_channel

Modul

Foundation

decompose2 (MultiChannelImage : Image1, Image2 : :)

Umwandlung eines zweikanaligen Bildes in zwei Bilder.

decompose2 erzeugt aus einem 2-kanaligen Bild zwei einkanalige Bilder mit gleichem Definitionsbereich. Beim Erstellen der Ausgabebilder werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

▷ **Image1** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 1.

▷ **Image2** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 2.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

count_channels

Nachfolger

disp_image

Alternativen

access_channel, image_to_channels

Siehe auch

compose2

Modul

Foundation

decompose3 (MultiChannelImage : Image1, Image2, Image3 : :)

Umwandlung eines dreikanaligen Bildes in drei Bilder.

decompose3 erzeugt aus einem 3-kanaligen Bild drei einkanalige Bilder mit gleichem Definitionsbereich. Beim Erstellen der Ausgabebilder werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

▷ **Image1** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 1.

▷ **Image2** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 2.

▷ **Image3** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 3.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`count_channels`

Nachfolger

`disp_image`

Alternativen

`access_channel`, `image_to_channels`

Siehe auch

`compose3`

Modul

Foundation

decompose4 (MultiChannelImage : Image1, Image2, Image3, Image4 : :)

Umwandlung eines vierkanaligen Bildes in vier Bilder.

`decompose4` erzeugt aus einem 4-kanaligen Bild vier einkanalige Bilder mit gleichem Definitionsbereich. Beim Erstellen der Ausgabebilder werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

▷ **Image1** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 1.

▷ **Image2** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 2.

▷ **Image3** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 3.

▷ **Image4** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 4.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`count_channels`

Nachfolger

`disp_image`

Alternativen

`access_channel`, `image_to_channels`

Siehe auch

`compose4`

Modul

Foundation

decompose5 (MultiChannelImage : Image1, Image2, Image3, Image4, Image5 : :)

Umwandlung eines fünfkanaligen Bildes in fünf Bilder.

`decompose5` erzeugt aus einem 5-kanaligen Bild fünf einkanalige Bilder mit gleichem Definitionsbereich. Beim Erstellen der Ausgabebilder werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

▷ **Image1** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 1.

▷ **Image2** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 2.

▷ **Image3** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 3.

▷ **Image4** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 4.

▷ **Image5** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 5.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`count_channels`

Nachfolger

`disp_image`

Alternativen

`access_channel`, `image_to_channels`

Siehe auch

`compose5`

Modul

Foundation

decompose6 (MultiChannelImage : Image1, Image2, Image3, Image4, Image5, Image6 : :)

Umwandlung eines sechskanaligen Bildes in sechs Bilder.

`decompose6` erzeugt aus einem 6-kanaligen Bild sechs einkanalige Bilder mit gleichem Definitionsbereich. Beim Erstellen der Ausgabebilder werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

▷ **Image1** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 1.

▷ **Image2** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 2.

▷ **Image3** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 3.

▷ **Image4** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 4.

▷ **Image5** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 5.

▷ **Image6** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 6.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`count_channels`

Nachfolger

`disp_image`

Alternativen

`access_channel`, `image_to_channels`

Siehe auch

`compose6`

Modul

Foundation

`decompose7` `( MultiChannelImage : Image1, Image2, Image3, Image4, Image5, Image6, Image7 : : )`

Umwandlung eines siebenkanaligen Bildes in sieben Bilder.

`decompose7` erzeugt aus einem 7-kanaligen Bild sieben einkanalige Bilder mit gleichem Definitionsbereich. Beim Erstellen der Ausgabebilder werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **`MultiChannelImage`** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanaliges Bild.

▷ **`Image1`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 1.

▷ **`Image2`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 2.

▷ **`Image3`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 3.

▷ **`Image4`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 4.

▷ **`Image5`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 5.

▷ **Image6** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 6.

▷ **Image7** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Ausgabebild 7.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

count_channels

Nachfolger

disp_image

Alternativen

access_channel, image_to_channels

Siehe auch

compose7

Modul

Foundation

image_to_channels (MultiChannelImage : Images : :)

Umwandlung eines Mehrkanalbildes in Einkanalbilder

image_to_channels erzeugt aus dem Mehrkanalbild in MultiChannelImage für jeden Kanal ein einkanaliges Bild. Die Definitionsbereiche werden von dem Eingabebild übernommen. Es werden so viele Bilder erzeugt wie MultiChannelImage Kanäle hat. Beim Erstellen der Ausgabebilder werden die Bilder nicht dupliziert. Es werden neue Bilder angelegt, die Referenzen auf die Eingabebilder enthalten.

Parameter

▷ **MultiChannelImage** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Mehrkanalbild das zerlegt werden soll.

▷ **Images** (output_object) ..... singlechannelimage-array ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Erzeugte Einkanalbilder.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

count_channels

Nachfolger

disp_image

Alternativen

access_channel, decompose2, decompose3, decompose4, decompose5

Siehe auch

channels_to_image

Modul

Foundation

6.6 Manipulation

overpaint_gray (ImageDestination, ImageSource : : :)

Überschreibt die Grauwerte eines Bildes.

overpaint_gray zeichnet die Grauwerte des Bildes aus ImageSource in das Bild in ImageDestination ein. Kopiert werden nur die Grauwerte aus dem Definitionsbereich von ImageSource (siehe reduce_domain).

Als Alternative zu overpaint_gray zeichnet der Operator paint_gray die Grauwerte nicht direkt in ImageDestination ein, sondern gibt das Ergebnis in einem neu erzeugten Bild zurück.

Achtung

overpaint_gray verändert den Inhalt eines bereits bestehenden Bildes (ImageDestination). Zudem können aber auch andere Bildobjekte betroffen sein: Wenn man z.B. ImageDestination mit Hilfe von copy_obj aus einem anderen Bildobjekt erzeugt hat (oder umgekehrt), wird auch dessen Bildmatrix überschrieben. Deshalb sollte overpaint_gray nur zum Einzeichnen in neu erzeugte Bildobjekte verwendet werden. Typische Operatoren hierfür sind z.B. gen_image_const (erzeugt ein neues Bild mit angegebener Größe), gen_image_proto (erzeugt ein Bild mit der Größe eines angegebenen Prototyp-Bildes) oder copy_image (erzeugt ein Bild als Kopie eines angegebenen Bildes).

Parameter

▷ **ImageDestination** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex / vector_field

Eingabebild, in das eingezeichnet werden soll.

▷ **ImageSource** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex / vector_field

Eingabebild, das die zusätzlichen Grauwerte liefert.

Beispiel

```
* Copy a circular part of the image 'monkey' into a new image (New1):

read_image(Image,'monkey')
gen_circle(Circle,200,200,150)
reduce_domain(Image,Circle,Mask)
* New image with black (0) background
gen_image_proto(Image,New1,0.0)
* Copy a part of the image 'monkey' into New1
overpaint_gray(New1,Mask)
```

Ergebnis

Sind die Parameterwerte korrekt, liefert overpaint_gray den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ImageDestination

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`read_image`, `gen_image_const`, `gen_image_proto`

Alternativen

`get_image_pointer1`, `paint_gray`, `set_grayval`, `copy_image`

Siehe auch

`paint_region`, `overpaint_region`

Modul

Foundation

`overpaint_region ( Image, Region : : Grayval, Type : )`

Zeichnet Regionen in ein Bild ein.

`overpaint_region` zeichnet die Regionen aus `Region` in das Bild `Image` mit den vorgegebenen Grauwerten `Grayval` ein. Die Grauwerte können entweder einmal für jeden Bildkanal, geltend für alle Regionen, definiert werden, oder für jede Region und jeden Bildkanal einzeln. Um Letzteres zu definieren, werden die Kanal-Grauwerte g zu jeder Region gruppiert und zu einem Tupel entsprechend der Reihenfolge der Regionen verknüpft, z.B. für ein dreikanaliges Bild:

$$[g(\mathit{channel1},\mathit{region1}), g(\mathit{channel2},\mathit{region1}), g(\mathit{channel3},\mathit{region1}), g(\mathit{channel1},\mathit{region2}), \ldots]$$

Ist das Bild vom Typ 'direction', werden Grauwerte, die nicht innerhalb des für 'direction'-Bilder erlaubten Wertebereichs liegen, auf den Wert 255 gesetzt, um sie als ungültig zu kennzeichnen. Der Parameter `Type` gibt an, ob die Regionen ausgefüllt (*'fill'*) oder nur der Rand (*'margin'*) eingetragen werden soll.

Als Alternative zu `overpaint_region` zeichnet der Operator `paint_region` die Regionen nicht direkt in `Image` ein, sondern gibt das Ergebnis in einem neu erzeugten Bild zurück.

Achtung

`overpaint_region` verändert den Inhalt eines bereits bestehenden Bildes (`Image`). Zudem können aber auch andere Bildobjekte betroffen sein: Wenn man z.B. `Image` mit Hilfe von `copy_obj` aus einem anderen Bildobjekt erzeugt hat (oder umgekehrt), wird auch dessen Bildmatrix überschrieben. Deshalb sollte `overpaint_region` nur zum Einzeichnen in neu erzeugte Bildobjekte verwendet werden. Typische Operatoren hierfür sind z.B. `gen_image_const` (erzeugt ein neues Bild mit angegebener Größe), `gen_image_proto` (erzeugt ein Bild mit der Größe eines angegebenen Prototyp-Bildes) oder `copy_image` (erzeugt ein Bild als Kopie eines angegebenen Bildes).

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object* : byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex
Bild, in das die Regionen eingezeichnet werden.

▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
Regionen, die eingezeichnet werden soll.

▷ **`Grayval`** (input_control) ..... number(-array) ⇝ *real* / integer
Mit diesen Grauwerten werden die Regionen in `Image` eingetragen.
Defaultwert: 255.0
Wertevorschläge: `Grayval` ∈ {0.0, 1.0, 2.0, 5.0, 10.0, 16.0, 32.0, 64.0, 128.0, 253.0, 254.0, 255.0}

▷ **Type** (input_control) ..... string ⇝ *string*
Regionen ausfüllen oder nur den Rand eintragen.
Defaultwert: 'fill'
Werteliste: `Type` ∈ {'fill', 'margin'}

Beispiel

```
* Paint a rectangle into a new image (New1)

gen_rectangle1(Rectangle,100.0,100.0,300.0,300.0)
* generate a black image
gen_image_const(New1,'byte', 768, 576)
* paint a white rectangle
overpaint_region(New1,Rectangle,255.0,'fill')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `overpaint_region` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- Image

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`read_image`, `gen_image_const`, `gen_image_proto`, `reduce_domain`

Alternativen

`set_grayval`, `paint_region`, `paint_xld`

Siehe auch

`reduce_domain`, `set_draw`, `paint_gray`, `overpaint_gray`, `gen_image_const`

Modul

Foundation

paint_gray (ImageSource, ImageDestination : MixedImage : :)

Zeichnet die Grauwerte eines Bildes in ein anderes Bild ein.

`paint_gray` zeichnet die Grauwerte des Bildes aus `ImageSource` in das Bild in `ImageDestination` ein und liefert das resultierende Bild in `MixedImage` zurück. Kopiert werden nur die Grauwerte aus dem Definitionsbereich von `ImageSource` (siehe `reduce_domain`).

Als Alternative zu `paint_gray` zeichnet der Operator `overpaint_gray` die Grauwerte direkt in `ImageDestination` ein.

Parameter

▷ **ImageSource** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex / vector_field
Eingabebild, das die zusätzlichen Grauwerte liefert.

▷ **ImageDestination** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex / vector_field

Eingabebild, in das eingezeichnet werden soll.

▷ **MixedImage** (output_object) ..... image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex / vector_field

Ergebnisbild.

Beispiel

```
* Copy a circular part of the image 'monkey' into the image 'fabrik':

read_image(Image,'monkey')
gen_circle(Circle,200,200,150)
reduce_domain(Image,Circle,Mask)
read_image(Image2,'fabrik')
* Copy a part of the image 'monkey' into 'fabrik'
paint_gray(Mask,Image2,MixedImage)
```

Ergebnis

Sind die Parameterwerte korrekt, liefert `paint_gray` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`, `gen_image_const`, `gen_image_proto`

Alternativen

`get_image_pointer1`, `set_grayval`, `copy_image`, `overpaint_gray`

Siehe auch

`paint_region`, `overpaint_region`

Modul

Foundation

paint_region (Region, Image : ImageResult : Grayval, Type :)

Zeichnet Regionen in ein Bild ein.

`paint_region` zeichnet die Regionen aus `Region` in das Bild `Image` mit den vorgegebenen Grauwerten `Grayval` ein und liefert das Ergebnis in `ImageResult` zurück. Die Grauwerte können entweder einmal für jeden Bildkanal, geltend für alle Regionen, definiert werden, oder für jede Region und jeden Bildkanal einzeln. Um Letzteres zu definieren, werden die Kanal-Grauwerte g zu jeder Region gruppiert und zu einem Tupel entsprechend der Reihenfolge der Regionen verknüpft, z.B. für ein dreikanaliges Bild:

$$[g(channel1,region1), g(channel2,region1), g(channel3,region1), g(channel1,region2), \ldots]$$

Ist das Eingabebild vom Typ 'direction', werden Grauwerte, die nicht innerhalb des für 'direction'-Bilder erlaubten Wertebereichs liegen, auf den Wert 255 gesetzt, um sie als ungültig zu kennzeichnen. Der Parameter `Type` gibt an, ob die Regionen ausgefüllt (*'fill'*) oder nur der Rand (*'margin'*) eingetragen werden soll.

Als Alternative zu `paint_region` zeichnet der Operator `overpaint_region` die Regionen direkt in `Image` ein.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die eingezeichnet werden sollen.

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex
Bild, in das die Regionen eingezeichnet werden.

▷ **ImageResult** (output_object) ..... image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex
Bild, in das die Regionen eingezeichnet wurden.

▷ **Grayval** (input_control) ..... number(-array) ⇝ *real* / integer
Mit diesen Grauwerten werden die Regionen in `ImageResult` eingetragen.
Defaultwert: 255.0
Wertevorschläge: `Grayval` ∈ {0.0, 1.0, 2.0, 5.0, 10.0, 16.0, 32.0, 64.0, 128.0, 253.0, 254.0, 255.0}

▷ **Type** (input_control) ..... string ⇝ *string*
Regionen ausfüllen oder nur den Rand eintragen.
Defaultwert: 'fill'
Werteliste: `Type` ∈ {'fill', 'margin'}

Beispiel

```
* Paint a rectangle into the image 'monkey'

read_image(Image,'monkey')
gen_rectangle1(Rectangle,100.0,100.0,300.0,300.0)
* paint a white rectangle
paint_region(Rectangle,Image,ImageResult,255.0,'fill')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `paint_region` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`, `gen_image_const`, `gen_image_proto`, `reduce_domain`

Alternativen

`set_grayval`, `overpaint_region`, `paint_xld`

Siehe auch

`reduce_domain`, `paint_gray`, `overpaint_gray`, `set_draw`, `gen_image_const`

Modul

Foundation

paint_xld (XLD, Image : ImageResult : Grayval :)

Zeichnet XLD-Objekte in ein Bild ein.

`paint_xld` zeichnet die XLD-Objekte `XLD` vom Typ *Contour* oder *Polygon* in die einzelnen Bildkanäle des Hintergrundbildes `Image` mit den vorgegebenen Grauwerten `Grayval` ein und liefert das Ergebnis in `ImageResult` zurück. Offene XLD-Objekte werden zuvor geschlossen und die eingeschlossene Region gefüllt.

Die Ränder der subpixelgenauen XLD-Objekte werden im Anti-Aliasing-Verfahren auf den Hintergrund gezeichnet.

Beachten Sie, dass XLD-Objekte, die sich selbst schneiden oder bei denen Segmente sehr nah beisammen sind (in benachbarten Pixeln liegen), nicht korrekt gefüllt werden können; es entstehen Artefakte. Der `set_system` Parameter `'neighborhood'` definiert, ob auch diagonal oder nur orthogonal benachbarte Pixel gefüllt werden.

`Grayval` legt den Grauwert fest, mit welchem jedes XLD-Objekt in den entsprechenden Bildkanal gezeichnet werden soll. Die Grauwerte können entweder einmal für jeden Bildkanal, geltend für alle XLD-Objekte, definiert werden, oder für jedes XLD-Objekt und jeden Bildkanal einzeln. Um Letzteres zu definieren, werden die Kanal-Grauwerte g zu jedem XLD-Objekt gruppiert und zu einem Tupel entsprechend der Reihenfolge der XLD-Objekte verknüpft, z.B. für ein dreikanaliges Bild:

$$[g(channel1,xld1), g(channel2,xld1), g(channel3,xld1), g(channel1,xld2), \ldots]$$

Ist das Eingabebild vom Typ 'direction', werden Grauwerte, die nicht innerhalb des für 'direction'-Bilder erlaubten Wertebereichs liegen, auf den Wert 255 gesetzt, um sie als ungültig zu kennzeichnen.

Parameter

- ▷ **`XLD`** (input_object) ..... xld(-array) ⇝ *object*
  XLD-Objekte, die eingezeichnet werden sollen.
- ▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex
  Bild, in das die XLD-Objekte eingezeichnet werden.
- ▷ **`ImageResult`** (output_object) ..... image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex
  Bild, in das die XLD-Objekte eingezeichnet wurden.
- ▷ **`Grayval`** (input_control) ..... number(-array) ⇝ *real* / integer
  Mit diesen Grauwerten werden die XLD-Objekte in `ImageResult` eingetragen.
  Defaultwert: 255.0
  Wertevorschläge: `Grayval` ∈ {0.0, 1.0, 2.0, 5.0, 10.0, 16.0, 32.0, 64.0, 128.0, 253.0, 254.0, 255.0}

Beispiel

```
* Paint colored xld objects into a gray image

* read and copy image to generate a three channel image
read_image(Image1,'green-dot')
copy_image(Image1,Image2)
copy_image(Image1,Image3)
compose3(Image1,Image2,Image3,Image)
* extract subpixel border
threshold_sub_pix(Image1,Border,128)
* select the circle and the arrows
select_obj(Border,circle,14)
select_obj(Border,arrows,16)
concat_obj(circle,arrows,green_dot)
* paint a green circle and white arrows (to paint all
* objects e.g. blue, pass [0,0,255] tuple for GrayVal)
paint_xld(green_dot,Image,ImageResult,[0,255,0,255,255,255])
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `paint_xld` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabe-XLD-Objekte vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_image, gen_image_const, gen_image_proto, gen_contour_polygon_xld, threshold_sub_pix

Alternativen

set_grayval, paint_gray, paint_region

Siehe auch

gen_image_const

Modul

Foundation

set_grayval (Image : : Row, Column, Grayval :)

Setzen von einzelnen Grauwerten in einem Bildobjekt.

set_grayval setzt die Grauwerte des Eingabebildes Image. Grayval ist dabei ein Tupel von Gleitkomma- bzw. ganzen Zahlen, das die Grauwerte enthält, die den ausgewählten Bildpunkten von Image zugewiesen werden sollen. Die Zeilenkoordinaten der Bildpunkte stehen in Row, die Spaltenkoordinaten in Column. Die Grauwerte können entweder einmal für jedes Pixel, geltend für alle Kanäle, definiert werden, oder für jedes Pixel und jeden Bildkanal einzeln. Um Letzteres zu definieren, werden die Kanal-Grauwerte g zu jedem Pixel gruppiert und zu einem Tupel entsprechend der Reihenfolge der Kanäle verknüpft, z.B. für ein dreikanaliges Bild:

$$[g(\mathit{channel0},\mathit{pixel0}), g(\mathit{channel1},\mathit{pixel0}), g(\mathit{channel2},\mathit{pixel0}), g(\mathit{channel0},\mathit{pixel1}), \ldots]$$

Dabei ist zu beachten, dass für komplexe und Vektorfeld-Bilder zwei Grauwerte pro Pixel (pro Kanal) übergeben werden müssen.

Ist das Bild vom Typ 'direction', werden Grauwerte, die nicht innerhalb des für 'direction'-Bilder erlaubten Wertebereichs liegen, auf den Wert 255 gesetzt, um sie als ungültig zu kennzeichnen.

Achtung

Der Aufwand von set_grayval ist relativ hoch. Normalerweise wird der Operator verwendet, um einzelne Grauwerte eines Bildes zu setzen. Für die Programmierung von Bildverarbeitungsoperationen wie Filter ist er nicht geeignet. Hier ist es sinnvoller, den Operator get_image_pointer1 und direkt die C- oder C++-Schnittstelle zum Einbinden von eigenen Prozeduren zu verwenden.

Darüberhinaus ist zu beachten, dass set_grayval den Inhalt eines bereits bestehenden Bildes (Image) verändert. Zudem können aber auch andere Bildobjekte betroffen sein: Wenn z.B. Image mit Hilfe von copy_obj aus einem anderen Bildobjekt erzeugt wurde, wird auch dessen Bildmatrix überschrieben. Deshalb sollte set_grayval nur zum Einzeichnen in neu erzeugte Bildobjekte verwendet werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Dieses Bild wird modifiziert.

▷ **Row** (input_control) ..... point.y(-array) ⇝ *integer*

Zeilenkoordinaten der zu modifizierenden Bildpunkte.
Defaultwert: 0
Wertevorschläge: Row ∈ {0, 10, 50, 127, 255, 511}
Typischer Wertebereich: 0 ≤ Row
Restriktion: 0 <= Row && Row < height(Image)

▷ **Column** (input_control) ..... point.x(-array) ⇝ *integer*

Spaltennummern der zu modifizierenden Bildpunkte.
Defaultwert: 0
Wertevorschläge: Column ∈ {0, 10, 50, 127, 255, 511}
Typischer Wertebereich: 0 ≤ Column
Restriktion: 0 <= Column && Column < width(Image)

▷ **Grayval** (input_control) . . . . . grayval(-array) ⇝ *real* / integer
Zu setzende Grauwerte.
Defaultwert: 255.0
Wertevorschläge: Grayval ∈ {0.0, 1.0, 10.0, 128.0, 255.0}

Ergebnis

Bei korrekter Besetzung der Parameter liefert set_grayval den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- Image

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

read_image, get_image_pointer1, gen_image_proto, gen_image1

Alternativen

get_image_pointer1, paint_gray, paint_region

Siehe auch

get_grayval, gen_image_const, gen_image1, gen_image_proto

Modul

Foundation

6.7 Merkmale

area_center_gray (Regions, Image : : : Area, Row, Column)

Berechnung der Fläche und des Schwerpunktes von Regionen in Grauwertbildern.

area_center_gray berechnet die Fläche und den Schwerpunkt der Eingaberegionen Regions, die die durch das Eingabebild Image definierten Grauwerte besitzen. Dieser Operator verhält sich ähnlich wie der Operator area_center, nur dass hier die Grauwerte zur Berechnung der Fläche und des Schwerpunktes verwendet werden.

Die Fläche A einer Region R im Bild mit den Grauwerten $g(r, c)$ wird definiert durch

$$A = \sum_{(r,c) \in R} g(r, c).$$

Das bedeutet, dass die Fläche durch das Volumen der durch die Grauwerte definierten Funktion $g(r, c)$ definiert wird. Der Schwerpunkt wird durch die ersten zwei normalisierten Momente der Funktion $g(r, c)$ definiert, d.h. durch $(m_{1,0}, m_{0,1})$, wobei

$$m_{p,q} = \frac{1}{A} \sum_{(r,c) \in R} r^p c^q g(r, c).$$

In dem Fall, dass Area Null ist, sind die Koordinaten des Schwerpunkts Row und Column ebenfalls Null.

Achtung

Der Operator area_center_gray berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde. area_center_gray kann auf OpenCL-Geräten ausgeführt werden, sofern die OpenCL-Erweiterungen cl_khr_fp64 and cl_khr_int64_base_atomics vom Gerät unterstützt werden.

Parameter

- ▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
  Zu untersuchende Region(en).
- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
  Grauwertdaten.
- ▷ **Area** (output_control) . . . . . real(-array) ⇝ *real*
  Grauwertvolumen der Region.
- ▷ **Row** (output_control) . . . . . point.y(-array) ⇝ *real*
  Zeilenkoordinate des Grauwert-Schwerpunktes.
- ▷ **Column** (output_control) . . . . . point.x(-array) ⇝ *real*
  Spaltenkoordinate des Grauwert-Schwerpunktes.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `area_center_gray` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`area_center`

Siehe auch

`area_center_xld`, `elliptic_axis_gray`

Modul

Foundation

cooc_feature_image (Regions, Image : : LdGray, Direction : Energy, Correlation, Homogeneity, Contrast)

Berechnung der Co-Occurrence-Matrix und daraus abgeleiteter Grauwertmerkmale.

Der Aufruf von `cooc_feature_image` entspricht der Hintereinanderausführung von `gen_cooc_matrix` und `cooc_feature_matrix`. Wenn mehrere Richtungsmatrizen der Co-Occurrence-Matrix hintereinander ausgewertet werden sollen, ist es günstiger, die Matrix mit `gen_cooc_matrix` zu erstellen und danach `cooc_feature_matrix` für die Ergebnismatrix aufzurufen. Bei dem Parameter `Direction` wird die Richtung der Nachbarschaft in Winkel oder *'mean'* übergeben. Bei *'mean'* wird der Mittelwert aller vier Richtungen berechnet.

Achtung

Der Operator `cooc_feature_image` berücksichtigt nur die mittels `Regions` übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild `Image` festgelegt wurde.

Parameter

- ▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
  Zugehörige Grauwerte.

▷ **LdGray** (input_control) ..... integer ⇝ *integer*
Anzahl der zu unterscheidenden Grauwerte ($2^{\texttt{LdGray}}$).
Defaultwert: 6
Werteliste: `LdGray` ∈ {1, 2, 3, 4, 5, 6, 7, 8}

▷ **Direction** (input_control) ..... integer ⇝ *integer* / string
Auszuwertende Richtung in der Matrix.
Defaultwert: 0
Werteliste: `Direction` ∈ {0, 45, 90, 135, 'mean'}

▷ **Energy** (output_control) ..... real(-array) ⇝ *real*
Energie der Grauwerte.

▷ **Correlation** (output_control) ..... real(-array) ⇝ *real*
Korrelation der Grauwerte.

▷ **Homogeneity** (output_control) ..... real(-array) ⇝ *real*
Lokale Homogenität der Grauwerte.

▷ **Contrast** (output_control) ..... real(-array) ⇝ *real*
Kontrast der Grauwerte.

Ergebnis

`cooc_feature_image` liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten (*'byte'*) eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`gen_cooc_matrix`

Alternativen

`cooc_feature_matrix`

Siehe auch

`intensity`, `min_max_gray`, `entropy_gray`, `select_gray`

Modul

Foundation

cooc_feature_matrix (CoocMatrix : : : Energy, Correlation, Homogeneity, Contrast)

Berechnung von Grauwertmerkmalen aus einer Co-Occurrence-Matrix.

Die Prozedur berechnet aus einer Co-Occurence-Matrix (`CoocMatrix`) die Energie (`Energy`), Korrelation (`Correlation`), lokale Homogenität (`Homogeneity`) und den Kontrast (`Contrast`).

`cooc_feature_matrix` berechnet aus dem Teil der mit `gen_cooc_matrix` generierten Eingabe-Matrix, der der mit dem dortigen Parametern $LdGray$ und $Direction$ angegebenen Richtungsmatrix entspricht, die Grauwertmerkmale nach folgenden Formeln:

Energie:

$$Energy = \sum_{i,j=0}^{width} c_{ij}^2$$

(Maß für Bildhomogenität)

Korrelation:

$$Correlation = \frac{\sum_{i,j=0}^{width}(i-u_x)(j-u_y)c_{ij}}{s_x s_y}$$

(Maß für die Grauwertabhängigkeiten)

lokale Homogenität:

$$Homogeneity = \sum_{i,j=0}^{width} \frac{1}{1+(i-j)^2} c_{ij}$$

Kontrast:

$$Contrast = \sum_{i,j=0}^{width} (i-j)^2 c_{ij}$$

(Maß für die Größe der Intensitätsunterschiede)

wobei

$$\begin{array}{rcl} width & = & \text{Breite von } \texttt{CoocMatrix} \\ c_{ij} & = & \text{Eintrag in der Co-Occurrence-Matrix} \\ u_x & = & \sum_{i,j=0}^{width} i * c_{ij} \\ u_y & = & \sum_{i,j=0}^{width} j * c_{ij} \\ s_x^2 & = & \sum_{i,j=0}^{width} (i-u_x)^2 * c_{ij} \\ s_y^2 & = & \sum_{i,j=0}^{width} (i-u_y)^2 * c_{ij} \end{array}$$

Achtung

Die Region des Eingabebildes wird nicht beachtet.

Parameter

- ▷ **CoocMatrix** (input_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Co-Occurrence-Matrix.
- ▷ **Energy** (output_control) ..... real ⇝ *real*
  Homogenität der Grauwerte.
- ▷ **Correlation** (output_control) ..... real ⇝ *real*
  Korrelation der Grauwerte.
- ▷ **Homogeneity** (output_control) ..... real ⇝ *real*
  Lokale Homogenität der Grauwerte.
- ▷ **Contrast** (output_control) ..... real ⇝ *real*
  Kontrast der Grauwerte.

Ergebnis

`cooc_feature_matrix` liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`gen_cooc_matrix`

Alternativen

`cooc_feature_image`

Siehe auch

`intensity`, `min_max_gray`, `entropy_gray`, `select_gray`

Modul

Foundation

elliptic_axis_gray (Regions, Image : : : Ra, Rb, Phi)

Berechnung der Orientierung und der Hauptachsen von Regionen in Grauwertbildern.

elliptic_axis_gray berechnet die Hauptachsen Ra und Rb und die Orientierung Phi der Ellipse, die die „gleiche Orientierung“ und das “gleiche Seitenverhältnis“ wie die Eingaberegion in Regions haben. Dabei werden die aus dem Bild Image abgeleiteten Grauwertmomente verwendet. Es können auch mehrere Eingaberegionen als Tupel in Regions übergeben werden. Es wird die Länge der großen Halbachse Ra und der kleinen Halbachse Rb sowie die Orientierung der großen Halbachse bezüglich der x-Achse (Phi) bestimmt. Der Winkel wird dabei im Bogenmaß angegeben. Die Berechnung erfolgt dabei analog zu elliptic_axis, nur dass hier Grauwertmomente verwendet werden. Zur Definition der Grauwertmomente siehe area_center_gray.

In dem Fall, dass das Grauwertvolumen der Ellipse Null ist, sind die Ellipsenparameter Ra, Rb und Phi ebenfalls Null.

Achtung

Der Operator elliptic_axis_gray berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Region(en).
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
  Grauwertdaten.
- ▷ **Ra** (output_control) ..... real(-array) ⇝ *real*
  Große Halbachse der Region.
- ▷ **Rb** (output_control) ..... real(-array) ⇝ *real*
  Kleine Halbachse der Region.
- ▷ **Phi** (output_control) ..... angle.rad(-array) ⇝ *real*
  Winkel zwischen der großen Halbachse und der x-Achse.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert elliptic_axis_gray den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Nachfolger

gen_ellipse

Alternativen

elliptic_axis

Siehe auch

area_center_gray

Modul

Foundation

entropy_gray (Regions, Image : : : Entropy, Anisotropy)

Bestimmung der Entropie und der Anisotropie von Bildern.

entropy_gray erstellt das Histogramm der relativen Häufigkeiten der Grauwerte im Eingabebild und errechnet daraus nach folgenden Formeln im Eingabebild und errechnet daraus für jede Region aus Regions nach folgenden Formeln die Entropie und den Anisotropiekoeffizienten:

Entropie:

$$Entropy = -\sum_{0}^{255} rel[i] * \log_2(rel[i])$$

Anisotropiekoeffizient:

$$Anisotropy = \frac{\sum_{0}^{k} rel[i] * \log_2(rel[i])}{Entropy}$$

wobei:

$rel[i]$ = Histogramm der relativen Grauwerthäufigkeiten
i = Grauwert des Eingabebildes $(0 \ldots 255)$
k = Kleinstmöglicher Grauwert mit $\sum_{0}^{k} rel[i] \geq 0.5$

Achtung

Der Operator entropy_gray berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde.

Parameter

- ▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
  Regionen, in denen die Merkmale bestimmt werden sollen.
- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
  Grauwertdaten.
- ▷ **Entropy** (output_control) . . . . . real(-array) ⇝ *real*
  Informationsgehalt (Entropie) der Grauwerte.
  Zusicherung: 0 <= Entropy && Entropy <= 8
- ▷ **Anisotropy** (output_control) . . . . . real(-array) ⇝ *real*
  Bewertung der Symmetrie der Grauwertverteilung.

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F + 255)$.

Ergebnis

entropy_gray liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Alternativen

select_gray

Siehe auch

entropy_image, gray_histo, gray_histo_abs, fuzzy_entropy, fuzzy_perimeter

Modul

Foundation

estimate_noise (Image : : Method, Percent : Sigma)

Schätzung des Bildrauschens aus einem Bild.

Der Operator estimate_noise schätzt die Standardabweichung von additivem Rauschen innerhalb des Definitionsbereiches des in Image übergebenen Bildes.

In folgenden Anwendungsfällen ist der Operator hilfreich:

- Bestimmung von MinContrast, beispielsweise für die Erstellung von Formmodellen mittels create_shape_model,
- Bestimmung der Kantenamplitude bei Kantenfiltern,
- Beurteilung einer Kamera,
- Überwachung der Fehlbedienung einer Kamera (z.B. zu hoher Gain durch den Anwender).

Die folgenden vier Methoden zur Schätzung des Rauschens stehen zur Verfügung:

- *'foerstner'*: Falls für Method *'foerstner'* gewählt wird, wird zunächst für jedes Pixel ein Homogenitätsmaß basierend auf den ersten Ableitungen der Grauwerte von Image berechnet. Durch Anwendung eines Schwellenwertes auf das Homogenitätsmaß erhält man die homogenen Bildbereiche. Der Schwellenwert wird aus einem Näherungswert für das Bildrauschen gewonnen. Als Näherungswert dient der mit Hilfe der Methode *'immerkaer'* (siehe unten) ermittelte Wert. Die Rauschschätzung basiert auf der Annahme, dass die Grauwertschwankungen innerhalb der homogenen Bildbereiche ausschließlich auf das Bildrauschen zurückzuführen sind. Außerdem wird angenommen, dass das Bildrauschen Gaußverteilt ist. Das über die homogenen Bereiche gemittelte Homogenitätsmaß wird dazu genutzt, eine verbesserte Schätzung für das Bildrauschen zu erhalten. Aus der verbesserten Schätzung kann schließlich ein neuer Schwellenwert für das Homogenitätsmaß abgeleitet werden. Der bisher beschriebene Vorgang wird solange wiederholt, bis die verbesserte Schätzung genügend genau ermittelt worden ist. Letztendlich wird die Standardabweichung des geschätzten Bildrauschens in Sigma zurückgeliefert.

  Es ist zu beachten, dass die Iteration in einigen Fällen fälschlicherweise gegen den Wert *0* konvergiert. Dies kann z.B. dann auftreten, wenn Lücken im Grauwerthistogramm des Eingabebildes vorhanden sind. Lücken im Grauwerthistogramm können sowohl durch eine automatische radiometrische Skalierung innerhalb der Kamera oder des Framegrabbers verursacht werden als auch durch eine manuelle Spreizung der Grauwerte im Bild mit einem Skalierungsfaktor > 1.

  Das mit dieser Methode ermittelte Ergebnis ist unabhängig von dem in Percent übergebenen Wert.

- *'immerkaer'*: Bei der Methode *'immerkaer'* wird zunächst folgende Filtermaske auf das Eingabebild angewendet:

$$M = \begin{vmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{vmatrix} .$$

  Der Vorteil dieser Methode ist, dass M weitgehend unabhängig vom Bildinhalt ist und nur vom Bildrauschen beeinflusst wird. Geht man von einem Gaußverteilten Rauschen aus, so kann dessen Standardabweichung schließlich wie folgt ermittelt werden:

$$\texttt{Sigma} = \sqrt{\frac{\pi}{2}} \frac{1}{6N} \sum_{\texttt{Image}} |\texttt{Image} * M| ,$$

  Dabei beschreibt N die Anzahl der Pixel, auf die M angewendet wurde.

  Das mit dieser Methode ermittelte Ergebnis ist unabhängig von dem in Percent übergebenen Wert.

- *'least_squares'*: Falls für Method *'least_squares'* gewählt wird, werden die Grauwertschwankungen relativ zu einer lokal geschätzten Grauwertebene genutzt, um das Bildrauschen zu ermitteln. Dafür wird zunächst ein Homogenitätsmaß basierend auf den ersten Ableitungen der Grauwerte von Image berechnet. Homogene Bildbereiche werden ermittelt, indem die Percent Prozent homogensten Pixel, d.h. Pixel mit kleiner erster Ableitung, im Definitionsbereich des Eingabebildes ausgewählt werden. Für jedes ausgewählte Pixel wird

eine Ebene an die Grauwerte innerhalb einer 3×3-Umgebung angepasst. Die Differenzen zwischen dieser Ebene und den Grauwerten in der 3×3-Umgebung dienen dazu, die Standardabweichung des Rauschens zu ermitteln. Schließlich wird die über alle homogenen Pixel gemittelte Standardabweichung in `Sigma` zurückgegeben.

- *'mean'*: Falls für `Method` *'mean'* gewählt wird, basiert die Ermittlung des Bildrauschens auf der Differenz zwischen dem Eingabebild und einer rauschfreien Version des Eingabebildes. Dafür wird zunächst ein Homogenitätsmaß basierend auf den ersten Ableitungen der Grauwerte von `Image` berechnet. Homogene Bildbereiche werden ermittelt, indem die `Percent` Prozent homogensten Pixel, d.h. Pixel mit kleiner erster Ableitung, im Definitionsbereich des Eingabebildes ausgewählt werden. Die anschließende Anwendung eines Mittelwertfilters auf die homogenen Bildregionen dient der Eliminierung des Bildrauschens. Es wird angenommen, dass die Differenz zwischen dem Eingabebild und dessen rauschfreier Version das zu ermittelnde Rauschen enthält. Schließlich wird die Standardabweichung der Differenzen in `Sigma` zurückgegeben. Bei der Anwendung dieses Verfahrens ist darauf zu achten, dass eine zuverlässige Ermittlung des Bildrauschens große zusammenhängende homogene Bildregionen voraussetzt.

Es gilt zu beachten, dass die Methoden *'foerstner'* und *'immerkaer'* von einem Gaußverteilten Bildrauschen ausgehen, während die Methoden *'least_squares'* und *'mean'* auch auf Bilder mit beliebig verteiltem Rauschen angewendet werden können. Allgemein gilt, dass die Methode *'foerstner'* die genauesten Ergebnisse liefert, während die Methode *'immerkaer'* die kürzeste Berechnungszeit aufweist.

Sollte das Bildrauschen nicht zuverlässig geschätzt werden können, wird der Fehler 3175 zurückgegeben. Dies tritt insbesondere dann auf, wenn das Eingabebild nicht genügend homogene Regionen beinhaltet, das Eingabebild künstlich erzeugt wurde oder das Rauschen nicht Gaußverteilt ist. Um diesen Fehler zu vermeiden, kann es in manchen Fällen sinnvoll sein, in Abhängigkeit der verwendeten Methode `Method`, eine der folgenden Änderungen vorzunehmen:

- Vergrößern des Definitionsbereiches des Eingabebildes (bei allen Methoden anwendbar).
- Erhöhen des Wertes für den Parameter `Percent` (bei den Methoden *'least_squares'* und *'mean'* anwendbar).
- Verwenden der Methode *'immerkaer'* anstatt der Methoden *'foerstner'*, *'least_squares'*, oder *'mean'*. Die Methode *'immerkaer'* ist nicht auf das Vorhandensein homogener Bildregionen angewiesen und ist somit in fast allen Fällen anwendbar.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **`Image`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2
Eingabebild.

▷ **`Method`** (input_control) ..... string ⇝ *string*
Methode zur Schätzung des Bildrauschens.
Defaultwert: 'foerstner'
Werteliste: `Method` ∈ {'foerstner', 'immerkaer', 'least_squares', 'mean'}

▷ **`Percent`** (input_control) ..... number ⇝ *real* / integer
Prozentsatz der verwendeten Bildpunkte.
Defaultwert: 20
Wertevorschläge: `Percent` ∈ {1, 2, 5, 7, 10, 15, 20, 30, 40, 50}
Restriktion: 0 < Percent && Percent <= 50.

▷ **`Sigma`** (output_control) ..... real(-array) ⇝ *real*
Standardabweichung des Bildrauschens.
Zusicherung: Sigma >= 0

Beispiel

```
read_image (Image, 'combine')
estimate_noise (ImageNoise, 'foerstner', 20, SigmaFoerstner)
estimate_noise (ImageNoise, 'immerkaer', 20, SigmaImmerkaer)
estimate_noise (ImageNoise, 'least_squares', 20, SigmaLeastSquares)
estimate_noise (ImageNoise, 'mean', 20, SigmaMean)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `estimate_noise` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Konnte das Rauschen nicht zuverlässig geschätzt werden, wird die Fehlermeldung 3175 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`grab_image`, `grab_image_async`, `read_image`, `reduce_domain`

Nachfolger

`binomial_filter`, `gauss_filter`, `mean_image`, `smooth_image`

Alternativen

`noise_distribution_mean`, `intensity`, `min_max_gray`

Siehe auch

`gauss_distribution`, `add_noise_distribution`

Literatur

W. Förstner: „Image Preprocessing for Feature Extraction in Digital Intensity, Color and Range Images„, Springer Lecture Notes on Earth Sciences, Summer School on Data Analysis and the Statistical Foundations of Geomatics, 1999

J. Immerkaer: „Fast Noise Variance Estimation„, Computer Vision and Image Understanding, Vol. 64, No. 2, pp. 300-302, 1996

Modul

Foundation

fit_surface_first_order (Regions, Image : : Algorithm, Iterations, ClippingFactor : Alpha, Beta, Gamma)

Berechnung der Grauwertmomente und der Approximation durch eine Fläche erster Ordnung (Ebene).

`fit_surface_first_order` berechnet die Grauwertmomente und die Approximation der Grauwerte durch eine Fläche erster Ordnung. Dazu wird der Abstand zwischen der Fläche und den Grauwerten minimiert. Eine Fläche erster Ordnung wird durch folgende Gleichung beschrieben:

$$\texttt{Image}(r,c) = \texttt{Alpha}(r - r_center) + \texttt{Beta}(c - c_center) + \texttt{Gamma}$$

r_center und c_center sind die Koordinaten des Schwerpunkts des Schnitts der Eingaberegion mit dem vollen Definitionsbereich des Bildes. Bei der Minimierung des Abstands werden die Parameter `Alpha` bis `Gamma` berechnet.

Das gewünschte Approximationsverfahren wird über den Parameter `Algorithm` ausgewählt:

'regression' Standard 'least squares' Anpassung.

'huber' Gewichtete *least squares* Anpassung, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden.

'tukey' Gewichtete *least squares* Anpassung, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden.

Der Parameter `ClippingFactor` (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto stärker ist die Dämpfung. Die Ausreißerdetektion wird iteriert. Der Parameter `Iterations` enthält die Anzahl durchzuführender Iterationen. Ist *'regression'* als `Algorithm` gesetzt wird `Iterations` ignoriert.

Achtung

Der Operator `fit_surface_first_order` berücksichtigt nur die mittels `Regions` übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild `Image` festgelegt wurde.

Parameter

▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
Zu untersuchende Regionen.

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2 / direction / cyclic / real
Zugehörige Grauwerte.

▷ **Algorithm** (input_control) . . . . . string ⇝ *string*
Algorithmus Anpassung.
Defaultwert: 'regression'
Werteliste: `Algorithm` ∈ {'regression', 'huber', 'tukey'}

▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
Maximale Anzahl von Iterationen (unbenutzt bei 'regression').
Defaultwert: 5
Restriktion: Iterations >= 0

▷ **ClippingFactor** (input_control) . . . . . real ⇝ *real*
Clipping Faktor für die Ausreißerdämpfung.
Defaultwert: 2.0
Werteliste: `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
Restriktion: ClippingFactor > 0

▷ **Alpha** (output_control) . . . . . real(-array) ⇝ *real*
Parameter Alpha der Fläche.

▷ **Beta** (output_control) . . . . . real(-array) ⇝ *real*
Parameter Beta der Fläche.

▷ **Gamma** (output_control) . . . . . real(-array) ⇝ *real*
Parameter Gamma der Fläche.

Ergebnis

`fit_surface_first_order` liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten (*'byte'*) eingegeben wird und die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`gen_image_surface_first_order`

Siehe auch

`moments_gray_plane`, `fit_surface_second_order`

Modul

Foundation

fit_surface_second_order (Regions, Image : : Algorithm, Iterations, ClippingFactor : Alpha, Beta, Gamma, Delta, Epsilon, Zeta)

Berechnung der Grauwertmomente und der Approximation durch eine Fläche zweiter Ordnung.

`fit_surface_second_order` berechnet die Grauwertmomente und die Approximation der Grauwerte durch eine Fläche zweiter Ordnung. Dazu wird der Abstand zwischen der Fläche und den Grauwerten minimiert. Eine Fläche zweiter Ordnung wird durch folgende Gleichung beschrieben:

$$\begin{aligned} \texttt{Image}(r,c) &= \texttt{Alpha}(r - r_center)^2 \\ &+ \texttt{Beta}(c - c_center)^2 \end{aligned}$$

$$+ \quad \texttt{Gamma}(r - r_center) * (c - c_center)$$
$$+ \quad \texttt{Delta}(r - r_center)$$
$$+ \quad \texttt{Epsilon}(c - c_center)$$
$$+ \quad \texttt{Zeta}$$

r_center und c_center sind die Koordinaten des Schwerpunkts des Schnitts der Eingaberegion mit dem vollen Definitionsbereich des Bildes. Bei der Minimierung des Abstands werden die Parameter `Alpha` bis `Zeta` berechnet.

Das gewünschte Approximationsverfahren wird über den Parameter `Algorithm` ausgewählt:

'regression' Standard 'least squares' Anpassung.

'huber' Gewichtete *least squares* Geradenanpassung, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden.

'tukey' Gewichtete *least squares* Geradenanpassung, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden.

Der Parameter `ClippingFactor` (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto stärker ist die Dämpfung. Die Ausreißerdetektion wird iteriert. Der Parameter `Iterations` enthält die Anzahl durchzuführender Iterationen. Ist *'regression'* als `Algorithm` gesetzt wird `Iterations` ignoriert.

Achtung

Der Operator `fit_surface_second_order` berücksichtigt nur die mittels `Regions` übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild `Image` festgelegt wurde.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object* : byte / uint2 / direction / cyclic / real
  Zugehörige Grauwerte.
- ▷ **`Algorithm`** (input_control) ..... string ⇝ *string*
  Algorithmus Anpassung.
  Defaultwert: 'regression'
  Werteliste: `Algorithm` ∈ {'regression', 'tukey', 'huber'}
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von Iterationen (unbenutzt bei 'regression').
  Defaultwert: 5
  Restriktion: Iterations >= 0
- ▷ **`ClippingFactor`** (input_control) ..... real ⇝ *real*
  Clipping Faktor für die Ausreißerdämpfung.
  Defaultwert: 2.0
  Werteliste: `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
  Restriktion: ClippingFactor > 0
- ▷ **`Alpha`** (output_control) ..... real(-array) ⇝ *real*
  Parameter Alpha der Fläche.
- ▷ **`Beta`** (output_control) ..... real(-array) ⇝ *real*
  Parameter Beta der Fläche.
- ▷ **`Gamma`** (output_control) ..... real(-array) ⇝ *real*
  Parameter Gamma der Fläche.
- ▷ **`Delta`** (output_control) ..... real(-array) ⇝ *real*
  Parameter Delta der Fläche.
- ▷ **`Epsilon`** (output_control) ..... real(-array) ⇝ *real*
  Parameter Epsilon der Fläche.
- ▷ **`Zeta`** (output_control) ..... real(-array) ⇝ *real*
  Parameter Zeta der Fläche.

Ergebnis

`fit_surface_second_order` liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten (*'byte'*) eingegeben wird und die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`gen_image_surface_second_order`

Siehe auch

`moments_gray_plane`, `fit_surface_first_order`

Modul

Foundation

`fuzzy_entropy` (Regions, Image : : Apar, Cpar : Entropy)

Fuzzy-Entropie von Regionen.

`fuzzy_entropy` berechnet die Entropy einer Fuzzy Menge. Hierbei wird das Bild als eine Fuzzy Menge betrachtet. Die Entropy bewertet, wie gut das Bild ein schwarzes bzw. weißes Bild annähert. Sie wird durch folgende Gleichung definiert:

$$H(X) \quad = \quad \frac{1}{MNln2} \sum_l T_e(l) h(l)$$

wobei $M \times N$ die Dimension des Bildes und $h(l)$ das Histogramm des Bildes ist. Außerdem

$$T_e(l) \quad = \quad -\mu(l) \ln \mu(l) - (1 - \mu(l)) \ln(1 - \mu(l))$$

Hier ist $u(x(m,n))$ eine Zugehörigkeitsfunktion (siehe `fuzzy_perimeter`), die die Fuzzy-Menge definiert. Dabei gelten für die Parameter die gleiche Bedingungen wie in `fuzzy_perimeter`.

Achtung

`fuzzy_entropy` erwartet, dass die Region `Regions` vollständig innerhalb der angegebenen Domäne liegt. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Parameter

▷ **`Regions`** (input_object) ..... region(-array) ⇝ *object*
Regionen in den das Merkmal bestimmt werden soll.

▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte
Bild für die Grauwerte (Fuzzy-Zugehörigkeitswerte).

▷ **`Apar`** (input_control) ..... integer ⇝ *integer*
Anfang der Fuzzyfunktion
Defaultwert: 0
Wertevorschläge: `Apar` ∈ {0, 5, 10, 20, 50, 100}
Typischer Wertebereich: 0 ≤ `Apar` ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5

▷ **`Cpar`** (input_control) ..... integer ⇝ *integer*
Ende der Fuzzyfunktion.
Defaultwert: 255
Wertevorschläge: `Cpar` ∈ {50, 100, 150, 200, 220, 255}
Typischer Wertebereich: 0 ≤ `Cpar` ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: Apar <= Cpar

▷ **`Entropy`** (output_control) ..... real(-array) ⇝ *real*
Fuzzy Entropy einer Region.

Beispiel

```
* To find a Fuzzy Entropy from an Image
read_image(Image,'monkey')
fuzzy_entropy(Trans,Trans,0,255,Entro)
```

Ergebnis

`fuzzy_entropy` liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Siehe auch

`fuzzy_perimeter`

Literatur

M.K. Kundu, S.K. Pal: „Automatic selection of object enhancement operator with quantitative justification based on fuzzy set theoretic measures“; Pattern Recognition Letters 11; 1990; pp. 811-829.

Modul

Foundation

fuzzy_perimeter (Regions, Image : : Apar, Cpar : Perimeter)

Fuzzy-Umfang einer Region.

Mit Hilfe des Merkmals `fuzzy_perimeter` werden die Unterschiede in der Zugehörigkeitswerte zwischen einem Bildpunkt und seinem Nachbarbildpunkt gemessen. Dabei wird der rechte und der untere Nachbar berücksichtigt. Der Umfang ist wie folgt definiert:

$$p(X) = \sum_{m=1}^{M-1}\sum_{n=1}^{N-1} |\mu_X(x_{m,n}) - \mu_X(x_{m,n+1})| + \sum_{m=1}^{M-1}\sum_{n=1}^{N-1} |\mu_X(x_{m,n}) - \mu_X(x_{m+1,n})|$$

wobei $M \times N$ die Dimension des Bildes ist und $u(x(m,n))$ die Zugehörigkeitsfunktion. Bei dieser Implementation wurde die Standard-S Funktion nach Zadeh verwendet und wird wie folgt definiert:

$$\mu_X(x) = \begin{cases} 0, & x \le a \\ 2\left(\frac{x-a}{c-a}\right)^2, & a < x \le b \\ 1 - 2\left(\frac{x-a}{c-a}\right)^2, & b < x \le c \\ 1, & c \le x \end{cases}$$

Für die Parameter a, b und c gelten nachstehende Bedingungen: $b = \frac{a+c}{2}$ ist der Wendepunkt der Funktion, $\Delta b = b - a = c - b$ ist die Bandbreite, für $x = b$ gilt $\mu(x) = 0.5$. In `fuzzy_perimeter` sind die Parameter `Apar` und `Cpar` folgendermaßen definiert: b ist $\frac{\texttt{Apar}+\texttt{Cpar}}{2}$.

Achtung

`fuzzy_perimeter` erwartet, dass die Regionen `Regions` vollständig innerhalb der angegebenen Domäne liegen. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Parameter

▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
Regionen in den das Merkmal bestimmt werden soll.

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
Bild für die Grauwerte (Fuzzy-Zugehörigkeitswerte).

▷ **Apar** (input_control) . . . . . integer ⇝ *integer*
Anfang der Fuzzyfunktion
Defaultwert: 0
Wertevorschläge: `Apar` ∈ {0, 5, 10, 20, 50, 100}
Typischer Wertebereich: 0 ≤ `Apar` ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5

▷ **Cpar** (input_control) . . . . . integer ⇝ *integer*
Ende der Fuzzyfunktion.
Defaultwert: 255
Wertevorschläge: `Cpar` ∈ {50, 100, 150, 200, 220, 255}
Typischer Wertebereich: 0 ≤ `Cpar` ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: Apar <= Cpar

▷ **Perimeter** (output_control) . . . . . real(-array) ⇝ *real*
Fuzzy-Umfang einer Region.

Beispiel

```
* To find a Fuzzy Entropy from an Image
read_image(Image,'monkey')
fuzzy_perimeter(Trans,Trans,0,255,Per)
```

Ergebnis

`fuzzy_perimeter` liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Siehe auch

`fuzzy_entropy`

Literatur

M.K. Kundu, S.K. Pal: „Automatic selection of object enhancement operator with quantitative justification based on fuzzy set theoretic measures“; Pattern Recognition Letters 11; 1990; pp. 811-829.

Modul

Foundation

gen_cooc_matrix (Regions, Image : Matrix : LdGray, Direction :)

Berechnung der Co-Occurrence-Matrix einer Region in einem Bild.

`gen_cooc_matrix` ermittelt aus den Eingaberegionen, wie oft die Grauwerte i und j in einer bestimmten Richtung (*0*, *45*, *90*, *135* Grad) nebeneinander liegen, speichert diese Zahl in der Co-Occurrence-Matrix an den Stellen (i, j) und (j, i) ab (die Matrix ist also symmetrisch) und normiert am Schluss die Matrix mit der Zahl der Einträge. `LdGray` git die Anzahl der zu unterscheidenden Grauwerte (nämlich $2^{\texttt{LdGray}}$) an.

Beispiel:

Eingabebild mit Grauwerten (normiert mit LdGray=2)

$$\begin{matrix} 0 & 0 & 3 \\ 1 & 1 & 2 \\ 1 & 2 & 3 \end{matrix}$$

Co-occurrence Matrizen (nicht normiert)

0: $\begin{matrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 2 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{matrix}$ **135:** $\begin{matrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$ **90:** $\begin{matrix} 0 & 2 & 0 & 0 \\ 2 & 2 & 1 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 0 \end{matrix}$ **45:** $\begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$

Achtung

Der Operator `gen_cooc_matrix` berücksichtigt nur die mittels `Regions` übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild `Image` festgelegt wurde.

Parameter

- ▷ **`Regions`** (input_object) . . . . . region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **`Image`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
  Bild das die Grauwerte liefert.
- ▷ **`Matrix`** (output_object) . . . . . image(-array) ⇝ *object :* real
  Co-Occurrence-Matrix (Matrizen).
- ▷ **`LdGray`** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der zu unterscheidenden Grauwerte ($2^{\texttt{LdGray}}$).
  Defaultwert: 6
  Werteliste: `LdGray` ∈ {1, 2, 3, 4, 5, 6, 7, 8}
  (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **`Direction`** (input_control) . . . . . integer ⇝ *integer*
  Richtung der Nachbarschaftsbeziehung.
  Defaultwert: 0
  Werteliste: `Direction` ∈ {0, 45, 90, 135}

Ergebnis

`gen_cooc_matrix` liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Bilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`draw_region`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `threshold`, `erosion_circle`, `binomial_filter`, `gauss_filter`, `smooth_image`, `sub_image`

Alternativen

`cooc_feature_image`

Siehe auch

`cooc_feature_matrix`

Modul

Foundation

gray_features (Regions, Image : : Features : Value)

Berechnet Grauwertmerkmale von Regionen .

gray_features hat als Eingabe mehrere Regionen (Regions). Für jede dieser Regionen werden die Merkmale (Features) berechnet und in Value zurückgegeben.

Mögliche Werte für Features:

- *'area'*: Volumen des Grauwertgebirges (vgl. area_center_gray)
- *'row'*: Zeilenindex des Schwerpunkts des Grauwertgebirges (vgl. area_center_gray)
- *'column'*: Spaltenindex des Schwerpunkts des Grauwertgebirges (vgl. area_center_gray)
- *'ra'*: Hauptradius der äquivalenten Ellipse (vgl. elliptic_axis_gray)
- *'rb'*: Nebenradius der äquivalenten Ellipse (vgl. elliptic_axis_gray)
- *'phi'*: Hauptradius der äquivalenten Ellipse (vgl. elliptic_axis_gray)
- *'min'*: Minimaler Grauwert (vgl. min_max_gray)
- *'max'*: Maximaler Grauwert (vgl. min_max_gray)
- *'median'*: Zentraler Grauwert (vgl. min_max_gray, mit Percent=*50*)
- *'mean'*: Mittlerer Grauwert (vgl. intensity)
- *'deviation'*: Standardabweichung der Grauwerte (vgl. intensity)
- *'plane_deviation'*: Standardabweichung von der approximierten Grauwertebene (vgl. plane_deviation)
- *'anisotropy'*: Anisotropie (vgl. entropy_gray)
- *'entropy'*: Entropie (vgl. entropy_gray)
- *'fuzzy_entropy'*: Fuzzy-Entropie der Region (vgl. fuzzy_entropy, mit einer Fuzzyfunktion von Apar=*0* bis Cpar=*255*)
- *'fuzzy_perimeter'*: Fuzzy-Umfang der Region (vgl. fuzzy_perimeter, mit einer Fuzzyfunktion von Apar=*0* bis Cpar=*255*)
- *'moments_row'*: Gemischte Grauwertmomente der Zeile (vgl. moments_gray_plane)
- *'moments_column'*: Gemischte Grauwertmomente der Spalte (vgl. moments_gray_plane)
- *'alpha'*: Ebenengleichung, Parameter Alpha (vgl. moments_gray_plane)
- *'beta'*: Ebenengleichung, Parameter Beta (vgl. moments_gray_plane)

Achtung

Einige der Features werden in der Reihenfolge abgearbeitet, in der sie angegeben wurden. Der Operator gray_features berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde.

Parameter

▷ **Regions** (input_object) . . . . . region-array ⇝ *object*
Zu untersuchende Regionen.

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Grauwertdaten.

▷ **Features** (input_control) . . . . . string(-array) ⇝ *string*
Namen der Merkmale.
Defaultwert: 'mean'
Werteliste: Features ∈ {'area', 'row', 'column', 'ra', 'rb', 'phi', 'min', 'max', 'median', 'mean', 'deviation', 'plane_deviation', 'anisotropy', 'entropy', 'fuzzy_entropy', 'fuzzy_perimeter', 'moments_row', 'moments_column', 'alpha', 'beta'}

▷ **Value** (output_control) . . . . . real(-array) ⇝ *real*
Werte der Merkmale.

Komplexität

Sei F die Fläche der Region und N die Anzahl der Merkmale, dann beträgt die Laufzeitkomplexität $O(F * N)$.

Ergebnis

`gray_features` liefert den Wert TRUE, falls das Eingabebild definierte Grauwerten besitzt und die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`connection`, `mean_image`, `entropy_image`, `sobel_amp`, `median_separate`

Nachfolger

`select_gray`, `shape_trans`, `reduce_domain`, `count_obj`

Siehe auch

`select_gray`, `deviation_image`, `entropy_gray`, `intensity`, `mean_image`, `min_max_gray`, `select_obj`

Modul

Foundation

gray_histo (Regions, Image : : : AbsoluteHisto, RelativeHisto)

Verteilung der Grauwerte.

`gray_histo` berechnet für das übergebene Bild (`Image`) innerhalb von `Regions` das absolute (`AbsoluteHisto`) und relative (`RelativeHisto`) Histogramm der Grauwerte.

Beide Histogramme sind Tupel von 256 Werten, die (beginnend mit 0) die Häufigkeiten der einzelnen Grauwerte im Bild enthalten.

`AbsoluteHisto` gibt dabei die absoluten Häufigkeiten der Grauwerte durch ganze Zahlen an, `RelativeHisto` die relativen, d.h. die absoluten Häufigkeiten dividiert durch die Fläche des Bildes, als Gleitpunktzahlen.

'real'- *'int2'*-, *'uint2'*- und *'int4'*-Bilder werden in *'byte'*-Bilder umgewandelt (zuerst wird der größte und der kleinste Grauwert im Bild bestimmt und anschließend werden die Originalgrauwerte linear in den Bereich 0..255 abgebildet) und dann wie oben verarbeitet.

Achtung

Der Operator `gray_histo` berücksichtigt nur die mittels `Regions` übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild `Image` festgelegt wurde.

real-, int2, uint2 und int4-Bilder werden auf 256 Grauwerte reduziert.

Für byte-, int1-, direction- und cyclic-Bilder kann `gray_histo` auf OpenCL-Geräten ausgeführt werden, sofern das Gerät die in OpenCL 1.0 optionalen Erweiterungen cl_khr_local_int32_base_atomics und cl_khr_global_int32_base_atomics unterstützt. Im Gegensatz zu den meisten anderen HALCON Operatoren kann `gray_histo` auf OpenCL-Geräten deutlich langsamer laufen, wenn das Bild `Image` sich nicht komplett innerhalb der Regionen `Regions` befindet.

In HALCON XL ist keine Ausführung mit OpenCL möglich, da atomare 64-Bit-Festkommaarithmetik erforderlich wäre.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Region, in der das Histogramm berechnet werden soll.

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Bild, dessen Grauwertverteilung berechnet werden soll.

▷ **AbsoluteHisto** (output_control) . . . . . histogram-array ⇝ *integer*
Absolute Häufigkeiten der Grauwerte.

▷ **RelativeHisto** (output_control) . . . . . histogram-array ⇝ *real*
Häufigkeiten, normiert auf die Fläche der Region.

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F + 255)$.

Ergebnis

`gray_histo` liefert den Wert TRUE, falls das Bild definierte Grauwerte hat und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`histo_to_thresh`, `gen_region_histo`

Alternativen

`min_max_gray`, `intensity`, `gray_histo_abs`, `gray_histo_range`

Siehe auch

`set_paint`, `histo_2dim`, `scale_image_max`, `entropy_gray`

Modul

Foundation

gray_histo_abs (Regions, Image : : Quantization : AbsoluteHisto)

Verteilung der Grauwerte.

`gray_histo_abs` berechnet für das übergebene Bild (`Image`) innerhalb von `Regions` das absolute (`AbsoluteHisto`)) Histogramm der Grauwerte.

Der Parameter `Quantization` bestimmt die Granularität des Histogramms und gibt an, wie viele Häufigkeitswerte von benachbarte Grauwerten zu einem Häufigkeitswert zusammengefasst werden. Das resultierende Histogramm `AbsoluteHisto` ist ein Tupel, dessen Indizes sich auf die Grauwerte des Eingabebildes `Image` abbilden und dessen Elemente die Grauwerthäufigkeiten enthalten. Die Indizes i der Häufigkeitswerte berechnen sich aus den Grauwerten g und der Quantisierung q daher durch:

$$i = \left\lceil \frac{g + 0.5}{q} \right\rceil \quad \text{für vorzeichenlose Bildtypen,}$$

$$i = \left\lceil \frac{g - (MIN - 0.5)}{q} \right\rceil \quad \text{für vorzeichenbehaftete Bildtypen,}$$

wobei MIN den minimalen Grauwert bezeichnet, beispielsweise -128 für den *int1* Bildtyp. Die Größe des Tupels ergibt sich folglich aus dem Verhältnis vom Grauwertebereich und der Quantelung, für *int2* Bilder und der Quantelung 3.0 beispielsweise zu $\lceil \frac{65536}{3.0} \rceil = 21846$. Der Null-Grauwert wird bei den vorzeichenbehafteten Bildtypen *int1 b.z.w. int2* auf den Index 128 b.z.w. 32768 abgebildet, negative b.z.w. positive Grauwerte haben entsprechend kleinere b.z.w. größere Indizes.

Achtung

Der Operator `gray_histo_abs` berücksichtigt nur die mittels `Regions` übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild `Image` festgelegt wurde.

Parameter

▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
Region, in der das Histogramm berechnet werden soll.

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2
Bild, dessen Grauwertverteilung berechnet werden soll.

▷ **Quantization** (input_control) . . . . . number ⇝ *real* / integer
Quantelung der Grauwerte.
Defaultwert: 1.0
Wertevorschläge: `Quantization` ∈ {1.0, 2.0, 3.0, 5.0, 10.0}
Restriktion: Quantization >= 1.0

▷ **AbsoluteHisto** (output_control) . . . . . histogram-array ⇝ *integer*
Absolute Häufigkeiten der Grauwerte.

Ergebnis

`gray_histo_abs` liefert den Wert TRUE, falls das Bild definierte Grauwerte hat und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`histo_to_thresh`, `gen_region_histo`

Alternativen

`min_max_gray`, `intensity`, `gray_histo`, `gray_histo_range`

Siehe auch

`disp_image`, `histo_2dim`, `scale_image_max`, `entropy_gray`

Modul

Foundation

gray_histo_range (Regions, Image : : Min, Max, NumBins : Histo, BinSize)

Berechne die Grauwertverteilung eines einkanaligen Bildes innerhalb eines bestimmten Grauwertbereichs.

`gray_histo_range` erstellt die Grauwertverteilung `Histo` des einkanaligen Bildes `Image` innerhalb von `Regions` und des Grauwertbereichs [`Min`, `Max`]. Die Werte für `Min` und `Max` werden abgerundet falls das Bild `Image` nicht vom Datentyp *'real'* ist. Der Grauwertbereich wird in `NumBins` gleichgroße Klassen aufgeteilt. Die Größe einer solchen Klasse wird in `BinSize` ausgegeben. Liegt ein Grauwert zwischen zwei Klassen wird er immer der kleineren Klasse zugeordnet. Bei einem Bild vom Datentyp *'real'* ist die Klassengröße $\texttt{BinSize} = \frac{\texttt{Max}-\texttt{Min}}{\texttt{NumBins}}$. Bei einem Bild mit diskreten Grauwerten wird die Klassengröße über $\texttt{BinSize} = \frac{\texttt{Max}-\texttt{Min}+1}{\texttt{NumBins}}$ berechnet. Die Erstellung des Grauwerthistogramms `Histo` über `gray_histo_range` eignet sich vor allem bei Bildern vom Typ *'real'* und bei Bildern vom Typ *'integer'* mit hoher Bittiefe da über *'NumBins'* die Genauigkeit des Histogramms `Histo` beliebig angepasst werden kann.

Achtung

Der Operator `gray_histo_range` berücksichtigt nur die mittels `Regions` übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild `Image` festgelegt wurde.

Bei einem Bild mit diskreten Grauwerten kann es in Abhängigkeit von `BinSize` zu den folgenden Effekten kommen: Für $\texttt{BinSize} > 1$ werden mehrere benachbarte Grauwerte derselben Klasse zugeordnet. Ist `BinSize` dabei keine natürliche Zahl werden die Grauwerte ungleichmäßig auf die Klassen verteilt (Für $\texttt{BinSize} = 1.5$ werden beispielsweise die ersten beiden Grauwerte der ersten Klasse, der dritte Grauwert der zweiten Klasse

und der vierte und fünfte Grauwert der dritten Klasse zugeordnet). Dies macht sich durch einzelne Spitzen im Histogramm `Histo` bemerkbar. Gilt `BinSize` $<$ *1* werden je nach `BinSize` einigen Klassen keine Grauwerte zugeordnet (Für `BinSize` $=$ *0.5* wird beispielsweise der erste Grauwert der ersten Klasse und der zweite Grauwert der dritten Klasse zugeordnet). Das Histogramm `Histo` weist dann einige Lücken auf, was der Strukur eines Kamms ähnelt.

Bei einem Bild vom Datentyp *'real'* werden für `Min` $=$ `Max` alle Pixel des entsprechenden Grauwertes nur der ersten Klasse zugeordnet.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *object*
  Region, in der das Histogramm berechnet werden soll.
- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / cyclic / direction / int1 / uint2 / int2 / int4 / int8 / real
  Eingabebild.
- ▷ **`Min`** (input_control) ..... real ⇝ *real* / integer
  Minimaler Grauwert.
  Defaultwert: 0
  Wertevorschläge: `Min` ∈ {0}
- ▷ **`Max`** (input_control) ..... real ⇝ *real* / integer
  Maximaler Grauwert.
  Defaultwert: 255
  Wertevorschläge: `Max` ∈ {255}
  Restriktion: Max >= Min
- ▷ **`NumBins`** (input_control) ..... integer ⇝ *integer*
  Anzahl Klassen.
  Defaultwert: 256
  Wertevorschläge: `NumBins` ∈ {16, 32, 64, 128, 256}
  Restriktion: NumBins >= 1
- ▷ **`Histo`** (output_control) ..... histogram(-array) ⇝ *integer*
  Zu berechnendes Histogramm.
- ▷ **`BinSize`** (output_control) ..... real ⇝ *real*
  Klassengröße.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gray_histo_range` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`min_max_gray`

Nachfolger

`create_funct_1d_array`

Alternativen

`gray_histo`, `gray_histo_abs`

Siehe auch

`histo_2dim`, `scale_image_max`

Modul

Foundation

`gray_projections` `( Region, Image : : Mode : HorProjection, VertProjection )`

Horizontale und vertikale Grauwertprojektionen.

gray_projections berechnet die horizontalen und vertikalen Grauwertprojektionen, d.h. die Mittelwerte der Grauwerte des Eingabebildes Image in horizontaler und vertikaler Richtung innerhalb der Eingaberegion Region.

Falls Mode = *'simple'* gewählt wird, erfolgt die Projektion entlang der Koordinatenachsen des Bildes, d.h.:

$$\begin{aligned} \texttt{HorProjection}(r) &= \frac{1}{n(r+r')} \sum_{(r+r',c+c')\in\texttt{Region}} \texttt{Image}(r+r',c+c') \\ \texttt{VertProjection}(c) &= \frac{1}{n(c+c')} \sum_{(r+r',c+c')\in\texttt{Region}} \texttt{Image}(r+r',c+c') \end{aligned}$$

Hierbei ist (r', c') der linke obere Eckpunkt des kleinsten umschließenden achsenparallelen Rechtecks der Eingaberegion (siehe smallest_rectangle1) und $n(x)$ die Anzahl der Regionenpunkte in der entsprechenden Zeile $r + r'$ bzw. Spalte $c + c'$. Die horizontale Projektion liefert also eine eindimensionale Funktion, die den vertikalen Grauwertverlauf wiedergibt. Entsprechend liefert die vertikale Projektion eine Funktion, die den horizontalen Grauwertverlauf wiedergibt.

Falls Mode = *'rectangle'* gewählt wird, erfolgt die Projektion entlang der Hauptachsen des kleinsten umschließenden nicht-achsenparallelen Rechtecks der Eingaberegion (siehe smallest_rectangle2). Hierbei entspricht die horizontale Projektionsrichtung der Richtung der großen Hauptachse des Rechtecks, die vertikale Richtung die der kleinen Achse. In diesem Modus werden alle Grauwerte innerhalb des kleinsten umschließenden nicht-achsenparallelen Rechtecks der Eingaberegion zur Berechnung der Projektionen verwendet.

Achtung

Der Operator gray_projections berücksichtigt nur die mittels Region übergebene Region und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde.

Ist $n(x) = 0$, gibt es also keine Regionenpunkte in der entsprechenden Zeile $r + r'$ bzw. Spalte $c + c'$, so wird der entsprechende Wert von HorProjection bzw. VertProjection auf -1 gesetzt.

gray_projections kann für den *'simple'* Modus auf OpenCL Geräten ausgeführt werden, sofern das OpenCL-Gerät die cl_khr_global_int32_base_atomics Erweiterung unterstützt. Für Bilder vom Typ real muss das OpenCL-Gerät die Erweiterungen cl_khr_fp64 und cl_khr_int64_base_atomics unterstützen.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *object*
  Zu untersuchende Region.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / int2 / uint2 / real
  Grauwerte für die Messung.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Art der Projektionsberechnung.
  Defaultwert: 'simple'
  Werteliste: Mode ∈ {'simple', 'rectangle'}
- ▷ **HorProjection** (output_control) ..... real-array ⇝ *real*
  Horizontale Projektion.
- ▷ **VertProjection** (output_control) ..... real-array ⇝ *real*
  Vertikale Projektion.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

1D Metrology

histo_2dim (Regions, ImageCol, ImageRow : Histo2Dim : :)

Histogramm von zweikanaligen Grauwertbildern.

histo_2dim berechnet das 2-dimensionale Histogramm von zwei Bildern innerhalb von Regions. Die Grauwerte von Kanal 1 (ImageCol) werden als Zeilenindex, diejenigen von Kanal 2 (ImageRow) als Spaltenindex interpretiert. Der Grauwert an einem Punkt $P(g1, g2)$ im Ergebnisbild Histo2Dim gibt die Häufigkeit der Grauwertkombination ($g1$,$g2$) an, wobei $g1$ den Zeilen- und $g2$ den Spaltenindex darstellt.

Achtung

Der Operator histo_2dim berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für die Eingabebilder festgelegt wurde.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Region, in der das Histogramm berechnet werden soll.
- ▷ **ImageCol** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1
  Kanal 1.
- ▷ **ImageRow** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1
  Kanal 2.
- ▷ **Histo2Dim** (output_object) ..... image ⇝ *object :* int4
  Zu berechnendes Histogramm.

Beispiel

```
read_image(Image,'monkey')
get_domain (Image, Domain)
gauss_filter (Image, ImageGauss, 7)
histo_2dim(Domain,ImageGauss, Image,Histo2Dim)
dev_display(Histo2Dim)
```

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F + 256^2)$.

Ergebnis

histo_2dim liefert den Wert TRUE, falls die beiden Bilder definierte Grauwerte haben. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

decompose3, decompose2, draw_region

Nachfolger

threshold, class_2dim_sup, pouring, local_max, gray_skeleton

Alternativen

gray_histo, gray_histo_abs

Siehe auch

get_grayval

Modul

Foundation

intensity (Regions, Image : : : Mean, Deviation)

Mittelwert und Abweichung der Grauwerte.

intensity berechnet den Mittelwert und die Abweichung der Grauwerte im Eingabebild innerhalb von Regions. Sei R eine Region, p ein Punkt aus R mit Grauwert $g(p)$ und F die Fläche ($F = |R|$), dann sind die Merkmale definiert durch:

$$\texttt{Mean} := \frac{\sum_{p \in R} g(p)}{F}$$

$$\texttt{Deviation} := \sqrt{\frac{\sum_{p \in R} (g(p) - \texttt{Mean})^2}{F}}$$

Achtung

Der Operator intensity berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde. Es ist zu beachten, dass die Berechnung von Deviation nicht der üblichen Definition folgt. Sie ist hier so festgelegt, dass sie für ein Bild mit nur einem Pixel den Wert 0.0 liefert.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Regionen, deren Merkmale berechnet werden sollen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
  Grauwertdaten.
- ▷ **Mean** (output_control) ..... real(-array) ⇝ *real*
  Mittlerer Grauwert einer Region.
- ▷ **Deviation** (output_control) ..... real(-array) ⇝ *real*
  Abweichung der Grauwerte innerhalb einer Region.

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F)$.

Ergebnis

intensity liefert den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

threshold

Alternativen

select_gray, min_max_gray

Siehe auch

mean_image, gray_histo, gray_histo_abs

Modul

Foundation

min_max_gray (Regions, Image : : Percent : Min, Max, Range)

Minimale und maximale Grauwerte innerhalb von Regionen.

min_max_gray erstellt das Histogramm der absoluten Häufigkeiten der Grauwerte innerhalb von Regions im Eingabebild Image (siehe gray_histo) und berechnet die Anzahl der Bildpunkte, die Percent der Fläche des Eingabebildes entspricht. Daraufhin wird auf beiden Seiten des Histogramms um diese Anzahl von Bildpunkten nach innen gegangen und der kleinste und größte Grauwert bestimmt:

z.B.:

Fläche = 60, Prozent = 5, d.h. 3 Bildpunkte
Histogramm = [2,8,0,7,13,0,0,...,0,10,10,5,3,1,1]
⇒ Maximum = 255, Minimum = 0, Spanne = 255

min_max_gray liefert: Maximum = 253, Minimum = 1, Spanne = 252

Für Bilder vom Typ int4, int8 und real wird die obige Berechnung nicht über Histogramme sondern über Selektion des entsprechenden Grauwertes durchgeführt. Wird Percent auf 50 gesetzt, dann ist Min = Max = Median. Wenn Percent gleich 0 ist, dann wird kein Histogramm berechnet, um die Laufzeit zu verbessern.

Achtung

Der Operator min_max_gray berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Regionen, deren Merkmale berechnet werden sollen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
  Grauwertdaten.
- ▷ **Percent** (input_control) ..... number ⇝ *real* / integer
  Prozentsatz unterhalb (oberhalb) des absoluten Maximums (Minimums).
  Defaultwert: 0
  Wertevorschläge: Percent ∈ {0, 1, 2, 5, 7, 10, 15, 20, 30, 40, 50}
  Restriktion: 0 <= Percent && Percent <= 50
- ▷ **Min** (output_control) ..... real(-array) ⇝ *real*
  „Minimaler“ Grauwert.
- ▷ **Max** (output_control) ..... real(-array) ⇝ *real*
  „Maximaler“ Grauwert.
  Zusicherung: Max >= Min
- ▷ **Range** (output_control) ..... real(-array) ⇝ *real*
  Differenz aus Max und Min.
  Zusicherung: Range >= 0

Beispiel

```
* Threshold segmentation with training region:
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
min_max_gray(Region,Image,5,Min,Max,Range)
threshold(Image,SegmentedRegion,Min,Max)
dev_display(SegmentedRegion)
```

Ergebnis

min_max_gray liefert den Wert TRUE, falls das Eingabebild definierte Grauwerte besitzt und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

draw_region, gen_circle, gen_ellipse, gen_rectangle1, threshold, regiongrowing

Nachfolger

threshold

Alternativen

select_gray, intensity

Siehe auch

gray_histo, scale_image, scale_image_max, learn_ndim_norm

Modul

Foundation

moments_gray_plane (Regions, Image : : : MRow, MCol, Alpha, Beta, Mean)

Berechnung der Grauwertmomente und der Approximation durch eine Ebene.

moments_gray_plane berechnet die Grauwertmomente und die Approximation der Grauwerte durch eine Ebene. Die Berechnung erfolgt nach folgender Formel:

$$\mathtt{MRow} = \frac{1}{F^2} \sum_{(r,c) \in \mathtt{Regions}} (r - \overline{r})(\mathtt{Image}(r,c) - \mathtt{Mean}) \qquad \mathtt{MCol} = \frac{1}{F^2} \sum_{(r,c) \in \mathtt{Regions}} (c - \overline{c})(\mathtt{Image}(r,c) - \mathtt{Mean})$$

$$\mathtt{Alpha} = \frac{\mathtt{MRow} F m_{02} - m_{11} \mathtt{MCol}}{F} m_{20} m_{02} - m_{11}^2 \qquad \mathtt{Beta} = \frac{m_{20} \mathtt{MCol} F - \mathtt{MRow} F m_{11}}{m_{20} m_{02} - m_{11}^2}$$

wobei F die Fläche, $\overline{r}$, $\overline{c}$ der Schwerpunkt, m_{11}, m_{20} und m_{02} die normierten Momente von Regions sind.

Die Parameter Alpha, Beta und Mean beschreiben dabei eine Ebene über der Region:

$$\mathtt{Image}'(r,c) = \mathtt{Alpha}(r - \overline{r}) + \mathtt{Beta}(c - \overline{c}) + \mathtt{Mean}$$

Alpha gibt also die Steigung in Richtung der Zeilenachse (nach „unten“), Beta die Steigung in Richtung der Spaltenachse (nach „rechts“) an.

Achtung

Der Operator moments_gray_plane berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / uint2 / real
  Zugehörige Grauwerte.
- ▷ **MRow** (output_control) ..... real(-array) ⇝ *real*
  Gemischte Momente der Zeile.
- ▷ **MCol** (output_control) ..... real(-array) ⇝ *real*
  Gemischte Momente der Spalte.
- ▷ **Alpha** (output_control) ..... real(-array) ⇝ *real*
  Parameter Alpha der Ebenengleichung.
- ▷ **Beta** (output_control) ..... real(-array) ⇝ *real*
  Parameter Beta der Ebenengleichung.
- ▷ **Mean** (output_control) ..... real(-array) ⇝ *real*
  Mittlerer Grauwert.

Ergebnis

moments_gray_plane liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten (*'byte'*) eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird

mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

draw_region, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, threshold, regiongrowing

Siehe auch

intensity, moments_region_2nd

Literatur

R. Haralick, L. Shapiro; „Computer and Robot Vision“; Addison-Wesley, 1992, pp 75-76

Modul

Foundation

plane_deviation (Regions, Image : : : Deviation)

Abweichung der Grauwerte von der approximierten Bildebene.

plane_deviation berechnet die Abweichung der Grauwerte in Image von der Approximation der Grauwerte durch eine Ebene. Im Gegensatz zur Standardabweichung bei intensity erhalten auch geneigte Grauwertflächen den Wert Null. Die Berechnung der Grauwertebene erfolgt gemäß gen_image_gray_ramp. Sei F die Fläche der Region, α, β, μ die Parameter der Bildebene und (r', c') der Schwerpunkt der Region, dann ist Deviation definiert durch:

$$\texttt{Deviation} = \sqrt{\frac{sum_{(r,c)\in \texttt{Regions}}((\alpha(r-r')+\beta(c-c')+\mu)-\texttt{Image}(r,c))^2}{F}} \; .$$

Achtung

Der Operator plane_deviation berücksichtigt nur die mittels Regions übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde. Es ist zu beachten, dass die Berechnung von Deviation nicht der üblichen Definition folgt. Sie ist hier so festgelegt, dass sie für ein Bild mit nur einem Pixel den Wert 0.0 liefert.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Regionen, deren Abweichung berechnet werden sollen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / cyclic
  Grauwertdaten.
- ▷ **Deviation** (output_control) ..... real(-array) ⇝ *real*
  Abweichung der Grauwerte innerhalb einer Region.

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F)$.

Ergebnis

plane_deviation liefert den Wert TRUE, falls Image vom Typ *'byte'* ist. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Alternativen

intensity, gen_image_gray_ramp, sub_image

Siehe auch

moments_gray_plane

Modul

Foundation

select_gray (Regions, Image : SelectedRegions : Features, Operation, Min, Max :)

Auswahl von Regionen aufgrund von Grauwertmerkmalen.

select_gray hat als Eingabe eine Anzahl von Regionen (Regions). Für jede dieser Regionen werden die Merkmale (Features) berechnet. Wenn jedes (Operation = *'and'*) oder mindestens eines (Operation = *'or'*) der berechneten Merkmale in den für Features festgelegten Grenzen liegt, wird die Region in die Ausgabe SelectedRegions übernommen (dupliziert). Die Grenzen werden von den Parametern Min und Max definiert. Für diese Parameter können dabei auch *'min'* beziehungsweise *'max'* als Werte gesetzt werden, wodurch die Grenze auf den niedrigsten beziehungsweise höchsten für dieses Merkmal möglichen Wert gesetzt wird. Der Parameter Image enthält ein Bild, das die Grauwerte zur Berechnung der Merkmale liefert.

Bedingung:

$$\mathtt{Min}_i \leq \mathtt{Features}_i(\mathtt{Regions}, \mathtt{Image}) \leq \mathtt{Max}_i$$

Mögliche Werte für Features:

- *'area'*: Volumen des Grauwertgebirges (vgl. area_center_gray)
- *'row'*: Zeilenindex des Schwerpunkts des Grauwertgebirges (vgl. area_center_gray)
- *'column'*: Spaltenindex des Schwerpunkts des Grauwertgebirges (vgl. area_center_gray)
- *'ra'*: Hauptradius der äquivalenten Ellipse (vgl. elliptic_axis_gray)
- *'rb'*: Nebenradius der äquivalenten Ellipse (vgl. elliptic_axis_gray)
- *'phi'*: Hauptradius der äquivalenten Ellipse (vgl. elliptic_axis_gray)
- *'min'*: Minimaler Grauwert (vgl. min_max_gray)
- *'max'*: Maximaler Grauwert (vgl. min_max_gray)
- *'median'*: Zentraler Grauwert (vgl. min_max_gray, mit Percent=*50*)
- *'mean'*: Mittlerer Grauwert (vgl. intensity)
- *'deviation'*: Standardabweichung der Grauwerte (vgl. intensity)
- *'plane_deviation'*: Standardabweichung von der approximierten Grauwertebene (vgl. plane_deviation)
- *'anisotropy'*: Anisotropie (vgl. entropy_gray)
- *'entropy'*: Entropie (vgl. entropy_gray)
- *'fuzzy_entropy'*: Fuzzy-Entropie der Region (vgl. fuzzy_entropy, mit einer Fuzzyfunktion von Apar=*0* bis Cpar=*255*)
- *'fuzzy_perimeter'*: Fuzzy-Umfang der Region (vgl. fuzzy_perimeter, mit einer Fuzzyfunktion von Apar=*0* bis Cpar=*255*)
- *'moments_row'*: Gemischte Grauwertmomente der Zeile (vgl. moments_gray_plane)
- *'moments_column'*: Gemischte Grauwertmomente der Spalte (vgl. moments_gray_plane)
- *'alpha'*:Ebenengleichung, Parameter Alpha (vgl. moments_gray_plane)

- *'beta'*: Ebenengleichung, Parameter Beta (vgl. `moments_gray_plane`)

Achtung

Der Operator `select_gray` berücksichtigt nur die mittels `Regions` übergebenen Regionen und ignoriert jede Domäne, die zuvor für das Eingabebild `Image` festgelegt wurde. Wird nur ein Merkmal verwendet, dann ist der Wert von `Operation` bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie eingegeben werden. Es können maximal 100 Merkmale angegeben werden.

Parameter

▷ **`Regions`** (input_object) . . . . . region-array ⇝ *object*
Zu untersuchende Regionen.

▷ **`Image`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Grauwertdaten.

▷ **`SelectedRegions`** (output_object) . . . . . region-array ⇝ *object*
Regionen, deren Merkmale innerhalb der Grenzen liegen.

▷ **`Features`** (input_control) . . . . . string(-array) ⇝ *string*
Namen der Merkmale.
Defaultwert: 'mean'
Werteliste: `Features` ∈ {'area', 'row', 'column', 'ra', 'rb', 'phi', 'min', 'max', 'median', 'mean', 'deviation', 'plane_deviation', 'anisotropy', 'entropy', 'fuzzy_entropy', 'fuzzy_perimeter', 'moments_row', 'moments_column', 'alpha', 'beta'}

▷ **`Operation`** (input_control) . . . . . string ⇝ *string*
Logische Verknüpfung der Merkmale.
Defaultwert: 'and'
Werteliste: `Operation` ∈ {'and', 'or'}

▷ **`Min`** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
Untergrenze(n) der Merkmale oder 'min'.
Defaultwert: 128.0
Wertevorschläge: `Min` ∈ {0.5, 1.0, 10.0, 20.0, 50.0, 128.0, 255.0, 1000.0, 'min'}

▷ **`Max`** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
Obergrenze(n) der Merkmale oder 'max'.
Defaultwert: 255.0
Wertevorschläge: `Max` ∈ {0.5, 1.0, 10.0, 20.0, 50.0, 128.0, 255.0, 1000.0, 'max'}

Komplexität

Sei F die Fläche der Region und N die Anzahl der Merkmale, dann beträgt die Laufzeitkomplexität $O(F * N)$.

Ergebnis

`select_gray` liefert den Wert TRUE, falls das Eingabebild definierte Grauwerten besitzt und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`connection`, `mean_image`, `entropy_image`, `sobel_amp`, `median_separate`

Nachfolger

`select_shape`, `shape_trans`, `reduce_domain`, `count_obj`

Siehe auch

`deviation_image`, `entropy_gray`, `intensity`, `mean_image`, `min_max_gray`, `select_obj`

Modul

Foundation

shape_histo_all (Region, Image : : Feature : AbsoluteHisto, RelativeHisto)

Ausprägung von Merkmalen entlang aller Schwellenwerte.

shape_histo_all führt 255 Schwellenwertoperationen innerhalb von Region mit den Grauwerten von Image durch. Der Eintrag i im Histogramm entspricht dann der Anzahl der Zusammenhangskomponenten/Löcher dieses mit der Schwelle i segmentierten Bildes (Feature = *'connected_components'*, *'holes'*) bzw. dem Mittelwert der Merkmalswerte der so segmentierten Regionen (Feature = *'convexity'*, *'compactness'*, *'ansisometry'*).

Achtung

shape_histo_all erwartet als Eingabe eine Region und genau ein Grauwertbild. Wegen der Mächtigkeit dieser Prozedur ist die Laufzeit von shape_histo_all relativ groß!

Der Operator shape_histo_all berücksichtigt nur die mittels Region übergebene Region und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *object*
  Region, in der die Merkmale betrachtet werden.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object : byte*
  Grauwertdaten.
- ▷ **Feature** (input_control) ..... string ⇝ *string*
  Merkmal, das untersucht werden soll.
  Defaultwert: 'connected_components'
  Werteliste: Feature ∈ {'connected_components', 'convexity', 'compactness', 'anisometry', 'holes'}
- ▷ **AbsoluteHisto** (output_control) ..... histogram-array ⇝ *real / integer*
  Absolute Verteilung des Merkmals.
- ▷ **RelativeHisto** (output_control) ..... histogram-array ⇝ *real*
  Relative Verteilung des Merkmals.

Beispiel

```
* Simulation of shape_histo_all with feature 'connected_components':
* my_shape_histo_all(Region,Image,AbsHisto,RelHisto):
reduce_domain(Region,Image,RegionGray)
for i := 0 to 255 by 1
    threshold(RegionGray,Seg,i,255)
    connect_and_holes (Seg, NumConnected, _)
    AbsHisto[i] := NumConnected
endfor
Sum := 0
for i := 0 to 255 by 1
    Sum := Sum+AbsHisto[i]
endfor
for i := 0 to 255 by 1
    RelHisto[i] := AbsHisto[i]/Sum
endfor
```

Komplexität

Sei F die Fläche der Eingaberegion und N die mittlere Anzahl der Zusammenhangskomponenten, dann beträgt die Laufzeitkomplexität $O(255(F + \sqrt{F}\sqrt{N}))$.

Ergebnis

shape_histo_all liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten eingegeben wird. Das Verhalten bei leerer Eingabe (keine Bilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

histo_to_thresh, threshold, gen_region_histo

Alternativen

shape_histo_point

Siehe auch

compactness, connection, connect_and_holes, convexity, count_obj, entropy_gray, gray_histo

Modul

Foundation

shape_histo_point (Region, Image : : Feature, Row, Column : AbsoluteHisto, RelativeHisto)

Ausprägung von Merkmalen entlang aller Schwellenwerte.

shape_histo_point führt wie shape_histo_all 255 Schwellenwertoperationen innerhalb von Region mit den Grauwerten von Image durch. Im Gegensatz zu shape_histo_all wird hier nur diejenige segmentierte Region betrachtet, die den Punkt (Row, Column) enthält. Der Eintrag i im Histogramm entspricht dann der Anzahl der Löcher dieser mit der Schwelle i segmentierten Region (Feature = *'holes'*) bzw. dem Merkmalswert der Region (Feature = *'convexity'*, *'compactness'*, *'ansisometry'*).

Achtung

Der Operator shape_histo_point berücksichtigt nur die mittels Region übergebene Region und ignoriert jede Domäne, die zuvor für das Eingabebild Image festgelegt wurde.

Parameter

▷ **Region** (input_object) . . . . . region ⇝ *object*
Region, in der die Merkmale betrachtet werden.

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object : byte*
Grauwertdaten.

▷ **Feature** (input_control) . . . . . string ⇝ *string*
Merkmal, das untersucht werden soll.
Defaultwert: 'convexity'
Werteliste: Feature ∈ {'convexity', 'compactness', 'anisometry', 'holes'}

▷ **Row** (input_control) . . . . . point.y ⇝ *integer*
Zeile des Punktes, den die Region enthalten muss.
Defaultwert: 256
Wertevorschläge: Row ∈ {10, 50, 100, 200, 300, 400}

▷ **Column** (input_control) . . . . . point.x ⇝ *integer*
Spalte des Punktes, den die Region enthalten muss.
Defaultwert: 256
Wertevorschläge: Column ∈ {10, 50, 100, 200, 300, 400}

▷ **AbsoluteHisto** (output_control) . . . . . histogram-array ⇝ *real* / integer
Absolute Verteilung des Merkmals.

▷ **RelativeHisto** (output_control) . . . . . histogram-array ⇝ *real*
Relative Verteilung des Merkmals.

Ergebnis

shape_histo_point liefert den Wert TRUE, falls ein Bild mit definierten Grauwerten eingegeben wird. Das Verhalten bei leerer Eingabe (keine Bilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_mbutton, area_center

Nachfolger

histo_to_thresh, threshold, gen_region_histo

Alternativen

shape_histo_all

Siehe auch

connection, connect_and_holes, convexity, compactness

Modul

Foundation

6.8 Typanpassung

complex_to_real (ImageComplex : ImageReal, ImageImaginary : :)

Umwandlung eines komplexwertigen Bildes in zwei real-Bilder.

complex_to_real wandelt das komplexwertige Bild ImageComplex in zwei reellwertige Bilder ImageReal und ImageImaginary um, die den Real- bzw. Imaginärteil des komplexen Bildes enthalten.

Parameter

- ▷ **ImageComplex** (input_object) ..... singlechannelimage(-array) ⇝ *object :* complex
  Komplexes Bild.
- ▷ **ImageReal** (output_object) ..... image(-array) ⇝ *object :* real
  Realteil.
- ▷ **ImageImaginary** (output_object) ..... image(-array) ⇝ *object :* real
  Imaginärteil.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Siehe auch

real_to_complex

Modul

Foundation

convert_image_type (Image : ImageConverted : NewType :)

Umwandeln des Bildtyps.

convert_image_type konvertiert Bilder eines beliebigen Bildtyps in einen beliebigen Bildtyp. Erfolgt die Konvertierung von einem größeren in einen kleineren Grauwertbereich (z.B. von 'int4' nach 'byte') werden zu große oder zu kleine Werte einfach „geclippt". Sind die Ergebnisbilder vom Typ 'direction', werden Grauwerte, die nicht innerhalb des für 'direction'-Bilder erlaubten Wertebereichs liegen, auf den Wert 255 gesetzt, um sie als ungültig zu kennzeichnen. Es empfiehlt sich daher, vor Aufruf der Routine den Wertebereich der Grauwerte im Eingabebild mittels scale_image entsprechend anzupassen. Bei der Konversion von Bildern vom Typ complex wird nur der Realteil konvertiert. Der Imaginärteil wird ignoriert. Dies erlaubt eine effiziente Konversion von

Bildern, die mit `fft_generic` aus dem Frequenzraum zurücktransformiert worden sind. Solche Bilder haben immer einen Imaginärteil von 0.

Achtung

Stimmen der Ziel- und der Quelltyp überein, dann wird keine neue Bildmatrix angelegt. Die Konvertierung in 'int8' Bilder ist nur auf 64-Bit-Systemen möglich!

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex
  Bilder, deren Grauwerttyp geändert werden sollen.
- ▷ **ImageConverted** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex
  Ergebnisbild.
- ▷ **NewType** (input_control) ..... string ⇝ *string*
  Gewünschter Bildtyp (i.e. Typ der Grauwerte).
  Defaultwert: 'byte'
  Werteliste: `NewType` ∈ {'int1', 'int2', 'uint2', 'int4', 'int8', 'byte', 'real', 'direction', 'cyclic', 'complex'}

Ergebnis

`convert_image_type` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`scale_image`

Siehe auch

`scale_image`, `abs_image`

Modul

Foundation

real_to_complex (ImageReal, ImageImaginary : ImageComplex : :)

Umwandlung von zwei real-Bildern in ein komplexes Bild.

`real_to_complex` wandelt zwei reellwertige Bilder `ImageReal` und `ImageImaginary` (Real- bzw. Imaginärteil eines komplexen Bildes) in ein komplexwertiges Bild `ImageComplex` um.

Parameter

- ▷ **ImageReal** (input_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Realteil.
- ▷ **ImageImaginary** (input_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Imaginärteil.
- ▷ **ImageComplex** (output_object) ..... image(-array) ⇝ *object :* complex
  Komplexes Bild

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Siehe auch

complex_to_real

Modul

Foundation

real_to_vector_field (Row, Col : VectorField : Type :)

Umwandlung von zwei reellwertigen Bildern in ein Vektorfeldbild.

real_to_vector_field wandelt zwei reellwertige Bilder Row und Col in ein Vektorfeldbild VectorField um. Die Eingabebilder enthalten dabei die Vektorkomponenten in Zeilen- bzw. Spalten-Richtung. Der semantische Typ von VectorField wird in Type gesetzt.

Parameter

- ▷ **Row** (input_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Vektorkomponente in Zeilenrichtung.
- ▷ **Col** (input_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Vektorkomponente in Spaltenrichtung.
- ▷ **VectorField** (output_object) ..... image(-array) ⇝ *object :* vector_field
  Verschiebungsvektorfeld
- ▷ **Type** (input_control) ..... string ⇝ *string*
  Semantischer Typ
  Defaultwert: 'vector_field_relative'
  Werteliste: Type ∈ {'vector_field_relative', 'vector_field_absolute'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

vector_field_to_real

Modul

Foundation

vector_field_to_real (VectorField : Row, Col : :)

Umwandlung eines Vektorfeldes in zwei reellwertige Bilder.

vector_field_to_real wandelt das Vektorfeldbild (VectorField) in zwei reellwertige Bilder Row und Col um. Die Ausgabe-Bilder enthalten die Vektorkomponenten in Zeilen- bzw. Spalten-Richtung.

Parameter

- ▷ **VectorField** (input_object) ..... singlechannelimage(-array) ⇝ *object :* vector_field
  Vektorfeld.
- ▷ **Row** (output_object) ..... image(-array) ⇝ *object :* real
  Vektorkomponente in Zeilenrichtung.
- ▷ **Col** (output_object) ..... image(-array) ⇝ *object :* real
  Vektorkomponente in Spaltenrichtung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`optical_flow_mg`

Siehe auch

`optical_flow_mg`

Modul

Foundation

6.9 Zugriff

`get_grayval` (`Image` : : `Row`, `Column` : `Grayval`)

Zugriff auf Grauwerte eines Bildobjektes.

`Grayval` ist ein Tupel von Gleitkomma- bzw. ganzen Zahlen, das die Grauwerte ausgewählter Bildpunkte von `Image` liefert. Bei einem mehrkanaligen Bild wird pro Bildpunkt eine Gruppe von Elementen, d.h. ein Grauwert pro Kanal (oder zwei Grauwerte für komplexe oder Vektorfeld-Bilder), zurückgegeben. Die Zeilenkoordinaten der Bildpunkte stehen im Tupel `Row`, die Spaltenkoordinaten in `Column`.

Es ist zu beachten, dass `get_grayval` den Definitionsbereich des Bildes nicht berücksichtigt. D.h. wenn der Definitionsbereich z.B. mit `reduce_domain` eingeschränkt wurde, werden auch für Punkte die außerhalb des Definitionsbereichs liegen Grauwerte zurück gegeben.

Achtung

Der Typ der Werte von `Grayval` hängt von dem Typ der Grauwerte ab. Der Aufwand von `get_grayval` ist relativ hoch. Normalerweise wird der Operator verwendet, um einzelne Grauwerte eines Bildes zu erhalten (z.B. durch `get_mposition` gefolgt von `get_grayval`). Für die Programmierung von Bildverarbeitungsoperationen wie Filter ist er nicht geeignet. Hier ist es sinnvoller, den Operator `get_image_pointer1` und direkt die C- oder C++-Schnittstelle zum Einbinden von eigenen Prozeduren zu verwenden.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Bild, auf dessen Grauwerte zugegriffen werden soll.

▷ **`Row`** (input_control) ..... point.y(-array) ⇝ *integer*
Zeilennummern der zu betrachtenden Bildpunkte.
Defaultwert: 0
Wertevorschläge: `Row` ∈ {0, 64, 128, 256, 512, 1024}
Typischer Wertebereich: 0 ≤ `Row` ≤ 32768 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: 0 <= Row && Row < height(Image)

▷ **`Column`** (input_control) ..... point.x(-array) ⇝ *integer*
Spaltennummern der zu betrachtenden Bildpunkte.
Parameteranzahl: Column == Row
Defaultwert: 0
Wertevorschläge: `Column` ∈ {0, 64, 128, 256, 512, 1024}
Typischer Wertebereich: 0 ≤ `Column` ≤ 32768 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: 0 <= Column && Column < width(Image)

▷ **Grayval** (output_control) . . . . . grayval-array ⇝ *real* / integer
Grauwerte an den angegebenen Bildpunkten.

Ergebnis

Bei korrekter Besetzung der Parameter liefert get_grayval den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_image

Alternativen

get_image_pointer1, get_grayval_interpolated, get_grayval_contour_xld

Siehe auch

set_grayval

Modul

Foundation

get_grayval_contour_xld (Image,
Contour : : Interpolation : Grayval)

Gibt Grauwerte eines Bildes an den Positionen einer XLD-Kontur zurück.

Der Operator get_grayval_contour_xld liefert die interpolierten Grauwerte einzelner Subpixel-Positionen des Bildes Image. Die Koordinaten der Positionen werden durch eine XLD-Kontur Contour vorgegeben. Die Grauwerte werden in Grayval zurückgegeben.

Das Interpolationsverfahren kann mittels des Parameters Interpolation gewählt werden:

'nearest_neighbor': Die Ergebnisse in Grayval sind die Grauwerte der nächstgelegenen Pixel der gegebenen Koordinaten. Für Bilder des Typs byte, direction, cyclic, uint2, int1, int2, int4 und int8 ist der Parameter Grayval ein Tupel von ganzen Zahlen. Für Bilder des Typs real und complex ist der Parameter Grayval ein Tupel von Gleitkommazahlen.

'bilinear': Die Ergebnisse in Grayval werden mittels einer bilinearen Interpolation berechnet, die die vier benachbarten Grauwerte der gegebenen Koordinaten nutzt. Der Parameter Grayval mit dem Ergebnis ist ein Tupel von Gleitkommazahlen. Die Laufzeit steigt signifikant gegenüber *'nearest_neighbor'*. Direction- und Cyclic-Bilder werden behandelt wie Byte-Bilder.

'bicubic': Die Ergebnisse in Grayval werden mittels einer bikubischen Interpolation berechnet, die sechzehn benachbarte Grauwerte der gegebenen Koordinaten nutzt. Der Parameter Grayval mit dem Ergebnis ist ein Tupel von Gleitkommazahlen. Die Laufzeit steigt signifikant gegenüber *'bilinear'*. Direction- und Cyclic-Bilder werden behandelt wie Byte-Bilder. In diesem Modus können Grauwerte als Ergebnis auftreten, die außerhalb des darstellbaren Zahlenbereiches des Eingabebildtyps liegen.

'bicubic_clipped': Die Ergebnisse in Grayval werden mittels einer bikubischen Interpolation berechnet, die sechzehn benachbarte Grauwerte der gegebenen Koordinaten nutzt. Der Parameter Grayval mit dem Ergebnis ist ein Tupel von Gleitkommazahlen. Die Laufzeit steigt signifikant gegenüber *'bilinear'*. Direction- und Cyclic-Bilder werden behandelt wie Byte-Bilder. In diesem Modus werden Ergebnisse, die außerhalb des darstellbaren Zahlenbereiches des Eingabebildtyps liegen, auf den darstellbaren Bereich beschnitten.

Es ist zu beachten, dass get_grayval_contour_xld den Definitionsbereich des Bildes nicht berücksichtigt. D.h. wenn der Definitionsbereich z.B. mit reduce_domain eingeschränkt wurde, werden auch für Punkte die außerhalb des Definitionsbereichs liegen Grauwerte zurück gegeben.

Es ist außerdem zu beachten, dass jeder Punkt der XLD-Kontur im Bereich $-0.5 \leq$ Zeilenkoordinate $<$ Höhe(Image)-0.5 und $-0.5 \leq$ Spaltenkoordinate $<$ Breite(Image)-0.5 liegen muss.

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex
Bild, auf dessen Grauwerte zugegriffen werden soll.

▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
Eingabe-Kontur mit den Koordinaten der Positionen.

▷ **Interpolation** (input_control) ..... string ⇝ *string*
Interpolationsverfahren.
Defaultwert: 'nearest_neighbor'
Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'bicubic_clipped'}

▷ **Grayval** (output_control) ..... grayval(-array) ⇝ *real* / integer
Grauwerte an den angegebenen Bildkoordinaten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_grayval_contour_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`, `draw_xld`, `gen_contour_polygon_xld`, `edges_sub_pix`

Alternativen

`get_grayval_interpolated`

Modul

Foundation

get_grayval_interpolated (Image : : Row, Column,
Interpolation : Grayval)

Gibt Grauwerte eines Bildes an durch Zeilen- und Spaltenkoordinaten gegebenen Positionen zurück.

Der Operator `get_grayval_interpolated` liefert die interpolierten Grauwerte einzelner Subpixel-Positionen des Bildes `Image`. Die Koordinaten der Zeilenpositionen werden in dem Tupel `Row`, die Koordinaten der Spaltenpositionen in dem Tupel `Column` übergeben. Der Grauwerte werden in `Grayval` als ein Tupel von Gleitkommazahlen zurückgegeben.

Das Interpolationsverfahren kann mittels des Parameters `Interpolation` gewählt werden. Folgende Interpolationsmethoden werden unterstützt:

'bilinear': Die Ergebnisse in `Grayval` werden mittels einer bilinearen Interpolation berechnet, die die vier benachbarten Grauwerte der gegebenen Koordinaten nutzt. Direction- und Cyclic-Bilder werden behandelt wie Byte-Bilder.

'bicubic': Die Ergebnisse in `Grayval` werden mittels einer bikubischen Interpolation berechnet, die sechzehn benachbarte Grauwerte der gegebenen Koordinaten nutzt. Direction- und Cyclic-Bilder werden behandelt wie Byte-Bilder. In diesem Modus können Grauwerte als Ergebnis auftreten, die außerhalb des darstellbaren Zahlenbereiches des Eingabebildtyps liegen.

'bicubic_clipped': Die Ergebnisse in `Grayval` werden mittels einer bikubischen Interpolation berechnet, die sechzehn benachbarte Grauwerte der gegebenen Koordinaten nutzt. Direction- und Cyclic-Bilder werden behandelt wie Byte-Bilder. In diesem Modus werden Ergebnisse, die außerhalb des darstellbaren Zahlenbereiches des Eingabebildtyps liegen, auf den darstellbaren Bereich beschnitten.

Es ist zu beachten, dass `get_grayval_interpolated` den Definitionsbereich des Bildes nicht berücksichtigt. D.h. wenn der Definitionsbereich z.B. mit `reduce_domain` eingeschränkt wurde, werden auch für Punkte die außerhalb des Definitionsbereichs liegen Grauwerte zurück gegeben.

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex
Bild, auf dessen Grauwerte zugegriffen werden soll.

▷ **Row** (input_control) ..... point.y(-array) ⇝ *real* / integer
Zeilenkoordinaten der Positionen.
Defaultwert: 0
Wertevorschläge: `Row` ∈ {0, 64, 128, 256, 512, 1024}
Typischer Wertebereich: -0.5 ≤ Row
Restriktion: -0.5 <= Row && Row < height(Image) - 0.5

▷ **Column** (input_control) ..... point.x(-array) ⇝ *real* / integer
Spaltenkoordinaten der Positionen.
Parameteranzahl: Column == Row
Defaultwert: 0
Wertevorschläge: `Column` ∈ {0, 64, 128, 256, 512, 1024}
Typischer Wertebereich: -0.5 ≤ Column
Restriktion: -0.5 <= Column && Column < width(Image) - 0.5

▷ **Interpolation** (input_control) ..... string ⇝ *string*
Interpolationsverfahren.
Defaultwert: 'bilinear'
Werteliste: `Interpolation` ∈ {'bilinear', 'bicubic', 'bicubic_clipped'}

▷ **Grayval** (output_control) ..... grayval(-array) ⇝ *real*
Grauwerte an den angegebenen Bildkoordinaten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_grayval_interpolated` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`

Alternativen

`get_grayval_contour_xld`, `get_grayval`

Modul

Foundation

get_image_pointer1 (Image : : : Pointer, Type, Width, Height)

Zugriff auf den Zeiger eines Kanals.

`get_image_pointer1` liefert einen Pointer auf den ersten Kanal des Bildes `Image`. Außerdem wird der Bildtyp (`Type` = 'byte', 'int2', 'uint2' etc.) und die Bildgröße (Breite und Höhe) ausgegeben. In der Folge ist dann über den Zeiger ein direkter Zugriff auf die Bilddaten in der HALCON-Datenbank von der Programmiersprache aus möglich, in der HALCON verwendet wird. Ein Bild wird in HALCON zeilenweise linearisiert abgespeichert. Die Bildtypen 'complex' und 'vector_field' bilden Sonderfälle. Bilder vom Typ 'complex' sind interleaved, d.h., die Real- und Imaginärteile werden abwechselnd abgespeichert. Bilder vom Typ 'vector_field' bestehen dagegen aus zwei Matrizen, eine für die Zeilen und eine für die Spalten, die nacheinander in der HALCON-Datenbank liegen.

Achtung

Der von `get_image_pointer1` zurückgegebene Zeiger darf nur so lange verwendet werden, solange das Bildobjekt existiert. Das ist der Fall, solange die zugehörige Variable in der Programmiersprache, in der HALCON

verwendet wird, gültig ist. Wird dies nicht berücksichtigt, kann es zu unerwartetem Verhalten und zu Programmabstürzen kommen.

Werden über den Zeiger Daten in ein existierendes Bild geschrieben, so werden alle Bildobjekte modifiziert, die das Bild referenzieren. Wenn z.B. mit `reduce_domain` der Definitionsbereich eines Bildes eingeschränkt wird, so benutzen das originale Bildobjekt mit dem vollen Definitionsbereich und das Bildobjekt mit dem eingeschränkten Definitionsbereich dieselbe Bildmatrix (d.h. `get_image_pointer1` liefert für die beiden Bilder denselben Zeiger zurück). Falls also eines der beiden Bilder in diesem Beispiel überschrieben wird, werden beide Bildobjekte modifiziert. Falls der Zeiger in der Programmiersprache, in der HALCON verwendet wird, dazu benutzt wird, Bilddaten zu schreiben, sollten daher die Bilddaten in ein Bildobjekt geschrieben werden, das neu erzeugt worden ist, z.B. mit `gen_image1`. Bei mehrkanaligen Eingabebildern wird der Bildtyp und der Pointer des ersten Kanals zurückgegeben.

Parameter

▷ **`Image`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Eingabebild.

▷ **`Pointer`** (output_control) ..... pointer(-array) ⇝ *integer*
Zeiger auf die Bilddaten in der HALCON-Datenbank.

▷ **`Type`** (output_control) ..... string(-array) ⇝ *string*
Typ des Bildes.
Werteliste: `Type` ∈ {'int1', 'int2', 'uint2', 'int4', 'int8', 'byte', 'real', 'direction', 'cyclic', 'complex', 'vector_field_absolute', 'vector_field_relative'}

▷ **`Width`** (output_control) ..... extent.x(-array) ⇝ *integer*
Breite des Bildes.

▷ **`Height`** (output_control) ..... extent.y(-array) ⇝ *integer*
Höhe des Bildes.

Beispiel

```
Hobject  Image;
char     typ[128];
Hlong     width,height;
unsigned char *ptr;

read_image(&Image,"fabrik");
get_image_pointer1(Image,(Hlong*)&ptr,typ,&width,&height);
```

Ergebnis

`get_image_pointer1` liefert den Wert TRUE, falls genau ein Bild übergeben wurde. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`

Alternativen

`set_grayval`, `get_grayval`, `get_image_pointer3`, `get_image_size`, `get_image_type`

Siehe auch

`paint_region`, `paint_gray`

Modul

Foundation

get_image_pointer1_rect (Image : : : PixelPointer, Width, Height, VerticalPitch, HorizontalBitPitch, BitsPerPixel)

Zugriff auf Bilddatenzeiger und Bildpunkte innerhalb des kleinsten umschließenden Rechtecks der Domäne des Eingabebildes.

Der Operator get_image_pointer1_rect liefert den Zeiger PixelPointer auf den Anfang der Bilddaten, die sich innerhalb des kleinsten umschließenden Rechtecks der Domäne von Image befinden. VerticalPitch entspricht der Breite des Eingabebildes Image mal der Anzahl der Bytes pro Pixel (HorizontalBitPitch / 8). Width und Height entsprechen der Größe des kleinsten umschließenden Rechtecks der Eingaberegion. HorizontalBitPitch ist der horizontale Abstand (in Bits) benachbarter Pixel. BitsPerPixel ist die Anzahl der verwendeten Bits pro Pixel. get_image_pointer1_rect ist symmetrisch zu gen_image1_rect.

Achtung

get_image_pointer1_rect sollte nur zum Eintragen in neu erzeugte Bilder verwendet werden, da sonst die Grauwerte von anderen Bildern überschrieben werden könnten (siehe relationale Struktur).

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / int4
  Eingabebild (Himage).
- ▷ **PixelPointer** (output_control) ..... pointer ⇝ *integer*
  Zeiger auf die Bilddaten.
- ▷ **Width** (output_control) ..... extent.x ⇝ *integer*
  Breite des Ausgabebildes.
- ▷ **Height** (output_control) ..... extent.y ⇝ *integer*
  Höhe des Ausgabebildes.
- ▷ **VerticalPitch** (output_control) ..... integer ⇝ *integer*
  Breite(Eingabebild)*(HorizontalBitPitch/8).
- ▷ **HorizontalBitPitch** (output_control) ..... integer ⇝ *integer*
  Abstand benachbarter Pixel in Bits.
- ▷ **BitsPerPixel** (output_control) ..... integer ⇝ *integer*
  Anzahl verwendeter Bits pro Pixel.

Beispiel

```
Hobject       image,reg,imagereduced;
char          typ[128];
Hlong          width,height,vert_pitch,hori_bit_pitch,bits_per_pix, winID;
unsigned char *ptr;

open_window(0,0,512,512,"black",winID);
read_image(&image,"monkey");
draw_region(&reg,winID);
reduce_domain(image,reg,&imagereduced);
get_image_pointer1_rect(imagereduced,(Hlong*)&ptr,&width,&height,
                        &vert_pitch,&hori_bit_pitch,&bits_per_pix);
```

Ergebnis

get_image_pointer1_rect liefert den Wert TRUE, falls genau ein Bild übergeben wurde. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_image, gen_image1_rect

Alternativen

set_grayval, get_grayval, get_image_pointer3, get_image_pointer1

Siehe auch

paint_region, paint_gray, gen_image1_rect

Modul

Foundation

get_image_pointer3 (ImageRGB : : : PointerRed, PointerGreen, PointerBlue, Type, Width, Height)

Zugriff auf die Zeiger eines Farbbildes.

get_image_pointer3 liefert einen C-Pointer auf die drei Kanäle eines Farbbildes (ImageRGB). Außerdem wird der Bildtyp (Type = 'byte', 'int2', 'float' etc.) und die Bildgröße (Width und Height) ausgegeben. In der Folge ist dann über den Zeiger ein direkter Zugriff auf die Bilddaten in der HALCON-Datenbank von der HALCON Wirtssprache aus möglich. Ein Bild ist in HALCON als ein Vektor von Bildzeilen abgelegt. Die Bildtypen 'complex' und 'vector_field' bilden Sonderfälle. Bilder vom Typ 'complex' sind interleaved, d.h., die Real- und Imaginärteile werden abwechselnd abgespeichert. Bilder vom Typ 'vector_field' bestehen dagegen aus zwei Matrizen, eine für die Zeilen und eine für die Spalten, die nacheinander in der HALCON-Datenbank liegen. Die drei Kanäle müssen den gleichen Pixeltyp und die gleiche Größe haben.

Achtung

Es darf nur ein Bild übergeben werden. get_image_pointer3 sollte nur zum Eintragen in neu erzeugte Bilder verwendet werden, da sonst die Grauwerte von anderen Bildern überschrieben werden könnten (siehe Relationale Struktur).

Parameter

- ▷ **ImageRGB** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
  Eingabebild.
- ▷ **PointerRed** (output_control) ..... pointer(-array) ⇝ *integer*
  Zeiger auf die Pixel des ersten Kanals.
- ▷ **PointerGreen** (output_control) ..... pointer(-array) ⇝ *integer*
  Zeiger auf die Pixel des zweiten Kanals.
- ▷ **PointerBlue** (output_control) ..... pointer(-array) ⇝ *integer*
  Zeiger auf die Pixel des dritten Kanals.
- ▷ **Type** (output_control) ..... string(-array) ⇝ *string*
  Typ des Bildes.
  Werteliste: Type ∈ {'int1', 'int2', 'uint2', 'int4', 'int8', 'byte', 'real', 'direction', 'cyclic', 'complex', 'vector_field_absolute', 'vector_field_relative'}
- ▷ **Width** (output_control) ..... extent.x(-array) ⇝ *integer*
  Breite des Bildes.
- ▷ **Height** (output_control) ..... extent.y(-array) ⇝ *integer*
  Höhe des Bildes.

Ergebnis

get_image_pointer3 liefert den Wert TRUE, falls genau ein Bild übergeben wird. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`

Alternativen

`set_grayval`, `get_grayval`, `get_image_pointer1`

Siehe auch

`paint_region`, `paint_gray`

Modul

Foundation

get_image_size (Image : : : Width, Height)

Liefert die Bildgröße.

`get_image_size` liefert die Größe (Breite und Höhe) des Eingabebildes zurück.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Eingabebild.

▷ **Width** (output_control) ..... extent.x(-array) ⇝ *integer*
Breite des Bildes.

▷ **Height** (output_control) ..... extent.y(-array) ⇝ *integer*
Höhe des Bildes.

Ergebnis

`get_image_size` liefert den Wert TRUE zurück. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`

Alternativen

`get_image_pointer1`

Siehe auch

`get_image_type`

Modul

Foundation

get_image_time (Image : : : MSecond, Second, Minute, Hour, Day, YDay, Month, Year)

Abfragen der Zeit, zu der das Bild erzeugt wurde.

`get_image_time` liefert den Zeitpunkt, zu dem das Bild intern in HALCON erzeugt wurde. Diese Zeit hängt nicht unbedingt mit der Zeit zusammen, zu der das Bild aufgenommen wurde.

Für manche Bildeinzugsschnittstellen kann die Zeit der Aufnahme mit `get_framegrabber_param` abgefragt werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
  Eingabebild.
- ▷ **MSecond** (output_control) ..... integer ⇝ *integer*
  Millisekunden (0..999).
- ▷ **Second** (output_control) ..... integer ⇝ *integer*
  Sekunden (0..59).
- ▷ **Minute** (output_control) ..... integer ⇝ *integer*
  Minuten (0..59).
- ▷ **Hour** (output_control) ..... integer ⇝ *integer*
  Stunden (0..23).
- ▷ **Day** (output_control) ..... integer ⇝ *integer*
  Tag innerhalb des Monats (1..31).
- ▷ **YDay** (output_control) ..... integer ⇝ *integer*
  Tag innerhalb des Jahres (1..366).
- ▷ **Month** (output_control) ..... integer ⇝ *integer*
  Monat (1..12).
- ▷ **Year** (output_control) ..... integer ⇝ *integer*
  Jahr (xxxx).

Ergebnis

`get_image_time` liefert den Wert TRUE, falls genau ein Bild übergeben wurde. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`, `grab_image`

Siehe auch

`count_seconds`

Modul

Foundation

`get_image_type` (Image : : : Type)

Liefert den Bildtyp.

`get_image_type` liefert den Bildtyp (`Type` = 'byte', 'int2', 'uint2' etc.) zurück.

Achtung

Bei mehrkanaligen Eingabebildern wird der Bildtyp des ersten Kanals zurückgegeben.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
  Eingabebild.

▷ **Type** (output_control) . . . . . string(-array) ⇝ *string*
Typ des Bildes.
Werteliste: `Type` ∈ {'int1', 'int2', 'uint2', 'int4', 'int8', 'byte', 'real', 'direction', 'cyclic', 'complex', 'vector_field_absolute', 'vector_field_relative'}

Ergebnis

`get_image_type` liefert den Wert TRUE zurück. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`

Alternativen

`get_image_pointer1`

Siehe auch

`get_image_size`

Modul

Foundation

Kapitel 7

Datei

7.1 Bilddaten

deserialize_image (: Image : SerializedItemHandle :)

Deserialisieren eines serialisierten Bildobjekts.

deserialize_image deserialisiert ein Bildobjekt, welches mit serialize_image serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Bildobjekt wird in dem Parameter SerializedItemHandle übergeben. Das deserialisierte Bildobjekt wird in dem durch den Parameter Image definierten Objekt gespeichert.

Parameter

▷ **Image** (output_object) ..... image(-array) ⇝ *object :* byte / direction / cyclic / int1 / complex / int2 / uint2 / vector_field / int4 / int8 / real
Bildobjekt.

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_image den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_image

Nachfolger

disp_image, threshold, regiongrowing, count_channels, decompose3, class_ndim_norm, gauss_filter, fill_interlace, zoom_image_size, zoom_image_factor, crop_part, write_image, rgb1_to_gray

Siehe auch

serialize_image

Modul

Foundation

read_image (: Image : FileName :)

Einlesen eines Bildes mit unterschiedlichen Dateiformaten.

Der Operator read_image liest die angegebenen Bilddateien vom Hintergrundspeicher und erzeugt daraus ein Bild. Es können ein oder mehrere Dateinamen in FileName angegeben werden. In letzterem Fall wird ein Bildobjekttupel mit entsprechend vielen Elementen zurückgegeben.

HALCON-Formate (HOBJ und IMA)

Bei Dateien im HALCON-Iconic-Object-Format (HOBJ) werden Multi-Bild-Dateien als Bildobjekttupel zurückgegeben. Falls die HOBJ-Datei zusätzlich noch andere Objekt-Typen (Regionen oder XLDs) enthält, wird ein Fehler erzeugt. Die Beschreibung des Operators write_object enthält weitere Informationen über das HOBJ-Format.

Alle Bilddateien, die mit write_image (Format 'ima') geschrieben werden, haben die Extension *'.ima'*. Zu jedem Bild im IMA-Format kann eine Beschreibungsdatei vorhanden sein (gleicher Dateiname mit Extension *'.exp'*). Der Typ der Pixeldaten (*'byte'*, *'int4'*, *'real'*) wird ebenfalls der Beschreibungsdatei entnommen. Ist diese Beschreibungsdatei nicht vorhanden, so wird der Typ *'byte'* als Voreinstellung verwendet, genau wie die Standardwerte für die Datengröße (*'width'* x *'height'*) die mit get_system abgefragt werden können. Wenn die Datengröße der *'.ima'*-Datei nicht mit der Standardgröße übereinstimmt, wird eine Fehlerbehandlung durchgeführt.

Andere Formate

Neben dem HALCON-Format können auch TIFF, GIF, BMP, JPEG, JPEG-XR JPEG-2000, PNG, PCX, SUN-Raster, PGM, PPM, PBM und XWD Dateien eingelesen werden. Die Grauwerte bei PBM-Bildern werden auf Werte 0 und 255 gesetzt. Die Dateiformate werden entweder an der Extension (soweit angegeben) oder aufgrund der inneren Struktur der Dateien erkannt. Bei Angabe der Extension wird das Bild schneller gefunden. Wird keine Extension angegeben, werden Dateien mit Extension (sofern vorhanden) Dateien ohne Extension vorgezogen. Bei PGM, PPM und PBM kann die jeweilige Extension (z.B. *'pgm'*) oder auch allgemein *'pnm'* verwendet werden. Bei TIFF ist *'tiff'* und *'tif'* zulässig. Bei JPEG-XR Dateien sind die Endungen *'jxr'*, *'wdp'*, *'wmp'* und *'hdp'* erlaubt. Bei JPEG-2000 Dateien ist nur die Endung *'jp2'* zulässig. Bei Farbbildern wird ein Bild mit drei Farbkanälen erzeugt, wobei der Rotkanal in der ersten, der Grünkanal in der zweiten und der Blaukanal in der dritten Komponente (Kanalnummer) abgelegt wird.

Im Falle der Bildformate TIFF, PNG und JPEG-2000 werden binäre Alphakanäle als Domänen interpretiert. Bei TIFF-Dateien werden zusätzlich binäre SubIFDs mit PhotometricInterpretation = TransparencyMask als Domänen interpretiert. Ansonsten wird die Domäne des erzeugten Bildobjektes maximal (= alle Bildpunkte der Matrix) gewählt.

Bei TIFF-Dateien werden Multi-Page-Dateien als Bildobjekttupel zurückgegeben. SubIFDs in einer TIFF-Datei werden nur dann eingelesen, wenn ihre Dimensionen den Dimensionen des Hauptbildes entsprechen.

Bilder des Typs 'int8' können nur auf 64-Bit-Systemen gelesen werden. Außerdem unterstützen nur die Formate IMA und TIFF diesen Bildtyp.

Bilddateien werden im aktuellen Verzeichnis und im Bilderverzeichnis von HALCON (vorgegeben durch die Umgebungsvariablen) gesucht. Das Bilderverzeichnis von HALCON ist in einer unixartigen Umgebung auf `'.'` und `'/usr/local/halcon/images'` voreingestellt und kann mit dem Befehl set_system gesetzt werden. Es kann mehr als ein Bilderverzeichnis angegeben werden. Dies geschieht, indem man die einzelnen Verzeichnisse durch einen Doppelpunkt trennt.

Weiterhin kann der Suchpfad mit der Environment-Variablen HALCONIMAGES gesetzt werden (gleicher Aufbau wie bei *'image_dir'*). Beispiel:

```
setenv HALCONIMAGES "/usr/images:/usr/local/halcon/images"
```

Ebenso sucht HALCON die Bilder in dem Unterverzeichnis *'images'* (Bilder für die Programmbeispiele). Für das HALCON-Verzeichnis wird die Environment-Variable HALCONROOT verwendet.

Achtung

Falls CMYK oder YCCK JPEG/JPEG-XR-Dateien gelesen werden, nimmt HALCON an, dass diese Dateien der Konvention von Adobe Photoshop folgen, gemäß der die CMYK-Kanäle invertiert gespeichert sind, d.h. 0 repräsentiert 100% Druckfarbe und nicht 0%, wie man es eigentlich erwarten würde. Die Bilder werden gemäß dieser Konvention in RGB-Bilder umgewandelt. Falls die JPEG-Dateien nicht dieser Konvention folgen und die CMYK-Kanäle in der üblichen Weise abgespeichert sind, muss invert_image nach dem Einlesen des Bildes aufgerufen werden.

Beim Lesen von PNG-Dateien mit Alphakanal wird der Alphakanal als zweiter bzw. vierter Kanal des Ausgabebildes zurückgegeben, außer der Alpha-Kanal enthält genau zwei verschiedene Grauwerte. In diesem Fall wird

ein ein- oder dreikanaliges Bild mit eingeschränktem Definitionsbereich zurückgegeben, in dem die Punkte des Definitionsbereiches den Pixeln des Alpha-Kanals mit dem höheren der zwei Grauwerte entspricht.

Parameter

▷ **Image** (output_object) ..... image(-array) ⇝ *object :* byte / direction / cyclic / int1 / complex / int2 / uint2 / vector_field / int4 / int8 / real

Eingelesenes Bild.

▷ **FileName** (input_control) ..... filename.read(-array) ⇝ *string*

Name des einzulesenden Bildes.
Defaultwert: 'printer_chip/printer_chip_01'
Wertevorschläge: `FileName` ∈ {'fabrik', 'fuse', 'mreut', 'multiple_dies_01', 'particle', 'patras', 'printer_chip/printer_chip_01', 'rings_and_nuts', 'tooth_rim'}
Dateiendung: `.hobj`, `.ima`, `.tif`, `.tiff`, `.gif`, `.bmp`, `.jpg`, `.jpeg`, `.jp2`, `.jxr`, `.png`, `.pcx`, `.ras`, `.xwd`, `.pbm`, `.pnm`, `.pgm`, `.ppm`

Beispiel

```
* Reading an image:
  read_image(Image,'mreut')

* Reading 3 images into an image array:
  read_image(Images,['ic0','ic1','ic2'])

* Setting of search path for images on '/mnt/images' and '/home/images':
  set_system('image_dir','/mnt/images:/home/images')
```

Ergebnis

Sind die Parameter korrekt, dann liefert `read_image` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`disp_image`, `threshold`, `regiongrowing`, `count_channels`, `decompose3`, `class_ndim_norm`, `gauss_filter`, `fill_interlace`, `zoom_image_size`, `zoom_image_factor`, `crop_part`, `write_image`, `rgb1_to_gray`

Alternativen

`read_sequence`

Siehe auch

`set_system`, `write_image`

Modul

Foundation

read_image_metadata (: : Format, TagName, FileName : TagValue)

Einlesen von Metadaten einer Bilddatei.

Der Operator `read_image_metadata` liest die Metadaten der Datei `FileName` und gibt die Information unter `TagValue` zurück. Dabei spezifiziert `TagName` einen einzelnen Tag oder ein Tupel von Tags, für welche die jeweilige Information gelesen werden soll. Der Parameter `Format` gibt an, in welchem Format die Metadaten vorliegen.

Der Operator unterstützt für `Format` die `TIFF` Metadatenformate `tiff` und `bigtiff`. Dabei wird eine Dateiendung `.tif` vorausgesetzt.

Die folgenden Tags sind lesbar:

TagName	**Datentyp von** TagValue
'tiff_aperture_value'	Gleitkommazahlen
'tiff_copyright'	String
'tiff_date_time'	String
'tiff_exposure_time'	Gleitkommazahlen
'tiff_image_description'	String
'tiff_light_source'	Ganzzahl
'tiff_make'	String
'tiff_software'	String

Es gilt zu beachten, dass der Operator einen Fehler wirft, wenn für den angegebenen Tag TagName kein Wert gesetzt ist.

Alle lesbaren Tags können mit dem String *'tiff_tags_supported_for_reading'* für TagName abgerufen werden, alle schreibbaren Tags mit dem String *'tiff_tags_supported_for_writing'*. Beide Strings dürfen nur als Einzelwerte für TagName verwendet werden.

Parameter

▷ **Format** (input_control) . . . . . string ⇝ *string*
Grafikformat.
Defaultwert: 'tiff'
Wertevorschläge: Format ∈ {'tiff', 'bigtiff'}

▷ **TagName** (input_control) . . . . . tuple ⇝ *string*
Name des einzulesenden Metadaten-Eintrags.
Defaultwert: 'tiff_image_description'
Wertevorschläge: TagName ∈ {'tiff_image_description', 'tiff_make', 'tiff_software', 'tiff_date_time', 'tiff_copyright', 'tiff_exposure_time', 'tiff_aperture_value', 'tiff_light_source', 'tiff_tags_supported_for_reading', 'tiff_tags_supported_for_writing'}

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Bilddatei.
Dateiendung: .tif, .tiff

▷ **TagValue** (output_control) . . . . . tuple ⇝ *string* / integer / real
Wert des eingelesenen Metadaten-Eintrags.

Ergebnis

Kann TagValue für die angegebenen Parameter erfolgreich gelesen werden, dann liefert der Operator read_image_metadata den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation, wie z.B. der verursachende Tag, gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

write_image_metadata

Nachfolger

read_image

Modul

Foundation

```
read_sequence ( : Image : HeaderSize, SourceWidth, SourceHeight,
    StartRow, StartColumn, DestWidth, DestHeight, PixelType,
    BitOrder, ByteOrder, Pad, Index, FileName : )
```

Einlesen von Bildern.

`read_sequence` liest Bilddaten, die unformatiert vorliegen, aus einer Datei und liefert ein „passendes“ Bild zurück. Die Bilddaten müssen dabei Pixel für Pixel zeilenweise hintereinander abgespeichert sein.

Ein etwaiger Datei-Header (mit Länge `HeaderSize` Bytes) wird übersprungen. Die Parameter `SourceWidth` und `SourceHeight` geben die Größe des Bildes auf Datei an. `DestWidth` und `DestHeight` geben an, wie groß das eingelesene Bild sein soll. Im einfachsten Fall stimmen diese Parameter überein. Es können aber auch Ausschnitte eingelesen werden. Dazu kann mittels `StartRow` und `StartColumn` die obere linke Ecke des gewünschten Bildausschnitts festgelegt werden.

An Pixeltypen werden *'bit'*, *'byte'*, *'short'* (16 Bit, ohne Vorzeichen), *'signed_short'* (16 Bit, mit Vorzeichen), *'long'* (32 Bit, mit Vorzeichen), *'swapped_long'* (32 Bit, mit vertauschten Halbwörtern) und *'real'* (32 Bit Gleitkommazahlen) (unterstützt. Weiter erlaubt `read_sequence` die Extraktion von Komponenten eines RGB Bildes, sofern für jedes Pixel ein Tripel von drei Bytes (in der Reihenfolge „rot“, „grün“, „blau“) in der Bilddatei abgelegt wurde. Für die Rotkomponente ist dann als Pixeltyp *'r_byte'* und entsprechend für die Grün- und Blaukomponente *'g_byte'* bzw. *'b_byte'* zu wählen.

Für die Bitreihenfolge (`BitOrder`) kann *'MSBFirst'* (most significant bit first) bzw. die Umkehrung davon (*'LSBFirst'*) gewählt werden. Analog dazu werden auch die Bytereihenfolgen (`ByteOrder`) *'MSBFirst'* (most significant byte first) bzw. *'LSBFirst'* verarbeitet. Schließlich lässt sich noch ein Alignement (`Pad`) am Zeilenende einstellen: *'byte'*, *'short'* oder *'long'*. Falls in der Datei eine ganze Bildsequenz abgelegt ist, erlaubt der Parameter `Index` die Wahl eines Einzelbildes (beginnend mit Index 1).

Bilddateien werden im aktuellen Directory (vorgegeben durch die Betriebssystemumgebung) und im Bilderdirectory von HALCON gesucht. Das Bilderdirectory von HALCON ist in einer unixartigen Umgebung auf `'.'` und `'/usr/local/halcon/images'` voreingestellt und kann mit dem Befehl `set_system` gesetzt werden. Es kann mehr als ein Bilderdirectory angegeben werden. Dies geschieht, indem man die einzelnen Directories durch einen Doppelpunkt trennt.

Weiterhin kann der Suchpfad mit der Environment-Variablen HALCONIMAGES gesetzt werden (gleicher Aufbau wie bei *'image_dir'*). Beispiel:

```
setenv HALCONIMAGES "/usr/images:/usr/local/halcon/images"\\*
```

Ebenso sucht HALCON die Bilder in dem Unter-Directory *'images'* (Bilder für die Programmbeispiele). Für das HALCON-Directory wird die Environment-Variable HALCONROOT verwendet.

Achtung

Falls Dateien vom Pixeltyp *'real'* eingelesen werden und die Bytereihenfolge falsch (d.h. nicht wie die Daten in der Datei gespeichert sind) gewählt wird, kann es zu Programmfehlern oder sogar Abstürzen aufgrund von Gleitkomma-Ausnahmebehandlungen kommen.

Parameter

- ▷ **`Image`** (output_object) ..... image ⇝ *object :* byte / int2 / uint2 / int4
  Eingelesenes Bild.
- ▷ **`HeaderSize`** (input_control) ..... integer ⇝ *integer*
  Anzahl Bytes für Dateikopf.
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ HeaderSize
- ▷ **`SourceWidth`** (input_control) ..... extent.x ⇝ *integer*
  Anzahl Bildspalten des Bildes auf Datei.
  Defaultwert: 512
  Typischer Wertebereich: 1 ≤ SourceWidth
- ▷ **`SourceHeight`** (input_control) ..... extent.y ⇝ *integer*
  Anzahl Bildzeilen des Bildes auf Datei.
  Defaultwert: 512
  Typischer Wertebereich: 1 ≤ SourceHeight
- ▷ **`StartRow`** (input_control) ..... point.y ⇝ *integer*
  Startpunkt Bildausschnitt (Zeile).
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ StartRow
  Restriktion: StartRow < SourceHeight

▷ **StartColumn** (input_control) . . . . . point.x ⇝ *integer*
Startpunkt Bildausschnitt (Spalte).
Defaultwert: 0
Typischer Wertebereich: 0 ≤ StartColumn
Restriktion: StartColumn < SourceWidth

▷ **DestWidth** (input_control) . . . . . extent.x ⇝ *integer*
Anzahl Bildspalten des erzeugten Bildes.
Defaultwert: 512
Typischer Wertebereich: 1 ≤ DestWidth
Restriktion: DestWidth <= SourceWidth

▷ **DestHeight** (input_control) . . . . . extent.y ⇝ *integer*
Anzahl Bildzeilen des erzeugten Bildes.
Defaultwert: 512
Typischer Wertebereich: 1 ≤ DestHeight
Restriktion: DestHeight <= SourceHeight

▷ **PixelType** (input_control) . . . . . string ⇝ *string*
Art der Pixelwerte.
Defaultwert: 'byte'
Werteliste: `PixelType` ∈ {'bit', 'byte', 'r_byte', 'g_byte', 'b_byte', 'short', 'signed_short', 'long', 'swapped_long', 'real'}

▷ **BitOrder** (input_control) . . . . . string ⇝ *string*
Reihenfolge der Bits innerhalb eines Bytes.
Defaultwert: 'MSBFirst'
Werteliste: `BitOrder` ∈ {'MSBFirst', 'LSBFirst'}

▷ **ByteOrder** (input_control) . . . . . string ⇝ *string*
Reihenfolge der Bytes innerhalb einer 'short'-Einheit.
Defaultwert: 'MSBFirst'
Werteliste: `ByteOrder` ∈ {'MSBFirst', 'LSBFirst'}

▷ **Pad** (input_control) . . . . . string ⇝ *string*
Dateneinheiten innerhalb einer Bildzeile (Alignement).
Defaultwert: 'byte'
Werteliste: `Pad` ∈ {'byte', 'short', 'long'}

▷ **Index** (input_control) . . . . . integer ⇝ *integer*
Nummer des Bildes innerhalb der Datei.
Defaultwert: 1
Typischer Wertebereich: 1 ≤ Index (lin)

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der einzulesenden Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_sequence` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`disp_image`, `count_channels`, `decompose3`, `write_image`, `rgb1_to_gray`

Alternativen

`read_image`

Siehe auch

`read_image`

Modul

Foundation

serialize_image (Image : : : SerializedItemHandle)

Serialisieren eines Bildobjekts.

serialize_image serialisiert ein Bildobjekt (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das Bildobjekt wird in dem Parameter Image übergeben. Das serialisierte Bildobjekt wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_image wieder deserialisiert werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / complex / int2 / uint2 / vector_field / int4 / int8 / real
Bildobjekt.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_image den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, read_image

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_image

Modul

Foundation

write_image (Image : : Format, FillColor, FileName :)

Schreiben von Bildern in Graphikformaten.

write_image gibt das angegebene Bild (Image) in verschiedenen Bildformaten in Dateien aus. Falls die Domäne im gewünschten Format nicht abgespeichert werden kann (das ist der Fall bei 'bmp', 'jpeg' und 'ima'), erhalten die Pixel außerhalb der Region die durch FillColor definierte Farbe. Für Grauwertbilder muss ein Wert zwischen 0 (Schwarz) und 255 (Weiß) angegeben werden, bei Farbbildern können direkt die RGB-Werte angegeben werden, z.B. hexadezimal: 0xffff00 für einen gelben Hintergrund (Rot=255, Grün=255, Blau=0).

Derzeit werden folgende Formate unterstützt:

'tiff', 'bigtiff': TIFF-Format, Dateiendung: .tif

Alle in HALCON definierten Pixeltypen werden unterstützt. Bildobjekttupel mit bis zu 65535 Bildobjekten mit 65535 Kanälen können abgespeichert werden. Bildobjekte mit Kanälen gemischten Pixeltyps können abgespeichert werden.

Komprimierung ist möglich mit

'deflate [num]':	Adobe-Deflate-Komprimierung (verlustfrei)
'jpeg [num]':	JPEG (verlustbehaftet)
'lzw':	LZW (verlustfrei)
'packbits':	PackBits (verlustfrei)

'[num]' bedeutet hier eine optionale Angabe der Komprimierung. Bei *'deflate'* kann eine Zahl zwischen *0* (keine Komprimierung) und *9* (maximale Komprimierung) angegeben werden. Bei *'jpeg'* kann eine Zahl zwischen *0* und *100* angegeben werden. Die Semantik ist identisch mit der unten bei 'jpeg' beschriebenen Semantik. *'jpeg'* kann nur bei Bildern vom Typ byte oder int1 verwendet werden. Außerdem können nur Bilder mit höchstens vier Kanälen geschrieben werden.

Abspeichern der Region jedes Bildobjektes ist in komprimierter Form über *'mask'* (Standardeinstellung) oder als Alphakanal über *'alpha'* möglich. Die Region wird nur abgespeichert, falls sie nicht das gesamte Bild umfasst. Die gleichzeitige Wahl von *'jpeg'* und *'alpha'* ist nicht möglich, da durch die verlustbehaftete Komprimierung die Region nicht mehr korrekt rekonstruiert werden könnte. Die Optionen können durch Leerzeichen getrennt aneinander gefügt werden.

Beispiele:

'tiff deflate 9':	Maximale Adobe-Deflate-Komprimierung, die Region wird falls nötig als Maskenbild abgespeichert
'tiff jpeg 90':	JPEG-Komprimierung mit hoher Qualität, die Region wird falls nötig als Maskenbild abgespeichert
'tiff lzw alpha':	LZW-Komprimierung, die Region wird falls nötig als Alpha-Kanal abgespeichert
'tiff' oder *'tiff none'*	Keine Komprimierung, die Region wird falls nötig als Maskenbild abgespeichert

Es ist zu beachten, dass *'bigtiff'* TIFF-Dateien bezeichnet, die größer als 4 GB sein können, während *'tiff'* TIFF-Dateien bezeichnet, die maximal 4 GB groß sein können. Die Dateigröße hängt von der Bildgröße und der verwendeten Kompression ab. Deshalb sollte *'bigtiff'* verwendet werden, falls die Möglichkeit besteht, dass die komprimierte Datei größer als 4 GB wird.

'bmp': Windows-BMP-Format, Dateiendung *.bmp

Dieses Format kann nur Bilder mit einem Kanal (Grauwertbilder) oder mit drei Kanälen (RGB-Farbbilder) speichern.

Für dieses Format werden nur Bilder mit dem Pixeltyp byte unterstützt.

'jpeg': JPEG-Format (verlustbehaftete Komprimierung), Dateiendung *.jpg

Zusammen mit dem Format kann ein Qualitätsmaß angeben werden, das die Komprimierungsrate und die Qualität des gespeicherten Bildes bestimmt. Große Werte (Maximum 100) erzeugen eine relativ große Datei mit hoher Qualität, bei kleine Werten nimmt die Qualität und die Dateigröße deutlich ab. Beispiel: *'jpeg 30'*.

Dieses Format kann nur Bilder mit einem Kanal (Grauwertbilder) oder mit drei Kanälen (RGB-Farbbilder) speichern.

Für dieses Format werden nur Bilder mit dem Pixeltyp byte unterstützt.

Achtung: Bilder, die zu einem späteren Zeitpunkt noch ausgewertet werden müssen, sollten aufgrund der Verluste nicht im JPEG-Format abgespeichert werden!

'jp2': JPEG-2000-Format (verlustfreie und verlustbehaftete Komprimierung), Dateiendung *.jp2

Zusammen mit dem Format kann ein Maß angegeben werden, welches die Komprimierungsrate und damit die Qualität des gespeicherten Bildes bestimmt (z.B. *'jp2 40'*). Der angegebene Wert entspricht ungefähr dem Verhältnis der Größe des komprimierten Bildes zur Größe des unkomprimierten Bildes (in Prozent). Da JPEG-2000 ohnehin stark komprimiert, haben erst kleinere Werte (typischerweise unter 50) Einfluss auf die Dateigröße. Wenn kein Wert angegeben wird (und nur dann), wird das Bild verlustfrei komprimiert.

Das zu speichernde Bild kann beliebig viele Kanäle enthalten. Als Typen sind byte, cyclic, direction, int1, uint2, int2 und int4 zugelassen, wobei im Falle von int4 nur eine Farbtiefe von 24 Bit möglich ist (wird diese überschritten, wird eine Fehlerbehandlung ausgelöst). Falls ein Bild mit eingeschränktem Definitionsbereich übergeben wird, wird die Region als Alphakanal mit einer Farbtiefe von einem Bit abgespeichert.

Die Kodierung eines Bildes ins JPEG-2000 Format benötigt sehr viel Speicher. Es wird deshalb empfohlen, für große Bilder ein anderes Format (z.B. 'tiff') zu verwenden. Die maximal unterstützte Bildgröße (Breite x Höhe) für JPEG-2000 beträgt $2^{31} - 1$ auch in HALCON-XL.

'jpegxr': JPEG-XR-Format (verlustfreie und verlustbehaftete Komprimierung), Dateiendung *.jxr

Zusammen mit dem Format kann ein Maß angegeben werden, welches die Komprimierungsrate und damit die Qualität des gespeicherten Bildes bestimmt (z.B. *'jpegxr 30'* oder *'jpegxr 100'* bzw. *'jpegxr'* für verlustfreie Komprimierung). Alle HALCON-Bildtypen außer int8 werden unterstützt. Es können Bildobjekttupel mit einer beliebigen Anzahl von Bildobjekten geschrieben werden. Bilder können bis zu 8 Kanäle aufweisen. Im Falle von int4 und float Bildern wird die numerische Genauigkeit auf 24 Bit eingeschränkt. Dies kann zu Verlusten führen unabhängig von der Einstellung für die Qualität. Wird ein Bild mit eingeschränktem Definitionsbereich geschrieben, wird die Region verlustfrei als 1-bit Alphakanal gespeichert. Die Bildtypen complex, vector_field und Zweikanalbilder werden mit einem leeren dritten Kanal abgespeichert.

Es ist zu beachten, dass Bilder, unabhängig vom Zielverzeichnis, nicht im JPEG-XR gespeichert werden können, falls keine Schreibrechte für das aktuelle Arbeitsverzeichnis vorliegen.

'png': PNG-Format (verlustfreie Komprimierung), Dateiendung *.png

Zusammen mit dem Format kann ein Komprimierungsgrad zwischen 0 und 9 angegeben werden, wobei 0 keiner Komprimierung und 9 der bestmöglichen Komprimierung entspricht. Alternativ kann die Komprimierung über folgende Konstanten ausgewählt werden: *'best'*, *'fastest'* und *'none'*. Korrekte Formatparameter sind also z.B. *'png'*, *'png 7'* und *'png none'*.

In PNG-Dateien können Bilder vom Typ byte und uint2 abgespeichert werden. Falls ein Bild mit eingeschränktem Definitionsbereich übergeben wird, wird die Region als Alpha-Kanal abgespeichert, wobei die Punkte innerhalb des Definitionsbereichs mit dem maximalen Grauwert des Eingabebildtyps abgespeichert werden und die Punkte außerhalb des Definitionsbereichs mit dem Grauwert 0. Falls ein Bild mit vollem Definitionsbereich übergeben wird, wird kein Alpha-Kanal abgespeichert.

Dieses Format kann nur Bilder mit einem Kanal (Grauwertbilder) oder mit drei Kanälen (Farbbilder) speichern. Die maximal unterstützte Bildgröße (Breite x Höhe) für PNG beträgt $2^{31} - 1$ auch in HALCON-XL.

'hobj': HALCON-Format, Dateiendung *.hobj

Alle HALCON-Bildtypen werden unterstützt. Die Beschreibung des Operators `write_object` enthält weitere Informationen über das HOBJ-Format.

'ima': HALCON-Format, Dateiendung *.ima und *.exp

Diese Format gilt als veraltet, und es sollte stattdessen das HOBJ-Format verwendet werden. Die Daten werden binär und zeilensequentiell (ohne Header oder Zeilentrenner) geschrieben. Die Größe des Bildes und der Pixeltyp wird in der Beschreibungsdatei *'FileName.exp'* abgelegt. Alle HALCON-Pixeltypen außer *'complex'* und *'vector_field'* können geschrieben werden. Es können nur einkanalige Bilder gespeichert werden.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / complex / int2 / uint2 / vector_field / int4 / int8 / real
Zu speichernde Bilder.

▷ **`Format`** (input_control) ..... string ⇝ *string*
Graphikformat.
Defaultwert: 'tiff'
Wertevorschläge: `Format` ∈ {'tiff', 'tiff mask', 'tiff alpha', 'tiff deflate 9', 'tiff deflate 9 alpha', 'tiff jpeg 90', 'tiff lzw', 'tiff lzw alpha ', 'tiff packbits', 'bigtiff', 'bigtiff mask', 'bigtiff alpha', 'bigtiff deflate 9', 'bigtiff deflate 9 alpha', 'bigtiff jpeg 90', 'bigtiff lzw', 'bigtiff lzw alpha ', 'bigtiff packbits', 'bmp', 'jpeg', 'jpeg 100', 'jpeg 80', 'jpeg 60', 'jpeg 40', 'jpeg 20', 'jp2', 'jp2 50', 'jp2 40', 'jp2 30', 'jp2 20', 'jpegxr', 'jpegxr 50', 'jpegxr 40', 'jpegxr 30', 'jpegxr 20', 'png', 'png best', 'png fastest', 'png none', 'ima', 'hobj'}

▷ **`FillColor`** (input_control) ..... number ⇝ *integer* / real
Füllgrauwert für Bildpunkte, die nicht zur Domäne (Region) der Bilder gehören.
Defaultwert: 0
Wertevorschläge: `FillColor` ∈ {-1, 0, 255, 65280, 16711680}

▷ **`FileName`** (input_control) ..... filename.write(-array) ⇝ *string*
Name der Bilddatei.
Dateiendung: `.hobj`, `.ima`, `.tif`, `.tiff`, `.bmp`, `.jpg`, `.jpeg`, `.jp2`, `.jxr`, `.png`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_image` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, read_image

Modul

Foundation

write_image_metadata (: : Format, TagName, TagValue, FileName :)

Schreiben von Metadaten in eine Bilddatei.

Der Operator write_image_metadata schreibt die Informationen in TagValue unter den Tags TagName in die Datei FileName. Dabei können TagValue und TagName einzelne Werte oder Tupel von Werten sein. Der Parameter Format gibt an, in welchem Format die Metadaten geschrieben werden sollen.

Der Operator unterstützt für Format die TIFF Metadatenformate tiff und bigtiff. Dabei wird eine Dateiendung .tif vorausgesetzt.

Eine Auflistung der unterstützten Tag-Namen TagName und dazugehörigen Datentypen von TagValue findet sich unter read_image_metadata.

Parameter

▷ **Format** (input_control) . . . . . string ⇝ *string*
Grafikformat.
Defaultwert: 'tiff'
Wertevorschläge: Format ∈ {'tiff', 'bigtiff'}

▷ **TagName** (input_control) . . . . . tuple ⇝ *string*
Name des zu schreibenden Metadaten-Eintrags.
Defaultwert: 'tiff_image_description'
Wertevorschläge: TagName ∈ {'tiff_image_description', 'tiff_make', 'tiff_software', 'tiff_date_time', 'tiff_copyright', 'tiff_exposure_time', 'tiff_aperture_value', 'tiff_light_source'}

▷ **TagValue** (input_control) . . . . . tuple ⇝ *string* / integer / real
Wert des zu schreibenden Metadaten-Eintrags.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Bilddatei.
Dateiendung: .tif, .tiff

Ergebnis

Kann TagValue für die angegebenen Parameter erfolgreich geschrieben werden, dann liefert der Operator write_image_metadata den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation, wie z.B. der verursachende Tag, gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

write_image

Nachfolger

read_image_metadata

Modul

Foundation

7.2 Objekt

deserialize_object (: Object : SerializedItemHandle :)

Deserialisieren eines serialisierten ikonischen Objektes.

deserialize_object deserialisiert ein ikonisches Objekt, welches mit serialize_object serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte ikonische Objekt wird in dem Parameter SerializedItemHandle übergeben. Das deserialisierte ikonische Objekt wird in dem durch den Parameter Object definierten ikonischen Objekt gespeichert.

Parameter

- ▷ **Object** (output_object) . . . . . object(-array) ⇝ *object*
  Ikonisches Objekt.
- ▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_object den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_object

Siehe auch

serialize_object

Modul

Foundation

read_object (: Object : FileName :)

Lesen eines ikonischen Objektes.

read_object liest aus der Datei FileName ein ikonisches Objekt. Die Datei muss im HALCON-Iconic-Object-Format (HOBJ) geschrieben worden sein, welches von den Operatoren write_object, write_image und write_region unterstützt wird. Die Beschreibung des Operators write_object enthält weitere Informationen über das HOBJ-Format.

Parameter

- ▷ **Object** (output_object) . . . . . object(-array) ⇝ *object*
  Ikonisches Objekt.
- ▷ **FileName** (input_control) . . . . . filename ⇝ *string*
  Name der Datei.
  Dateiendung: .hobj

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_object den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`write_image`, `write_object`

Siehe auch

`write_object`, `write_image`, `read_image`

Modul

Foundation

serialize_object (Object : : : SerializedItemHandle)

Serialisieren eines ikonischen Objektes.

`serialize_image` serialisiert ein ikonisches Objekt (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das ikonische Objekt ist ein Tupel von Bildobjekten, Regionen oder XLD Objekten, und wird in dem Parameter `Object` übergeben. Das serialisierte ikonische Objekt wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_object` wieder deserialisiert werden.

Parameter

▷ **`Object`** (input_object) ..... object(-array) ⇝ *object*
Ikonisches Objekt.

▷ **`SerializedItemHandle`** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_object` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_object`

Siehe auch

`deserialize_object`

Modul

Foundation

write_object (Object : : FileName :)

Schreiben eines ikonischen Objektes.

`write_object` schreibt das ikonische Objekt `Object` in die Datei `FileName`. Das ikonische Objekt ist ein (möglicherweise gemischtes) Tupel von Bildern, Regionen oder XLDs. Falls keine Dateiendung angegeben ist, wird die Dateiendung *'.hobj'* angehängt. Die ikonischen Daten werden im HALCON Iconic Object Format (HOBJ) geschrieben, welches unten beschrieben wird.

HALCON Iconic Object (HOBJ) (HALCON 12.0 oder neuer): HOBJ ist ein binäres Dateiformat, welches die Funktionalität besitzt, alle ikonischen HALCON-Objekte (Bilder, Regionen oder XLDs) zu schreiben und zu lesen. Dadurch, dass die Daten ohne Komprimierung und ohne Konvertierung geschrieben werden, wird dieses Format unter normalen Umständen schneller geschrieben als andere unterstützte Formate. Sollte ein Anwendungsfall es erforderlich machen, dass verschiedene ikonische HALCON-Objekte möglichst effizient

geschrieben werden müssen, ohne dass Komprimierung benötigt wird, so ist das HOBJ-Format zu bevorzugen. Die standardmäßige Dateiendung für dieses Format ist *'.hobj'*. Für zu schreibende Bilder gilt, dass alle HALCON-Pixel-Typen und mehrkanalige Bilder unterstützt werden. Die Kanäle können gemischte Pixeltypen haben, allerdings müssen sie die gleiche Breite und Höhe aufweisen. Die Domäne und das Erstellungsdatum des Bildes werden ebenfalls in der Datei abgespeichert. Ein Tupel von Objekten wird in eine einzelne Datei geschrieben.

Parameter

▷ **Object** (input_object) . . . . . object(-array) ⇝ *object*
Ikonisches Objekt.

▷ **FileName** (input_control) . . . . . filename ⇝ *string*
Name der Datei.
Dateiendung: `.hobj`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_object` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`read_object`, `read_image`

Siehe auch

`read_object`, `write_image`, `read_image`, `write_region`, `read_region`

Modul

Foundation

7.3 Region

deserialize_region (: Region : SerializedItemHandle :)

Deserialisieren einer serialisierten Region.

`deserialize_region` deserialisiert eine Region, welche mit `serialize_region` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte Region wird in dem Parameter `SerializedItemHandle` übergeben. Die deserialisierte Region wird in dem durch den Parameter `Region` definierten Region gespeichert.

Parameter

▷ **Region** (output_object) . . . . . region(-array) ⇝ *object*
Region.

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_region` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_image, fread_serialized_item, receive_serialized_item, serialize_region

Nachfolger

reduce_domain, disp_region

Siehe auch

serialize_region

Modul

Foundation

read_region (: Region : FileName :)

Einlesen von Binärbildern oder HALCON-Regionen.

read_region liest Regionen aus einer Binärdatei.

Folgende Datei-Formate werden unterstützt:

HALCON Iconic Object (HOBJ) (HALCON 12.0 oder neuer): Dateiformat von HALCON zum Speichern von ikonischen Objekten. Die HOBJ-Datei darf ausschließlich Regionen enthalten um für read_region lesbar zu sein. Falls andere Objekte (z.B. Bilder oder XLDs) enthalten sind, wird ein Fehler erzeugt. In diesem Fall kann die Datei mit read_object gelesen werden. Das HOBJ-Format ist das Standardformat für Regionen. Daher muss die Extension *'.hobj'* beim Schreiben und Lesen nicht angegeben werden. Die Beschreibung des Operators write_object enthält weitere Informationen über das HOBJ-Format.

HALCON-Regionen: Dateiformat von HALCON zum Speichern von Regionen. In diesem Format können auch mehrere Regionen mit write_region bzw. read_region gleichzeitig (in einer Datei) gespeichert bzw. gelesen werden. Alle Regionendateien haben die Extension *'.reg'*. Dieses Dateiformat wird nur noch aus Kompatibilitätsgründen weiter unterstützt. Stattdessen sollte das HOBJ-Format verwendet werden.

Tiff: Binäre Tiff-Bilder mit der Extension *'tiff'* oder *'tif'*. Eine Tiffdatei kann dabei beliebig viele Regionen enthalten. Die Farbe Weiß wird als Vordergrund interpretiert.

PNG: PNG-Format mit der Extension *'png'*. Man erhält immer $eine$ Region. Die Farbe Weiß wird als Vordergrund interpretiert.

BMP: Windows Bitmap-Format mit der Extension *'bmp'*. Man erhält immer $eine$ Region. Die Farbe Weiß wird als Vordergrund interpretiert.

Analog zu read_image kann ein Suchpfad (*'image_dir'*) definiert werden.

Achtung

Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'true'/'false'>) festgelegt. Falls also vor dem Aufruf von read_region noch kein Bild hinreichender Größe erzeugt worden ist, sollte set_system('clip_region','false') vor read_region aufgerufen werden, um sicherzustellen, dass die Region nicht geclippt wird.

Parameter

▷ **Region** (output_object) ..... region(-array) ⇝ *object*
Eingelesene Region.

▷ **FileName** (input_control) ..... filename.read ⇝ *string*
Name der einzulesenden Region.
Dateiendung: .hobj, .reg, .tif, .tiff, .png, .bmp

Beispiel

```
* Reading of regions and giving them gray values.
  read_image(Img,'ima_test')
  read_region(Regs,'reg_test')
  reduce_domain(Img,Regs,Res)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_region` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`

Nachfolger

`reduce_domain`, `disp_region`

Siehe auch

`write_region`, `read_image`, `write_object`

Modul

Foundation

serialize_region (Region : : : SerializedItemHandle)

Serialisieren einer Region.

`serialize_region` serialisiert eine Region (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Die Region wird in dem Parameter `Region` übergeben. Die serialisierte Region wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_region` wieder deserialisiert werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Region.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_region` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `read_image`, `read_region`, `threshold`, `regiongrowing`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_region`

Siehe auch

`deserialize_region`

Modul

Foundation

write_region (Region : : FileName :)

Schreiben von Regionen auf Datei.

write_region gibt die Regionen in Region in eine Binärdatei aus. Die Regionen können im HALCON-Iconic-Object-Format (HOBJ), als TIFF-Dateien oder im HALCON-Regionen-Format abgespeichert werden. Die Auswahl erfolgt über die Dateiendung von FileName. Falls keine Dateiendung oder die Dateiendung *'.hobj'* verwendet wird, werden die Regionen im HOBJ-Format abgespeichert. Falls keine Dateiendung angegeben wird, wird die Dateiendung *'.hobj'* angehängt. Falls die Dateiendung *'.tif'* oder *'.tiff'* verwendet wird, werden die Regionen als TIFF-Dateien abgespeichert. Falls die Dateiendung *'.reg'* verwendet wird, werden die Regionen als HALCON-Regionen-Datei abgespeichert. Das HALCON-Regionen-Format gilt als veraltet, und es sollte stattdessen das HOBJ-Format verwendet werden. Es können Bildobjekttupel mit beliebig vielen Regionen geschrieben werden. Die ausgegebenen Daten können mit read_region wieder eingelesen werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Bilder, deren Regionen ausgegeben werden.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Regionendatei.
Defaultwert: 'region.hobj'
Dateiendung: .hobj, .reg, .tif, .tiff

Beispiel

```
regiongrowing(Img,Segments,3,3,5,10)
write_region(Segments,'result1')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_region den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, read_image, read_region, read_object, threshold, regiongrowing

Siehe auch

read_region

Modul

Foundation

7.4 Sonstiges

copy_file (: : SourceFile, DestinationFile :)

Kopieren einer Datei.

copy_file kopiert SourceFile nach DestinationFile. Bereits existierende Dateien werden ohne Nachfrage überschrieben. Die Dateiattribute von SourceFile werden übernommen.

Parameter

▷ **SourceFile** (input_control) ..... filename.read ⇝ *string*
Zu kopierende Datei.

▷ **DestinationFile** (input_control) ..... filename.write ⇝ *string*
Zielort für die Kopie.

Ergebnis

`copy_file` liefert den Wert TRUE, falls die Datei kopiert werden konnte. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

delete_file (: : FileName :)

Löschen einer Datei.

`delete_file` löscht die durch `FileName` angegebene Datei.

Parameter

▷ **FileName** (input_control) . . . . . filename ⇝ *string*
Name der zu prüfenden Datei.

Ergebnis

`delete_file` liefert den Wert TRUE, falls die Datei existiert und gelöscht werden konnte. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

file_exists (: : FileName : FileExists)

Test ob Datei existiert.

`file_exists` überprüft, ob die angegebene Datei bereits existiert. Ist dies der Fall, wird `FileExists` auf TRUE, anderenfalls auf FALSE gesetzt.

Parameter

▷ **FileName** (input_control) . . . . . filename ⇝ *string*
Name der zu prüfenden Datei.
Defaultwert: '/bin/cc'

▷ **FileExists** (output_control) . . . . . integer ⇝ *integer*
boolescher Wert.

Ergebnis

Wenn die Parameter korrekt sind, liefert `file_exists` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`open_file`

Alternativen

`open_file`

Modul

Foundation

get_current_dir (: : : DirName)

Auslesen des aktuellen Arbeitsverzeichnisses.

`get_current_dir` liefert das aktuelle Arbeitsverzeichnis in `DirName` zurück.

Parameter

▷ **DirName** (output_control) ..... filename.dir ⇝ *string*
Name des aktuellen Arbeitsverzeichnisses.

Ergebnis

`get_current_dir` liefert den Wert TRUE, falls das aktuelle Arbeitsverzeichnis ausgelesen werden konnte. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

list_files (: : Directory, Options : Files)

Auflisten aller Dateien in einem Verzeichnis.

`list_files` gibt in `Files` alle Dateien im Verzeichnis `Directory` zurück. Das aktuelle Verzeichnis kann mit '' oder '.' angegeben werden. Mit dem Parameter `Options` können verschiedene Ausführungsoptionen festgelegt werden, indem ein Tupel von Werten übergeben wird. Wird in `Options` *'files'* übergeben, so werden ausschließlich die in `Directory` vorhandenen Dateien zurückgegeben. Wird *'directories'* übergeben, so werden ausschließlich die vorhandenen Verzeichnisse zurückgegeben. Die Verzeichnisse werden dabei durch ein abschließendes '\' (Windows) bzw. '/' (unixartige Systeme) gekennzeichnet. Falls sowohl Dateien als auch Verzeichnisse zurückgegeben werden sollen, muss *['files','directories']* übergeben werden. Falls weder *'files'* noch *'directories'* übergeben wird, liefert `list_files` ein leeres Tupel zurück. Mit *'recursive'* kann festgelegt werden, dass der Verzeichnisbaum rekursiv unter Einbeziehung aller Unterverzeichnisse durchsucht werden soll. Unter unixartigen Systemen kann mit *'follow_links'* festgelegt werden, ob symbolische Links auf Dateien oder Verzeichnisse verfolgt werden oder nicht. In der Voreinstellung ohne *'follow_links'* werden symbolische Links nicht dereferenziert und somit auch nicht verfolgt, falls sie auf Verzeichnisse zeigen, oder zurückgeliefert, falls sie auf Dateien zeigen. Bei der rekursiven Suche kann mit *'max_depth <d>'* eine maximale Tiefe der Suche festgelegt werden, wobei *'<d>'* eine Zahl ist, die die Tiefe festlegt. So legt z.B. *'max_depth 2'* fest, dass `Directory` und alle direkten Unterverzeichnisse von `Directory` durchsucht werden. Falls symbolische Links verfolgt werden sollen, könnte es sein, dass die Suche nicht terminiert, falls die symbolischen Links zu einem Zyklus in der Verzeichnisstruktur führen. Deswegen werden in `Files` höchstens 1000000 Dateien (und Verzeichnisse) zurückgegeben. Mit *'max_files <d>'* kann dieser Wert herabgesetzt werden.

Parameter

▷ **Directory** (input_control) . . . . . filename.dir ⇝ *string*
Name des aufzulistenden Verzeichnisses.

▷ **Options** (input_control) . . . . . string(-array) ⇝ *string*
Ausführungsoptionen.
Defaultwert: 'files'
Wertevorschläge: `Options` ∈ {'files', 'directories', 'recursive', 'follow_links', 'max_depth 5', 'max_files 1000'}

▷ **Files** (output_control) . . . . . string-array ⇝ *string*
Gefundene Dateien (und Verzeichnisse).

Ergebnis

`list_files` liefert den Wert TRUE, falls das zu untersuchende Verzeichnis existiert und gelesen werden konnte. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`tuple_regexp_select`

Modul

Foundation

make_dir (: : DirName :)

Anlegen eines Verzeichnisses.

`make_dir` legt ein Verzeichnis mit dem Namen `DirName` an.

Parameter

▷ **DirName** (input_control) . . . . . filename.dir ⇝ *string*
Name des anzulegenden Verzeichnisses.

Ergebnis

`make_dir` liefert den Wert TRUE, falls das Verzeichnis angelegt werden konnte. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

read_world_file (: : FileName : WorldTransformation)

Lesen der Geokodierung aus einem ARC/INFO World File.

`read_world_file` liest die Geokodierung aus dem ARC/INFO World File mit Namen `FileName` aus, und liefert sie als homogene 2D-Transformationsmatrix in `WorldTransformation` zurück. Dabei werden alle Verzeichnisse, die in der HALCON-Systemvariablen *'image_dir'* bzw. in der Umgebungsvariablen HALCONIMAGES enthalten sind, nach der Datei `FileName` durchsucht (siehe

read_image). Die Transformationsmatrix kann verwendet werden, um XLD-Konturen vor dem Schreiben mit write_contour_xld_arc_info in Weltkoordinaten zu transformieren. Wenn die Matrix WorldTransformation mit hom_mat2d_invert invertiert wird, kann die so entstehende Matrix verwendet werden, um Konturen, die mit read_contour_xld_arc_info eingelesen worden sind, in das Bildkoordinatensystem zu transformieren.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name des ARC/INFO World Files.

▷ **WorldTransformation** (output_control) . . . . . hom_mat2d ⇝ *real*
Transformationsmatrix von Bild- in Weltkoordinaten.

Ergebnis

Wenn die Parameter korrekt sind und das angegebene World File gelesen werden konnte, liefert read_world_file den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

hom_mat2d_invert, affine_trans_contour_xld, affine_trans_polygon_xld

Siehe auch

write_contour_xld_arc_info, read_contour_xld_arc_info, write_polygon_xld_arc_info, read_polygon_xld_arc_info

Modul

Foundation

remove_dir (: : DirName :)

Löschen eines leeren Verzeichnisses.

remove_dir löscht das leere Verzeichnis mit dem Namen DirName.

Parameter

▷ **DirName** (input_control) . . . . . filename.dir ⇝ *string*
Name des zu löschenden Verzeichnisses.

Ergebnis

remove_dir liefert den Wert TRUE, falls das Verzeichnis gelöscht werden konnte. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

set_current_dir (: : DirName :)

Setzen des aktuellen Arbeitsverzeichnisses.

set_current_dir setzt das aktuelle Arbeitsverzeichnis auf das Verzeichnis DirName.

Parameter

▷ **DirName** (input_control) ..... filename.dir ⇝ *string*
Name des zu setzenden aktuellen Arbeitsverzeichnisses.

Ergebnis

set_current_dir liefert den Wert TRUE, falls das aktuelle Arbeitsverzeichnis gesetzt werden konnte. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

7.5 Tupel

deserialize_handle (: : SerializedItem : Handle)

Deserialisieren eines Handles.

deserialize_handle deserialisiert den Inhalt von SerializedItem und gibt das deserialisierte Objekt in Handle zurück (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung).

Das serialisierte Objekt muss mit serialize_handle oder einem typspezifischen Serialisierungsoperator wie serialize_matrix oder serialize_shape_model erstellt worden sein.

Parameter

▷ **SerializedItem** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **Handle** (output_control) ..... handle ⇝ *handle*
Handle des deserialisierten Elements.

Beispiel

```
create_matrix (3, 3, 0, MatrixID)
serialize_handle (MatrixID, SerializedMatrix)
deserialize_handle (SerializedMatrix, MatrixID2)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_handle den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

serialize_handle, fread_serialized_item, receive_serialized_item

Modul

Foundation

deserialize_tuple (: : SerializedItemHandle : Tuple)

Deserialisieren eines serialisierten Tupels.

deserialize_tuple deserialisiert Daten eines Tupels, welches mit serialize_tuple serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierten Daten des Tupels werden in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Tupel mit dem Handle Tuple gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **Tuple** (output_control) . . . . . tuple(-array) ⇝ *real* / integer / string / handle
Tupel.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_tuple den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_tuple

Modul

Foundation

read_tuple (: : FileName : Tuple)

Lesen eines Tupels aus einer Datei.

read_tuple liest den Inhalt der Datei FileName und wandelt diesen in das Tupel Tuple um. Die Dateiendung für Tupel ist in HALCON 'tup'. Die Datei muss mit write_tuple erzeugt worden sein.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der zu lesenden Datei.
Dateiendung: .tup

▷ **Tuple** (output_control) . . . . . tuple(-array) ⇝ *real* / integer / string
Tupel mit beliebigen Werten.

Ergebnis

Sind die Parameter korrekt, dann liefert read_tuple den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

fwrite_string

Siehe auch

write_tuple, write_image, write_region, open_file

Modul

Foundation

serialize_handle (: : Handle : SerializedItem)

Serialisieren des Inhalts eines Handles.

serialize_handle serialisiert den Inhalt von Handle und gibt ihn als serialisiertes Objekt in SerializedItem zurück (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Objekt kann später mit deserialize_handle wieder deserialisiert werden.

Nicht alle Handletypen können serialisiert werden. Ob ein Handle serialisiert werden kann, kann zur Laufzeit mit tuple_is_serializable oder tuple_is_serializable_elem geprüft werden. Bereits freigegebene Handles können nicht serialisiert werden.

Die von diesem Operator erzeugten serialisierten Objekte sind kompatibel mit den typspezifischen Deserialisierungsoperatoren, wie beispielsweise deserialize_matrix oder deserialize_shape_model.

Parameter

- ▷ **Handle** (input_control) . . . . . handle ⇝ *handle*
  Zu serialisierendes Handle.
- ▷ **SerializedItem** (output_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Beispiel

```
create_matrix (3, 3, 0, MatrixID)
serialize_handle (MatrixID, SerializedMatrix)
deserialize_handle (SerializedMatrix, MatrixID2)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_handle den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

tuple_is_serializable, tuple_is_serializable_elem

Nachfolger

deserialize_handle, fwrite_serialized_item, send_serialized_item

Modul

Foundation

serialize_tuple (: : Tuple : SerializedItemHandle)

Serialisieren eines Tupels.

serialize_tuple serialisiert die Daten eines Tupels (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_tuple in eine Datei geschrieben werden. Das Tupel wird in dem Handle Tuple übergeben. Die serialisierten Daten des Tupels werden in dem Handle SerializedItemHandle zurückgegeben und können mit deserialize_tuple wieder deserialisiert werden.

Nicht alle Handle-Typen können serialisiert werden. Wenn Tuple ein Handle enthält, das nicht serialisiert werden kann oder das schon freigegeben wurde, bricht serialize_tuple mit einer Fehlermeldung ab. Die Operatoren tuple_is_serializable und tuple_is_serializable_elem können verwendet werden, um zu prüfen, ob ein Tupel oder dessen Einträge serialisiert werden können.

Parameter

▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *real* / integer / string / handle
Tupel.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_tuple` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_tuple`

Modul

Foundation

tuple_is_serializable (: : Tuple : IsSerializable)

Prüft ob ein Tupel serialisiert werden kann.

`tuple_is_serializable` prüft ob `Tuple` mit dem Operator `serialize_tuple` serialisiert werden kann. Wenn ja, wird `IsSerializable` auf *1* gesetzt, andernfalls auf *0*.

Ein Tupel kann serialisiert werden, wenn es nur ganze Zahlen, Gleitkommazahlen, Strings und serialisierbare Handles enthält. Handles die bereits freigegeben wurden können nicht serialisiert werden. Um die einzelnen Elemente eines Tupels auf Serialisierbarkeit zu prüfen, kann `tuple_is_serializable_elem` verwendet werden.

Parameter

▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *handle* / integer / real / string
Tupel das auf Serialisierbarkeit geprüft wird.

▷ **IsSerializable** (output_control) . . . . . number ⇝ *integer*
Kann das Eingabetupel serialisiert werden?

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_serializable` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`serialize_tuple`, `write_tuple`

Alternativen

`tuple_is_serializable_elem`

Siehe auch

`tuple_is_serializable_elem`, `serialize_tuple`, `serialize_handle`, `deserialize_tuple`, `deserialize_handle`, `write_tuple`

Modul

Foundation

tuple_is_serializable_elem (: : Tuple : IsSerializableElem)

Prüft ob die Elemente eines Tupel serialisiert werden können.

tuple_is_serializable prüft ob die Elemente von Tuple mit dem Operator serialize_tuple serialisiert werden können. Die Ausgabe IsSerializableElem hat die gleiche Länge wie Tuple und enthält an jeder Position eine *1* falls der entsprechende Eintrag in Tuple serialisiert werden kann, andernfalls eine *0*.

Ein Tupelelement kann serialisiert werden, wenn es eine ganze Zahle, eine Gleitkommazahl, ein String oder ein serialisierbares Handles ist. Handles die bereits freigegeben wurden können nicht serialisiert werden. Um das gesamte Tupel auf Serialisierbarkeit zu prüfen, kann tuple_is_serializable verwendet werden.

Parameter

▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *handle* / integer / real / string
Tupel dessen Elemente auf Serialisierbarkeit geprüft werden.

▷ **IsSerializableElem** (output_control) . . . . . number(-array) ⇝ *integer*
Können die Elemente des Eingabetupels serialisiert werden?

Beispiel

```
* Serialize all serializable elements of a tuple
tuple_is_serializable_elem (Tuple, IsSerializableElem)
if (sum(IsSerializableElem[=]0)>0)
  Tuple[find(IsSerializableElem,0)] := HNULL
endif
serialize_tuple (Tuple, SerializedItem)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert tuple_is_serializable_elem den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

serialize_tuple, write_tuple

Alternativen

tuple_is_serializable

Siehe auch

tuple_is_serializable, serialize_tuple, serialize_handle, deserialize_tuple, deserialize_handle, write_tuple

Modul

Foundation

write_tuple (: : Tuple, FileName :)

Schreiben des Inhalts eines Tupels in eine Datei.

write_tuple schreibt den Inhalt von Tuple auf Datei. Es handelt sich hierbei um ein ASCII Format, d.h. die Daten sind zwischen unterschiedlichen Rechnertypen austauschbar (siehe auch Tupel / String-Operationen). Die Daten können mit dem Operator read_tuple wieder eingelesen werden. Die Dateiendung für Tupel ist in HALCON 'tup'.

write_tuple kann keine Handles schreiben, da die entsprechenden binären Daten vom geschriebenen ASCII-Format nicht unterstützt werden. Um ein Tupel zu schreiben, das Handles enthält, können die Operatoren serialize_tuple und fwrite_serialized_item verwendet werden.

Parameter

▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *real / integer / string*
Tupel mit beliebigen Werten.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der zu schreibenden Datei.
Dateiendung: .tup

Ergebnis

Sind die Parameter korrekt, dann liefert write_tuple den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

fwrite_string

Siehe auch

read_tuple, write_image, write_region, open_file

Modul

Foundation

7.6 XLD

deserialize_xld (: XLD : SerializedItemHandle :)

Deserialisieren eines XLD-Objekts.

deserialize_xld deserialisiert ein XLD-Objekt, welches mit serialize_xld serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte XLD-Objekt wird in dem Parameter SerializedItemHandle übergeben. Das deserialisierte XLD-Objekt wird in dem durch den Parameter XLD definierten Objekt gespeichert.

Parameter

▷ **XLD** (output_object) . . . . . xld(-array) ⇝ *object*
XLD-Objekt.

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_image, fread_serialized_item, receive_serialized_item, serialize_xld

Nachfolger

clip_contours_xld, split_contours_xld

Siehe auch

serialize_xld

Modul

Foundation

read_contour_xld_arc_info (: Contours : FileName :)

Lesen von XLD-Konturen im ARC/INFO-Generate-Format.

read_contour_xld_arc_info liest die in der Datei FileName im ARC/INFO-Generate-Format gespeicherten Linien ein und liefert sie als XLD-Konturen in Contours zurück. Dabei werden alle Verzeichnisse, die in der HALCON-Systemvariablen *'image_dir'* bzw. in der Umgebungsvariablen HALCONIMAGES enthalten sind, nach der Datei FileName durchsucht (siehe read_image). Die eingelesenen Konturen liegen im Weltkoordinatensystem vor. Der Operator affine_trans_contour_xld kann verwendet werden, um sie in das Bildkoordinatensystem zu transformieren. Die dazu notwendige Transformationsmatrix kann erzeugt werden, indem mit read_world_file aus einem ARC/INFO World File die Transformationsmatrix von Bild- nach Weltkoordinatensystem gelesen wird, und diese mit hom_mat2d_invert invertiert wird.

Parameter

▷ **Contours** (output_object) ..... xld_cont(-array) ⇝ *object*
Eingelesene XLD-Konturen.

▷ **FileName** (input_control) ..... filename.read ⇝ *string*
Name der ARC/INFO-Datei.

Beispiel

```
* Read the transformation and invert it
read_world_file ('image.tfw', WorldTransformation)
hom_mat2d_invert (WorldTransformation, ImageTransformation)
* Read the image
read_image (Image, 'image.tif')
* Read the line data
read_contour_xld_arc_info (LinesWorld, 'lines.gen')
* Transform the line data to image coordinates
affine_trans_contour_xld (LinesWorld, Lines, ImageTransformation)
```

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei gelesen werden konnte, liefert read_contour_xld_arc_info den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

hom_mat2d_invert, affine_trans_contour_xld

Siehe auch

read_world_file, write_contour_xld_arc_info, read_polygon_xld_arc_info

Modul

Foundation

read_contour_xld_dxf (: Contours : FileName, GenParamName, GenParamValue : DxfStatus)

Lesen von XLD-Konturen aus einer DXF-Datei.

read_contour_xld_dxf liest den Inhalt der DXF-Datei FileName (DXF-Version AC1009, AutoCAD Release 12) und wandelt ihn in die XLD-Konturen Contours. Wenn kein absoluter Pfad in FileName angegeben wird, wird die DXF-Datei im aktuellen Verzeichnis des HALCON-Prozesses gesucht.

Der Ausgabeparameter DxfStatus enthält Informationen über die Anzahl der gelesenen Konturen und, falls erforderlich, Hinweise auf Teile der DXF-Datei, die nicht gelesen werden konnten.

Der Operator read_contour_xld_dxf unterstützt die folgenden DXF-Elemente:

- POLYLINE
  - 2D-Kurven, die aus Liniensegmenten bestehen
  - Geschlossene 2D-Kurven, die aus Liniensegmenten bestehen
- LWPOLYLINE
- LINE
- POINT
- CIRCLE
- ARC
- ELLIPSE
- SPLINE
- BLOCK
- INSERT

Die X- und Y-Koordinaten der DXF-Elemente werden in den Spalten- und Zeilenkoordinaten der XLD-Konturen `Contours` gespeichert. Die Z-Koordinaten der DXF-Elemente werden nicht berücksichtigt.

Wenn die Datei mit dem Operator `write_contour_xld_dxf` erstellt wurde, werden alle Attribute und globalen Attribute eingelesen, die für die ursprüngliche XLD-Kontur definiert waren. Das heißt, `read_contour_xld_dxf` unterstützt die erweiterten Elementdaten, die von `write_contour_xld_dxf` geschrieben werden. Das Einlesen dieser Attribute kann durch Setzen des generischen Parameters *'read_attributes'* auf *'false'* verhindert werden. Generische Parameter werden durch Angabe des Parameternamens in `GenParamName` und des dazugehörigen Wertes in `GenParamValue` gesetzt.

DXF-Elemente vom Typ CIRCLE, ARC, ELLIPSE und SPLINE werden durch XLD-Konturen approximiert. Die Genauigkeit dieser Approximation kann über die beiden generischen Parameter *'min_num_points'* und *'max_approx_error'* gesteuert werden. Der Parameter *'min_num_points'* legt fest, wie viele Stützpunkte mindestens für die Approximation verwendet werden. Der Parameter *'min_num_points'* bezieht sich immer auf den entsprechenden Vollkreis bzw. die gesamte Ellipse, auch für ARCs oder Ellipsenbögen, d.h., wenn *'min_num_points'* auf den Wert *50* gesetzt ist und ein DXF-Element vom Typ ARC eingelesen wird, das einen Halbkreis darstellt, so wird dieser durch mindestens 25 Stützpunkte approximiert. Der Parameter *'max_approx_error'* legt fest, wie groß die maximale Abweichung der XLD-Kontur von dem idealen Kreis bzw. der idealen Ellipse sein darf (Einheit: Pixel). Für die Bestimmung der Genauigkeit der Approximation werden beide Kriterien ausgewertet und dasjenige verwendet, das zu einer genaueren Approximation führt.

Die intern verwendeten Defaultwerte für die generischen Parameter sind:

- *'read_attributes' = 'true'*
- *'min_num_points' = 20*
- *'max_approx_error' = 0.25*

Um eine genauere Approximation zu erreichen, muss entweder ein größerer Wert für *'min_num_points'* oder ein kleinerer Wert für *'max_approx_error'* verwendet werden.

Parameter

▷ **`Contours`** (output_object) . . . . . xld_cont(-array) ⇝ *object*
Eingelesene XLD-Konturen.

▷ **`FileName`** (input_control) . . . . . filename.read ⇝ *string*
Name der DXF-Datei.
Dateiendung: `.dxf`

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für den DXF-Input eingestellt werden sollen.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'read_attributes', 'min_num_points', 'max_approx_error'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value(-array) ⇝ *real* / integer / string
Werte der generischen Parameter, die für den DXF-Input eingestellt werden sollen.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false', 0.1, 0.25, 0.5, 1, 2, 5, 10, 20}

▷ **DxfStatus** (output_control) . . . . . string(-array) ⇝ *string*
Statusinformationen.

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei gelesen werden konnte, liefert `read_contour_xld_dxf` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`write_contour_xld_dxf`

Siehe auch

`write_contour_xld_dxf`, `read_polygon_xld_dxf`, `query_contour_attribs_xld`, `query_contour_global_attribs_xld`, `get_contour_attrib_xld`, `get_contour_global_attrib_xld`

Modul

Foundation

read_polygon_xld_arc_info (: Polygons : FileName :)

Lesen von XLD-Polygonen im ARC/INFO-Generate-Format.

`read_polygon_xld_arc_info` liest die in der Datei `FileName` im ARC/INFO-Generate-Format gespeicherten Linien ein und liefert sie als XLD-Polygone in `Polygons` zurück. Dabei werden alle Verzeichnisse, die in der HALCON-Systemvariablen *'image_dir'* bzw. in der Umgebungsvariablen HALCONIMAGES enthalten sind, nach der Datei `FileName` durchsucht (siehe `read_image`). Die eingelesenen Polygone liegen im Weltkoordinatensystem vor. Der Operator `affine_trans_polygon_xld` kann verwendet werden, um sie in das Bildkoordinatensystem zu transformieren. Die dazu notwendige Transformationsmatrix kann erzeugt werden, indem mit `read_world_file` aus einem ARC/INFO World File die Transformationsmatrix von Bild- nach Weltkoordinatensystem gelesen wird, und diese mit `hom_mat2d_invert` invertiert wird.

Parameter

▷ **Polygons** (output_object) . . . . . xld_poly(-array) ⇝ *object*
Eingelesene XLD-Polygone.

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der ARC/INFO-Datei.

Beispiel

```
* Read the transformation and invert it
read_world_file ('image.tfw', WorldTransformation)
hom_mat2d_invert (WorldTransformation, ImageTransformation)
* Read the image
read_image (Image, 'image.tif')
* Read the line data
read_polygon_xld_arc_info (LinesWorld, 'lines.gen')
* Transform the line data to image coordinates
affine_trans_polygon_xld (LinesWorld, Lines, ImageTransformation)
```

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei gelesen werden konnte, liefert `read_polygon_xld_arc_info` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`hom_mat2d_invert`, `affine_trans_polygon_xld`

Siehe auch

`read_world_file`, `write_polygon_xld_arc_info`, `read_contour_xld_arc_info`

Modul

Foundation

read_polygon_xld_dxf (: Polygons : FileName, GenParamName, GenParamValue : DxfStatus)

Lesen von XLD-Polygonen aus einer DXF-Datei.

`read_polygon_xld_dxf` liest den Inhalt der DXF-Datei `FileName` (DXF-Version AC1009, AutoCAD Release 12) und wandelt ihn in die XLD-Polygone `Polygons`. Wenn kein absoluter Pfad in `FileName` angegeben wird, wird die DXF-Datei im aktuellen Verzeichnis des HALCON-Prozesses gesucht.

Der Ausgabeparameter `DxfStatus` enthält Informationen über die Anzahl der gelesenen Polygone und, falls erforderlich, Hinweise auf Teile der DXF-Datei, die nicht gelesen werden konnten.

Der Operator `read_polygon_xld_dxf` unterstützt die folgenden DXF-Elemente:

- POLYLINE
  - 2D-Kurven, die aus Liniensegmenten bestehen
  - Geschlossene 2D-Kurven, die aus Liniensegmenten bestehen
- LWPOLYLINE
- LINE
- POINT
- CIRCLE
- ARC
- ELLIPSE
- SPLINE
- BLOCK
- INSERT

Die X- und Y-Koordinaten der DXF-Elemente werden in den Spalten- und Zeilenkoordinaten der XLD-Polygone `Polygons` gespeichert. Die Z-Koordinaten der DXF-Elemente werden nicht berücksichtigt.

DXF-Elemente vom Typ CIRCLE, ARC, ELLIPSE und SPLINE werden durch XLD-Polygone approximiert. Die Genauigkeit dieser Approximation kann über die beiden generischen Parameter *'min_num_points'* und *'max_approx_error'* gesteuert werden (bei SPLINE nur *'max_approx_error'*). Generische Parameter werden durch Angabe des Parameternamens in `GenParamName` und des dazugehörigen Wertes in `GenParamValue` gesetzt. Der Parameter *'min_num_points'* legt fest, wie viele Stützpunkte mindestens für die Approximation verwendet werden. Der Parameter *'min_num_points'* bezieht sich immer auf den entsprechenden Vollkreis bzw. die gesamte Ellipse, auch für ARCs oder Ellipsenbögen, d.h., wenn *'min_num_points'* auf den Wert *50* gesetzt ist und ein DXF-Element vom Typ ARC eingelesen wird, das einen Halbkreis darstellt, so wird dieser durch mindestens 25 Stützpunkte dargestellt. Der Parameter *'max_approx_error'* legt fest, wie groß die maximale Abweichung des XLD-Polygons von dem idealen Kreis bzw. der idealen Ellipse sein darf (Einheit: Pixel). Für die Bestimmung der Genauigkeit der Approximation werden beide Kriterien ausgewertet und dasjenige verwendet, das zu einer genaueren Approximation führt.

Die intern verwendeten Defaultwerte für die generischen Parameter sind:

- *'min_num_points'* = 20

- *'max_approx_error'* = 0.25

Um eine genauere Approximation zu erreichen, muss entweder ein größerer Wert für *'min_num_points'* oder ein kleinerer Wert für *'max_approx_error'* verwendet werden.

Beachten Sie, dass das Einlesen einer DXF-Datei mit `read_polygon_xld_dxf` geometrisch zu dem gleichen Ergebnis führt wie das Einlesen der Datei mit `read_contour_xld_dxf`. Lediglich die Datenstruktur, in der das Ergebnis zurückgegeben wird, ist eine andere.

Parameter

▷ **Polygons** (output_object) . . . . . xld_poly(-array) ⇝ *object*
Eingelesene XLD-Polygone.

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der DXF-Datei.
Dateiendung: `.dxf`

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für den DXF-Input eingestellt werden sollen.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'min_num_points', 'max_approx_error'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *real* / integer / string
Werte der generischen Parameter, die für den DXF-Input eingestellt werden sollen.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {0.1, 0.25, 0.5, 1, 2, 5, 10, 20}

▷ **DxfStatus** (output_control) . . . . . string(-array) ⇝ *string*
Statusinformationen.

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei gelesen werden konnte, liefert `read_polygon_xld_dxf` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`write_polygon_xld_dxf`

Siehe auch

`write_polygon_xld_dxf`, `read_contour_xld_dxf`

Modul

Foundation

serialize_xld (XLD : : : SerializedItemHandle)

Serialisieren eines XLD-Objekts.

`serialize_xld` serialisiert ein XLD-Objekt (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das XLD-Objekt wird in dem Parameter `XLD` übergeben. Das serialisierte XLD-Objekt wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_xld` wieder deserialisiert werden.

Parameter

▷ **XLD** (input_object) . . . . . xld(-array) ⇝ *object*
Zu serialisiertes XLD-Objekt.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contour_polygon_xld`, `gen_parallels_xld`, `gen_polygons_xld`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_xld`

Siehe auch

`deserialize_xld`

Modul

Foundation

write_contour_xld_arc_info (Contours : : FileName :)

Schreiben von XLD-Konturen im ARC/INFO-Generate-Format.

`write_contour_xld_arc_info` schreibt die XLD-Konturen `Contours` im ARC/INFO-Generate-Format in die Datei `FileName`. Wenn kein absoluter Pfad in `FileName` angegeben wird, wird die Ausgabedatei im aktuellen Verzeichnis des HALCON-Prozesses geschrieben. Die Konturen müssen zuvor mit `affine_trans_contour_xld` in Weltkoordinaten transformiert worden sein. Die dazu notwendige Transformation kann mit `read_world_file` aus einem ARC/INFO World File eingelesen werden.

Parameter

▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
Zu schreibende XLD-Konturen.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der ARC/INFO-Datei.

Beispiel

```
* Read transformation and image
read_world_file ('image.tfw', WorldTransformation)
read_image (Image, 'image.tif')
* Segment image
* ...
* Write result
affine_trans_contour_xld (Contours, ContoursWorld, WorldTransformation)
write_contour_xld_arc_info (ContoursWorld, 'result.gen')
```

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei geschrieben werden konnte, liefert `write_contour_xld_arc_info` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

affine_trans_contour_xld

Siehe auch

read_world_file, read_contour_xld_arc_info, write_polygon_xld_arc_info

Modul

Foundation

write_contour_xld_dxf (Contours : : FileName :)

Schreiben von XLD-Konturen im DXF-Format.

write_contour_xld_dxf schreibt die XLD-Konturen Contours im DXF-Format in die Datei FileName. Wenn kein absoluter Pfad in FileName angegeben wird, wird die Ausgabedatei im aktuellen Verzeichnis des HALCON-Prozesses geschrieben.

Neben der Geometrie der Contours werden auch sämtliche Attribute und globalen Attribute, die für Contours definiert sind, in die Datei geschrieben.

write_contour_xld_dxf schreibt eine Datei im Format der DXF-Version AC1009 (AutoCAD Release 12). Jede Kontur wird als POLYLINE gespeichert. Die Attributwerte werden als erweiterte Elementdaten zu jedem VERTEX der POLYLINE geschrieben. Die globalen Attributwerte werden als erweiterte Elementdaten der POLYLINE geschrieben. Sämtliche Attributnamen werden ebenfalls als erweiterte Elementdaten der POLYLINE geschrieben.

Der Operator read_contour_xld_dxf kann verwendet werden, um die XLD Konturen zusammen mit ihren Attributen wieder einzulesen.

Andere Anwendungsprogramme, die DXF-Dateien einlesen können, importieren typischerweise nur die Geometrieinformation der Konturen. Die Attribute werden ignoriert.

Beschreibung des Formats der erweiterten Elementdaten

Die erweiterten Elementdaten werden jeweils durch folgende DXF-Gruppe eingeleitet:

1001 HALCON

Attribute werden in folgendem Format als erweiterte Elementdaten zu jedem VERTEX geschrieben:

DXF	Erläuterung
1000	Bedeutung
contour attributes	
1002	Beginn der Werteliste
{	
1070	Anzahl von Attributen (hier: 3)
3	
1040	Wert des ersten Attributs
5.00434303	
1040	Wert des zweiten Attributs
126.8638916	
1040	Wert des dritten Attributs
4.99164152	
1002	Ende der Werteliste
}	

Globale Attribute werden in folgendem Format als erweiterte Elementdaten zu jeder POLYLINE geschrieben:

DXF	Erläuterung
1000	Bedeutung
global contour attributes	
1002	Beginn der Werteliste
{	
1070	Anzahl globaler Attribute (hier: 5)
5	
1040	Wert des ersten globalen Attributs
0.77951831	
1040	Wert des zweiten globalen Attributs
0.62637949	
1040	Wert des dritten globalen Attributs
103.94314575	
1040	Wert des vierten globalen Attributs
0.21434096	
1040	Wert des fünften globalen Attributs
0.21921949	
1002	Ende der Werteliste
}	

Die Namen der Attribute werden in folgendem Format als erweiterte Elementdaten zu jeder POLYLINE geschrieben:

DXF	Erläuterung
1000	Bedeutung
names of contour attributes	
1002	Beginn der Werteliste
{	
1070	Anzahl von Attributnamen (hier: 3)
3	
1000	Name des ersten Attributs
angle	
1000	Name des zweiten Attributs
response	
1000	Name des dritten Attributs
edge_direction	
1002	Ende der Werteliste
}	

Die Namen der globalen Attribute werden in folgendem Format als erweiterte Elementdaten zu jeder POLYLINE geschrieben:

DXF	Erläuterung
1000	Bedeutung
names of global contour attributes	
1002	Beginn der Werteliste
{	
1070	Anzahl von globalen Attributnamen (hier: 5)
5	
1000	Name des ersten globalen Attributs
regr_norm_row	
1000	Name des zweiten globalen Attributs
regr_norm_col	
1000	Name des dritten globalen Attributs
regr_dist	
1000	Name des vierten globalen Attributs
regr_mean_dist	
1000	Name des fünften globalen Attributs
regr_dev_dist	
1002	Ende der Werteliste
}	

Parameter

▷ **Contours** (input_object) . . . . . xld_cont(-array) ⇝ *object*
Zu schreibende XLD-Konturen.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der DXF-Datei.
Dateiendung: `.dxf`

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei geschrieben werden konnte, liefert `write_contour_xld_dxf` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`edges_sub_pix`

Siehe auch

`read_contour_xld_dxf`, `write_polygon_xld_dxf`, `query_contour_attribs_xld`, `query_contour_global_attribs_xld`, `get_contour_attrib_xld`, `get_contour_global_attrib_xld`

Modul

Foundation

write_polygon_xld_arc_info (Polygons : : FileName :)

Schreiben von XLD-Polygonen im ARC/INFO-Generate-Format.

`write_polygon_xld_arc_info` schreibt die XLD-Polygone `Polygons` im ARC/INFO-Generate-Format in die Datei `FileName`. Wenn kein absoluter Pfad in `FileName` angegeben wird, wird die Ausgabedatei im aktuellen Verzeichnis des HALCON-Prozesses geschrieben. Die Polygone müssen zuvor mit

affine_trans_polygon_xld in Weltkoordinaten transformiert worden sein. Die dazu notwendige Transformation kann mit read_world_file aus einem ARC/INFO World File eingelesen werden.

Achtung

Die XLD-Konturen, die möglicherweise von Polygons referenziert werden, werden nicht in die ARC/INFO-Datei abgespeichert, da dies mit dem ARC/INFO-Generate-Format nicht möglich ist. Wenn die Polygone mit read_polygon_xld_arc_info wieder eingelesen werden, ist diese Information verlorengegangen, und deshalb können auch keine Referenzen auf Konturen für die Polygone generiert werden. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen, z.B. split_contours_xld, zu Fehlermeldungen kommen.

Parameter

- ▷ **Polygons** (input_object) ..... xld_poly(-array) ⇝ *object*
  Zu schreibende XLD-Polygone.
- ▷ **FileName** (input_control) ..... filename.write ⇝ *string*
  Name der ARC/INFO-Datei.

Beispiel

```
* Read transformation and image
read_world_file ('image.tfw', WorldTransformation)
read_image (Image, 'image.tif')
* Segment image
* ...
* Write result
affine_trans_polygon_xld (Polygons, PolygonsWorld, WorldTransformation)
write_polygon_xld_arc_info (PolygonsWorld, 'result.gen')
```

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei geschrieben werden konnte, liefert write_polygon_xld_arc_info den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

affine_trans_polygon_xld

Siehe auch

read_world_file, read_polygon_xld_arc_info, write_contour_xld_arc_info

Modul

Foundation

write_polygon_xld_dxf (Polygons : : FileName :)

Schreiben von XLD-Polygonen im DXF-Format.

write_polygon_xld_dxf schreibt die XLD-Polygone Polygons im DXF-Format in die Datei FileName. Wenn kein absoluter Pfad in FileName angegeben wird, wird die Ausgabedatei im aktuellen Verzeichnis des HALCON-Prozesses geschrieben.

write_polygon_xld_dxf schreibt eine Datei im Format der DXF-Version AC1009 (AutoCAD Release 12). Jedes Polygon wird als POLYLINE gespeichert.

Der Operator read_polygon_xld_dxf kann verwendet werden, um die XLD Polygone wieder einzulesen.

Achtung

Die XLD-Konturen, die möglicherweise von Polygons referenziert werden, werden nicht in der DXF-Datei

abgespeichert. Wenn die Polygone mit read_polygon_xld_dxf wieder eingelesen werden, ist diese Information nicht verfügbar, weswegen keine Referenzen auf die Konturen für die Polygone generiert werden können. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen (z.B. split_contours_xld) zu Fehlermeldungen kommen.

Parameter

▷ **Polygons** (input_object) . . . . . xld_poly(-array) ⇝ *object*
Zu schreibende XLD-Polygone.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der DXF-Datei.
Dateiendung: .dxf

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei geschrieben werden konnte, liefert write_polygon_xld_dxf den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_polygons_xld

Siehe auch

read_polygon_xld_dxf, write_contour_xld_dxf

Modul

Foundation

7.7 Zugriff

close_file (: : FileHandle :)

Schließen einer Textdatei.

close_file schließt eine Datei, die mit open_file geöffnet wurde.

Parameter

▷ **FileHandle** (input_control) . . . . . file(-array) ⇝ *handle*
Datei-Handle.

Beispiel

```
open_file ('standard', 'output', FileHandle)
* ....
close_file(FileHandle)
```

Ergebnis

Ist das Datei-Handle korrekt, dann liefert close_file den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_file

Siehe auch

open_file

Modul

Foundation

fnew_line (: : FileHandle :)

Ausgabe eines Zeilenumbruchs und Leeren des Ausgabepuffers in eine Textdatei.

fnew_line gibt in der Ausgabedatei, die durch das Handle FileHandle definiert ist und mit open_file im Textformat geöffnet worden sein muss, einen Zeilenumbruch aus. Dabei schreibt der Aufruf von fnew_line vorher möglicherweise zurückgehaltene Daten aus dem Ausgabepuffer in die Datei (s.a. fwrite_string und set_system(::'flush_file', <boolean-value>:)).

Welche Zeichen als Zeilenumbruch geschrieben werden hängt vom Betriebssystem ab: Unter Windows wird die Zeichenfolge '$\backslash r \backslash n$' (carriage return + line feed) als Standardzeilenumbruch verwendet, unter Linux dagegen nur '$\backslash n$' (line feed).

Parameter

▷ **FileHandle** (input_control) ..... file ⇝ *handle*
Datei-Handle.

Beispiel

```
fwrite_string(FileHandle,'Good Morning')
fnew_line(FileHandle)
```

Ergebnis

fnew_line liefert den Wert TRUE, falls eine Ausgabedatei geöffnet ist und geschrieben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fwrite_string

Siehe auch

fwrite_string

Modul

Foundation

fread_bytes (: : FileHandle, NumberOfBytes : ReadData, IsEOF)

Einlesen von Bytes aus einer Eingabebinärdatei.

fread_bytes liest Bytes aus der Eingabedatei, die durch das Handle FileHandle definiert ist und mit open_file im Binärformat geöffnet worden sein muss.

Die Anzahl der zu lesenden Bytes wird in NumberOfBytes übergeben, und muss größer als *0* sein.

Die gelesenen Bytes werden in ReadData zurückgegeben. IsEOF enthält den Wert *1*, wenn das Dateiende erreicht wurde, ansonsten *0*.

Wenn die Anzahl der zu lesenden Bytes größer ist als die Anzahl der Bytes in der Eingabebinärdatei, werden die gelesenen Bytes bis zum Dateiende in `ReadData` zurückgegeben und der Parameter `IsEOF` wird auf *1* gesetzt.

Wenn keine Bytes gelesen werden können, weil das Ende der Datei erreicht ist, wird in `ReadData` ein leeres Tupel zurückgegeben und der Parameter `IsEOF` auf *1* gesetzt.

Parameter

- ▷ **`FileHandle`** (input_control) ..... file ⇝ *handle*
  Datei-Handle.
- ▷ **`NumberOfBytes`** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu lesenden Bytes.
- ▷ **`ReadData`** (output_control) ..... integer-array ⇝ *integer*
  Gelesenen Bytes.
- ▷ **`IsEOF`** (output_control) ..... integer ⇝ *integer*
  Ende der Datei beim Lesen erreicht.

Beispiel

```
* Read a binary file 5 bytes at a time till EOF is reached.
open_file(Filename,'input_binary',FileHandle)
repeat
  fread_bytes(FileHandle, 5, BytesRead, IsEOF)
until (IsEOF)
close_file (FileHandle)
```

Ergebnis

`fread_bytes` liefert den Wert TRUE, falls die Eingabedatei im Binärformat geöffnet ist und gelesen werden konnte. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_file`

Nachfolger

`close_file`

Alternativen

`fread_char`

Siehe auch

`open_file`, `close_file`, `fwrite_bytes`

Modul

Foundation

`fread_char` (: : FileHandle : Char)

Einlesen eines Zeichens aus einer Textdatei.

`fread_char` liest aus der Eingabedatei, die durch das Handle `FileHandle` definiert ist und mit `open_file` im Textformat geöffnet worden sein muss, ein Zeichen ein.

Das eingelesene Zeichen oder der Steuerstring 'eof' wird im Parameter `Char` zurückgegeben. Dabei wird vom Operator `fread_char` das String-Encoding der Datei berücksichtigt, es sei denn, die Datei wurde mit der Option *'ignore_encoding'* geöffnet (s.a. `open_file`). Das bedeutet, dass bei korrektem Encoding der Operator `fread_char` auch Multi-Byte-Zeichen mit einem Aufruf zurückliefert. Falls notwendig wird das Zeichen anschließend in das aktuelle Encoding der HALCON-Bibliothek umgewandelt (s.a. `set_system(::'filename_encoding',<encoding>:)`).

Wurde die Datei mit der Option *'ignore_encoding'* geöffnet, und wurden noch nicht alle Zeichen der Datei gelesen, enthält `Char` immer genau das nächste Byte aus der Datei ohne weitere Interpretation oder Umcodierung. Das kann z.B. dann sinnvoll sein, wenn eine Datei spezielle benutzerdefinierte Steuerzeichen enthält, die in einer Art Rohdaten-Modus gelesen werden sollen. Wurde das Ende der Datei erreicht und sind keine weiteren Daten verfügbar, liefert `fread_char` die Zeichenkette *'eof'*.

Ergeben in dem angegebenen Encoding die nächsten zu lesenden Bytes kein gültiges Zeichen, wird von `fread_char` ein Low-Level-Fehler ausgegeben. Unabhängig von dem Low-Level-Fehler wird `fread_char` normal beendet und liefert das nächste Byte in `Char` zurück. Darüber hinaus wird ein Low-Level-Fehler erzeugt, wenn das gelesene Zeichen nicht in das aktuelle Encoding der HALCON-Bibliothek umgewandelt werden kann. Dies kann nur vorkommen, wenn die Datei UTF-8-kodiert ist und die HALCON-Bibliothek das lokale Encoding verwendet (s.a. `set_system(::'filename_encoding','locale':)`).

Parameter

- ▷ **`FileHandle`** (input_control) ..... file ⇝ *handle*
  Datei-Handle.
- ▷ **`Char`** (output_control) ..... string ⇝ *string*
  Eingelesenes Zeichen oder Steuerstring ('eof').

Beispiel

```
* Read a text file character by character.
open_file (FileName, 'input', FileHandle)
repeat
    fread_char (FileHandle, Char)
until (Char == 'eof')
close_file (FileHandle)
```

Ergebnis

fread_char liefert den Wert TRUE, falls eine Eingabedatei geöffnet ist. Ansonsten wird eine Fehlerbehandlung durchgeführt. Encodingfehler haben keinen Einfluss auf den Rückgabewert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_file`

Nachfolger

`close_file`

Alternativen

`fread_string`, `read_string`, `fread_line`

Siehe auch

`open_file`, `close_file`, `fread_string`, `fread_line`

Modul

Foundation

`fread_line ( : : FileHandle : OutLine, IsEOF )`

Einlesen von einer Zeile aus einer Textdatei.

`fread_line` liest aus der aktuellen Eingabedatei, die durch das Handle `FileHandle` definiert ist und mit `open_file` im Textformat geöffnet worden sein muss, eine komplette Zeile einschließlich des Zeilenumbruchzeichens ein.

Die eingelesene Zeile wird im Parameter `OutLine` zurückgegeben. Sie beginnt an der aktuellen Dateiposition und endet am Dateiende oder mit dem nächsten Zeilenumbruchzeichen. Nachfolgende Leseoperationen auf der Datei beginnen mit dem ersten Zeichen nach dem Zeilenumbruch, d.h. mit der nächsten Zeile. Falls notwendig, wird die eingelesene Zeile in das aktuell von der HALCON-Bibliothek verwendete Encoding (standardmäßig UTF-8) umgewandelt.

Welche Zeichen als Zeilenumbruch interpretiert werden, hängt vom String-Encoding ab, das beim Öffnen der Datei mit `open_file` angegeben wurde. Die folgenden Zeichen werden unabhängig vom Encoding immer als Zeilenumbruch akzeptiert: *'\n'* (line feed), *'\r'* (carriage return) und *'\f'* (form feed). Unter Windows wird ein Zeilenumbruch normalerweise durch die Zeichenfolge *'\r\n'* (carriage return + line feed) dargestellt. Diese Zeichenfolge wird auf allen Systemen als *ein* Zeilenumbruch interpretiert und im Ausgabestring als *'\n'* zurückgegeben. In UTF-8-kodierten Dateien werden darüber hinaus auch die folgenden Unicode Steuerzeichen als Zeilenende interpretiert: *'U+0085'* (next line), *'U+2028'* (line separator) und *'U+2029'* (paragraph separator).

Wenn das Ende der Datei erreicht wurde, ohne dass ein Zeichen in die Ausgabe übernommen wurde, liefert der Parameter `IsEOF` den Wert *1*, ansonsten *0*.

Stößt der Operator `fread_line` beim Lesen auf Bytes, die im angegebenen Encoding kein gültiges Zeichen ergeben, wird ein Low-Level-Fehler ausgegeben. Unabhängig von dem Low-Level-Fehler wird `fread_line` jedoch normal beendet, wobei der Ausgabestring `OutLine` ungültige Zeichen enthalten kann. Darüber hinaus wird ein Low-Level-Fehler erzeugt, wenn der eingelesene String nicht verlustfrei in das aktuelle Encoding der HALCON-Bibliothek umgewandelt werden kann. Dies kann für eine in der Datei korrekt kodierten Zeichenkette nur vorkommen, wenn die Datei UTF-8-kodiert ist und die HALCON-Bibliothek das lokale Encoding verwendet (s.a. `set_system(::'filename_encoding','locale':)`).

Parameter

- ▷ **`FileHandle`** (input_control) ..... file ⇝ *handle*
  Datei-Handle.
- ▷ **`OutLine`** (output_control) ..... string ⇝ *string*
  Eingelesene Zeile.
- ▷ **`IsEOF`** (output_control) ..... integer ⇝ *integer*
  Ende der Datei erreicht bevor ein Zeichen gelesen werden konnte.

Beispiel

```
do {
  fread_line(FileHandle,&Line,&IsEOF);
} while(IsEOF==0);
```

Ergebnis

`fread_line` liefert den Wert TRUE, falls die Datei geöffnet ist und eine geeignete Zeile eingelesen wird. Ansonsten wird eine Fehlerbehandlung durchgeführt. Encodingfehler haben keinen Einfluss auf den Rückgabewert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_file`

Nachfolger

`close_file`

Alternativen

`fread_char`, `fread_string`

Siehe auch

`open_file`, `close_file`, `fread_char`, `fread_string`

Modul

Foundation

fread_string (: : FileHandle : OutString, IsEOF)

Einlesen einer Zeichenkette aus einer Textdatei.

fread_string liest aus der aktuellen Eingabedatei einen String ein. Die Eingabedatei ist durch das Handle FileHandle definiert und muss mit open_file im Textformat geöffnet worden sein.

Ein String beginnt mit dem ersten Zeichen, das kein Trennzeichen ist. Das sind alle druckbaren Zeichen, aber auch alle Sonderzeichen, die weder Leerzeichen noch Zeilenumbruch sind. Ein String endet vor dem ersten Trennzeichen (Leerzeichen oder Zeilenumbruch), nachdem mindestens ein Zeichen in die Ausgabe übernommen wurde, oder wenn das Dateiende erreicht wurde. Das Trennzeichen, das den String beendet, wird nicht in die Ausgabe übernommen. Der Dateizeiger bleibt jedoch auf dem Zeichen nach dem Trennzeichen stehen, so dass ein nachfolgender Lesebefehl auf der Datei nach dem Trennzeichen weiterlesen würde. Die eingelesene Zeichenkette wird im Parameter OutString zurückgegeben. Falls notwendig, wird die Zeichenkette zuvor in das aktuell von der HALCON-Bibliothek verwendete Encoding (standardmäßig UTF-8) umgewandelt.

Welche Zeichen als Trennzeichen interpretiert werden, hängt von dem String-Encoding ab, das beim Öffnen der Datei mit open_file angegeben wurde. Das normale Leerzeichen *' '* (space) sowie die Standard-ASCII-Trennzeichen *'\t'* (tab), *'\f'* (form feed), *'\n'* (line feed), *'\r'* (carriage return) und *'\v'* (vertical tab) werden unabhängig vom Encoding immer als Trennzeichen akzeptiert.

Abhängig vom Encoding können weitere Trennzeichen hinzukommen. Latin-1 definiert z.B. *'0xA0'* (no-break space) und Shift-JIS enthält wie viele andere asiatische Encodings ein ideographisches Leerzeichen (*'0x8140'*, ideographic space). Neben diesen Zeichen definiert der Unicode-Standard noch weitere Trennzeichen in insgesamt drei Kategorien: Leerzeichen (*Zs*), Zeilentrenner (*Zl*) und Absatztrenner (*Zp*).

Wenn das Ende der Datei erreicht wurde, bevor mindestens ein Zeichen in die Ausgabe übernommen wurde, liefert der Parameter IsEOF den Wert *1*, ansonsten *0*.

Stößt der Operator fread_string beim Lesen auf Bytes, die im angegebenen Encoding kein gültiges Zeichen ergeben, wird ein Low-Level-Fehler ausgegeben. Unabhängig von dem Low-Level-Fehler wird fread_string jedoch normal beendet, wobei der Ausgabestring OutString ungültige Zeichen enthalten kann. Darüber hinaus wird ein Low-Level-Fehler erzeugt, wenn der eingelesene String nicht verlustfrei in das aktuelle Encoding der HALCON-Bibliothek umgewandelt werden kann. Dies kann für eine in der Datei korrekt kodierten Zeichenkette nur vorkommen, wenn die Datei UTF-8-kodiert ist und die HALCON-Bibliothek das lokale Encoding verwendet (s.a. set_system(::'filename_encoding','locale':)).

Parameter

- ▷ **FileHandle** (input_control) ..... file ⇝ *handle*
  Datei-Handle.
- ▷ **OutString** (output_control) ..... string ⇝ *string*
  Eingelesene Zeichenkette.
- ▷ **IsEOF** (output_control) ..... integer ⇝ *integer*
  Ende der Datei erreicht bevor mindestens ein Zeichen in die Ausgabe übernommen wurde.

Beispiel

```
fwrite_string(FileHandle,'Please enter text and return: ..')
fread_string(FileHandle,String,IsEOF)
fwrite_string(FileHandle,['here it is again: ',String])
fnew_line(FileHandle)
```

Ergebnis

fread_string liefert den Wert TRUE, falls eine Datei geöffnet ist und ein geeigneter String eingelesen wird. Ansonsten wird eine Fehlerbehandlung durchgeführt. Encodingfehler haben keinen Einfluss auf den Rückgabewert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_file

Nachfolger

close_file

Alternativen

fread_char, read_string, fread_line

Siehe auch

open_file, close_file, fread_char, fread_line

Modul

Foundation

fwrite_bytes (: : FileHandle, DataToWrite : NumberOfBytesWritten)

Schreiben von Bytes in eine Binärdatei.

fwrite_bytes schreibt Bytes in die Ausgabedatei, die durch das Handle FileHandle definiert ist und mit dem Operator open_file im Binärformat geöffnet worden sein muss.

Die Daten, die in die Datei geschrieben werden sollen, werden als DataToWrite übergeben.

Die Anzahl der Bytes, die in die Datei geschrieben wurden, wird in NumberOfBytesWritten zurückgegeben.

Parameter

- ▷ **FileHandle** (input_control) ..... file ⇝ *handle*
  Datei-Handle.
- ▷ **DataToWrite** (input_control) ..... integer-array ⇝ *integer*
  Zu schreibende Daten, die in die Datei geschrieben werden sollen.
- ▷ **NumberOfBytesWritten** (output_control) ..... integer ⇝ *integer*
  Anzahl der Bytes, die in die Ausgabedatei geschrieben wurden.

Beispiel

```
* Write a binary file byte by byte.
open_file (Filename, 'append_binary', FileHandle)
fwrite_bytes(FileHandle, [0x97, 99, 102], BytesWritten)
close_file (FileHandle)
```

Ergebnis

fwrite_bytes liefert den Wert TRUE, falls die Ausgabedatei im Binärformat geöffnet ist und geschrieben werden konnte. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_file

Nachfolger

close_file

Alternativen

fwrite_string

Siehe auch

open_file, close_file, fread_bytes

Modul

Foundation

fwrite_string (: : FileHandle, String :)

Ausgabe von Zeichenketten und Zahlen in eine Textdatei.

fwrite_string schreibt Zeichenketten und Zahlen in die Ausgabedatei, die durch das Handle FileHandle definiert ist und mit dem Operator open_file im Textformat geöffnet worden sein muss.

Die Werte, die in der Datei ausgegeben werden sollen, werden mit dem Parameter String übergeben. Der Parameter String akzeptiert auch Tupel, in denen beliebige Zahlen und Zeichenketten gemischt vorkommen können. Alle Elemente eines Tupels werden direkt hintereinander geschrieben, ohne Leer- oder andere Trennzeichen dazwischen. Zahlen werden vor der Ausgabe in eine Zeichenkette umgewandelt.

Der Operator fwrite_string erzeugt einen Low-Level-Fehler wenn der String nicht verlustfrei in das angegebene Datei-Encoding umgewandelt werden kann. Dies kann vorkommen, wenn das Datei-Encoding *'locale_encoding'* ist und das aktuelle Encoding der HALCON-Bibliothek *'utf8'* (s.a. set_system(::'filename_encoding','utf8':)).

Mit dem Aufruf set_system(::'flush_file', <Wahrheitswert>:) wird festgelegt, ob die Ausgabezeichen direkt oder verzögert in die Ausgabedatei geschrieben werden sollen. Ist *'flush_file'* auf *'false'* gesetzt, dann erfolgt die Ausgabe (insbesondere bei Bildschirmausgabe) erst nach dem Aufruf von fnew_line.

Parameter

- ▷ **FileHandle** (input_control) ..... file ⇝ *handle*
  Datei-Handle.
- ▷ **String** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Werte, die in der Datei ausgegeben werden sollen.
  Defaultwert: 'hallo'

Beispiel

```
fwrite_string(FileHandle,['text with numbers:',5,' and ',1.0])
* results in the following output:
* 'text with numbers:5 and 1.00000'
```

Ergebnis

fwrite_string liefert den Wert TRUE, falls der Schreibvorgang erfolgreich abgeschlossen wird. Ansonsten wird eine Fehlerbehandlung durchgeführt. Encodingfehler haben keinen Einfluss auf den Rückgabewert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_file

Nachfolger

close_file

Alternativen

write_string

Siehe auch

open_file, close_file, set_system

Modul

Foundation

open_file (: : FileName, FileType : FileHandle)

Öffnen einer Datei im Text- oder Binärformat.

open_file öffnet eine Datei im Text- oder im Binärformat. Der Name der Datei wird im Parameter FileName übergeben. Der Operator gibt einen Handle auf die Datei in FileHandle zurück. Mit dem Parameter FileType wird der Typ der Datei bestimmt. Für Textdateien kann zusammen mit dem Typ auch das Encoding der Datei spezifiziert werden, dabei wird neben UTF-8 nur das jeweils lokale 8-Bit-Encoding unterstützt.

Die folgenden Einstellungen für FileType werden unterstützt:

'input': Öffnet eine bereits vorhandene Eingabedatei im Textformat lesend.

'output': Öffnet eine neue Ausgabedatei im Textformat schreibend.

'append': Öffnet eine bereits vorhandene Ausgabedatei im Textformat am Ende der Datei schreibend.

'input_binary': Öffnet eine bereits vorhandene Eingabedatei im Binärformat lesend.

'output_binary': Öffnet eine neue Ausgabedatei im Binärformat schreibend.

'append_binary': Öffnet eine bereits vorhandene Ausgabedatei im Binärformat am Ende der Datei schreibend.

Das an FileType übergebene Tupel kann bei Textdateien als zweites Element einen der folgenden Encoding-Einstellungen enthalten:

'utf8_encoding': Zeichenketten werden in der Datei mit UTF-8 kodiert. Das ist der Standard, d.h. beim Öffnen von Dateien, die UTF-8 kodiert sind oder die nur reine ASCII-Zeichen enthalten, braucht das Encoding nicht angegeben werden.

'locale_encoding': Zeichenketten werden in der Datei mit dem lokalen 8-Bit-Encoding kodiert. Dieses hängt vom verwendeten Betriebssystem und der eingestellten Lokalisierung ab. Mögliche Encodings sind hier z.B. unter Windows die Codepages 1252 (Microsoft-Variante von Latin-1) oder 932 (Shift-JIS) und unter Linux die Locales en_US.utf8, de_DE.iso885915 oder ja_JP.sjis.

'ignore_encoding': Das Encoding der Strings, die aus der Datei gelesen oder in sie geschrieben werden, wird nicht berücksichtigt. In diesem Modus werden Multi-Byte-Zeichen weder behandelt noch interpretiert, was bedeutet, dass der Operator fread_char in diesem Modus immer genau ein Byte zurückliefert, und die Operatoren fread_line und fread_string keine lokalisierungsabhängigen Trennzeichen berücksichtigen. Schließlich werden die Strings auch nicht in das aktuelle Encoding der HALCON-Bibliothek umkodiert.

Hinweis: Die Strings werden dennoch in das Encoding der HALCON C/C++ Schnittstelle umcodiert, wenn sich das Encoding der Schnittstelle von dem der HALCON-Bibliothek unterscheidet. Für HDevelop Skripte ist das Encoding immer UTF-8. Um eine Umcodierung in HDevelop zu vermeiden, sollte das Encoding der HALCON-Bibliothek UTF-8 sein.

Für Aus- und Eingaben auf dem Terminal sind die Dateinamen *'standard'* (*'input'* und *'output'*) und *'error'* (nur *'output'*) reserviert.

Parameter

▷ **FileName** (input_control) ..... filename ⇝ *string*
Name der zu öffnenden Datei.
Defaultwert: 'standard'
Wertevorschläge: FileName ∈ {'standard', 'error', '/tmp/dat.dat'}

▷ **FileType** (input_control) ..... string(-array) ⇝ *string*
Art des Dateizugriffs und ggf. das String-Encoding.
Defaultwert: 'output'
Werteliste: FileType ∈ {'input', 'output', 'append', 'input_binary', 'output_binary', 'append_binary', 'utf8_encoding', 'locale_encoding', 'ignore_encoding'}

▷ **FileHandle** (output_control) ..... file ⇝ *handle*
Datei-Handle.

Beispiel

```
* Creating an output text file with the name '/tmp/log.txt' and writing
* of a string:
  open_file('/tmp/log.txt','output',FileHandle)
  fwrite_string(FileHandle,'these are the first and last lines')
  fnew_line(FileHandle)
  close_file(FileHandle)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `open_file` den Wert TRUE. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`fwrite_string`, `fread_char`, `fread_string`, `fread_line`, `fread_serialized_item`, `fwrite_serialized_item`, `fread_bytes`, `fwrite_bytes`, `close_file`

Siehe auch

`close_file`

Modul

Foundation

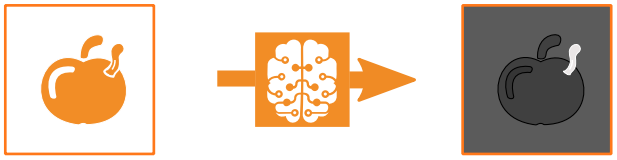
Kapitel 8

Deep Learning

Einleitung

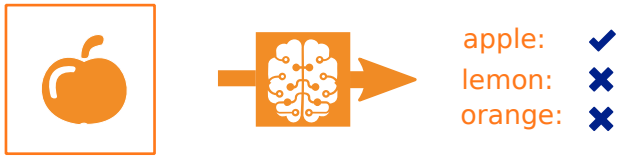
Der Begriff Deep Learning (DL) bezieht sich auf eine Familie von Verfahren des maschinellen Lernens. In HALCON sind folgende Methoden implementiert:

Anomalieerkennung: Jeder Pixel des Eingabebildes erhält einen Score, der angibt, wie wahrscheinlich der Pixel ein unbekanntes Merkmal, eine Anomalie, zeigt. Für weitere Informationen wird auf das Kapitel Deep Learning / Anomalieerkennung verwiesen.



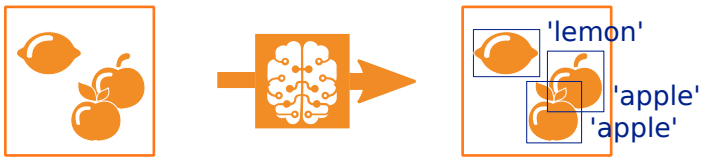
Ein mögliches Beispiel der Anomalieerkennung: Jeder Pixel des Eingabebildes erhält einen Score.

Klassifikation: Ein Bild wird einer Klasse aus einer gegebenen Menge an Klassen zugeordnet. Für weitere Informationen wird auf das Kapitel Deep Learning / Klassifikation verwiesen.



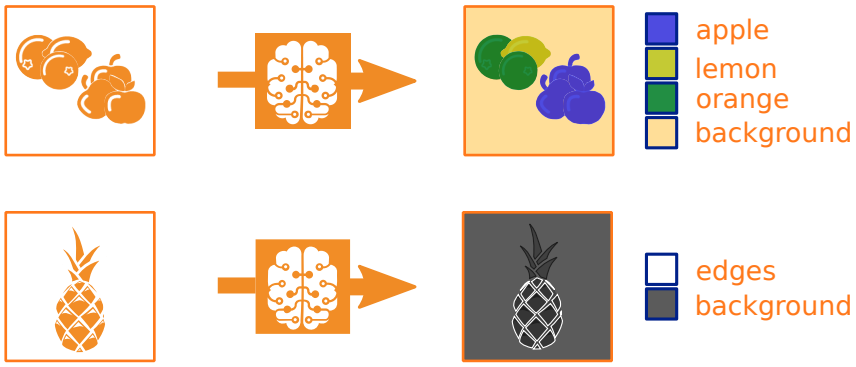
Ein mögliches Beispiel der Klassifikation: Dem Bild wird eine Klasse zugeordnet.

Objektdetektion: Objekte der gegebenen Klassen werden erkannt und innerhalb des Bildes lokalisiert. Für weitere Informationen wird auf das Kapitel Deep Learning / Objektdetektion verwiesen.



Ein mögliches Beispiel der Objektdetektion: Innerhalb des Bildes werden drei Instanzen gefunden und jeweils einer Klasse zugeordnet.

Semantische Segmentierung: Jedem Pixel eines Bildes wird eine Klasse zugeordnet. Für weitere Informationen wird auf das Kapitel Deep Learning / Semantische Segmentierung verwiesen.



Oben: Ein mögliches Beispiel der Semantischen Segmentierung: Jedem Pixel des Eingabebildes wird eine Klasse zugeordnet.
Unten: Ein mögliches Beispiel der Kantenextraktion: Ein Spezialfall der Semantischen Segmentierung, bei dem jedem Pixel des Eingabebildes eine der Klassen 'edge' und 'background' zugeordnet wird.

All diese Deep Learning Methoden verwenden für die Zuordnung ein Netzwerk. In HALCON sind sie im Rahmen des generellen DL Modells implementiert, siehe Deep Learning / Modell. Dieses Netzwerk wird nur mit Hilfe von Ein- und Ausgabe trainiert, was auch als End-to-End Lernen bezeichnet wird. Im Wesentlichen wird durch die Bilder und die Information, was darauf ersichtlich ist, durch den Trainingsalgorithmus das Modell so angepasst, dass dieses die verschiedenen Klassen unterscheiden kann und gegebenenfalls auch die Objekte findet. Für den Anwender hat dies den Vorteil, dass er nicht händisch Merkmale angeben muss. Stattdessen müssen geeignete Daten ausgewählt und gesammelt werden.

Systemvoraussetzungen

Für Deep Learning gelten zusätzliche Voraussetzungen, siehe den HALCON „`Installation Guide`", Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods".

Um den Trainingsprozess zu beschleunigen, empfehlen wir eine schnelle Festplatte. Solid-State-Drives (SSD) sind konventionellen Festplatten (HDD) vorzuziehen.

Genereller Ablauf

Die genannten DL Methoden unterscheiden sich sowohl bezüglich was sie tun als auch wie die Daten benötigt werden. Deshalb ist es wichtig, erst zu wissen, welche Methoden für die spezifische Aufgabe am geeignetsten ist. Ist dies klar, müssen genügend Daten gesammelt werden, sprich Bilder und die von der Methode dazu benötigten Informationen. Danach gilt für all diese Methoden grob der folgende Ablauf:

Vorbereiten des Netzwerks und der Daten Das Netzwerk muss für die spezifische Aufgabe vorbereitet werden. Ebenso müssen die Daten an die Anforderungen des Netzwerks angepasst werden.

- Ein Netzwerk besorgen: Einlesen oder Erzeugen eines Netzwerks.
- Es ist notwendig zu wissen, welches Problem das Netzwerk lösen soll, d.h., welche Klassen zu unterscheiden sind und wie mögliche Vertreter aussehen. Das wird durch den Datensatz repräsentiert, das heißt die Bilder mit den zugehörigen Ground Truth Informationen.
- Das Netzwerk stellt mehrere Anforderungen an die Bilder, wie z.B. die Bilddimensionen und der Grauwertbereich, ... Damit die Bilder vom Netzwerk verarbeitet werden können, müssen sie entsprechend vorverarbeitet werden.
- Als nächstes wird empfohlen die Daten in drei getrennte Datensätze zu teilen, die für Training, Validierung und Testen verwendet werden.

Training des Netzwerks und Auswertung des Trainingsfortschritts Sobald das Netzwerk und der Datensatz vorbereitet sind, kann das Netzwerk für die spezifische Aufgabe trainiert werden.

- Die Hyperparameter entsprechend der Aufgabe und System setzen.
- Eventuell die Art der Datenanreicherung bestimmen.
- Das Training starten und das Netzwerk evaluieren.

Anwenden und Evaluieren des finalen Netzwerks Nun ist das Netzwerk für die spezifische Anwendung trainiert und bereit zur Anwendung. Doch davor sollte noch evaluiert werden, wie gut es sich auf dem Test-Datensatz verhält.

Inferenzphase Ist das Netzwerk trainiert und seine Leistung zufriedenstellend, kann es für die Inferenz neuer Bilder verwendet werden. Dafür müssen die Bilder entsprechend den Anforderungen des Netzwerks vorverarbeitet werden (d.h., auf dieselbe Art und Weise wie dies beim Training geschah).

Daten

Der Begriff 'Daten' wird innerhalb DL für Bilder und die Information, was darauf ist, verwendet. Letztere muss so gegeben sein, damit das Netzwerk sie verarbeiten kann. Es verwundert nicht, dass die verschiedenen Methoden ihre eigenen Anforderungen haben, was für Informationen gegeben sein müssen und wie diese vorzuliegen haben. Für die spezifischen Anforderungen wird auf das jeweilige Kapitel verwiesen.

Das Netzwerk stellt auch Anforderungen an die Bilder betreffend der Bilddimensionen, dem Grauwertbereich und dem Bildtyp. Diese Voraussetzungen hängen vom verwendeten Netzwerk ab und können mit `get_dl_model_param` abgefragt werden. Zusätzlich gibt es je nach Methode auch Anforderungen an die dazugehörigen Information, wie z.B. die umschließenden Rechtecke. Um diese Anforderungen zu erfüllen, müssen die Daten vorverarbeitet werden, was komfortabel mit der entsprechenden Prozedur `preprocess_dl_samples` gemacht werden kann.

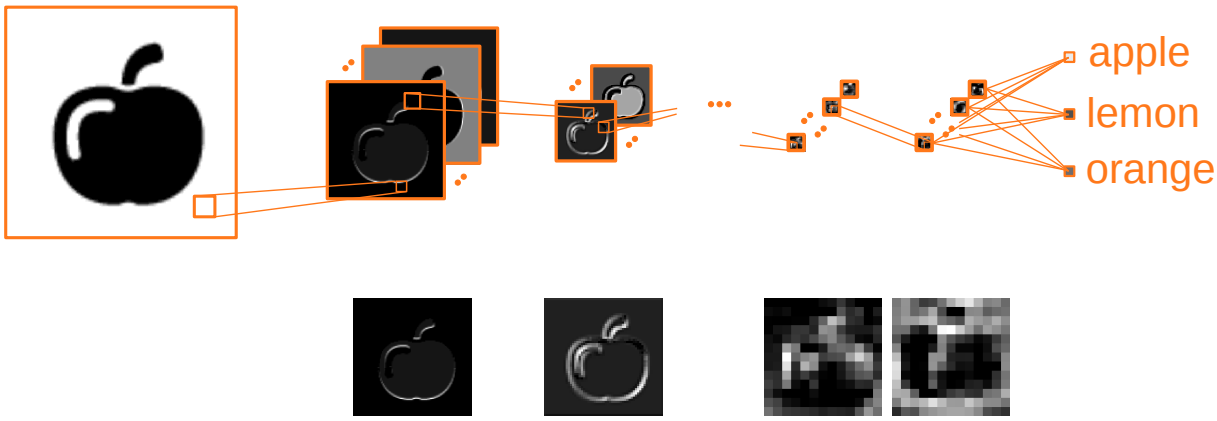
Wenn das Netzwerk trainiert wird, adaptiert es sich an seine neue Aufgabe. Aber zu gewissen Zeitpunkten soll die Leistung des Netzwerks überprüft und zu einem späteren Zeitpunkt auch getestet werden. Dafür wird der Datensatz in drei Teilmengen aufgespalten, die unabhängig und gleichverteilt sein sollten. Das bedeutet, dass die Teilmengen auf keine Art und Weise voneinander abhängen sollten und jede Klasse die gleiche Verteilung von Bildern enthalten sollte. Diese Aufteilung kann bequem mit der Prozedur `split_dl_dataset` vorgenommen werden. Die größte Teilmenge wird für das Nachtrainieren verwendet. In HALCON wird dieses Set Trainings-Datensatz bezeichnet. Zu bestimmten Zeitpunkten wird die Leistung des Netzwerks überprüft, um festzustellen ob es sich lohnt, die Optimierung des Netzwerks fortzusetzen. Für diese Validierung wird die zweite Teilmenge des Datensatzes verwendet, der Validierungs-Datensatz. Auch wenn diese Teilmenge disjunkt zur ersten Menge ist, hat sie Einfluss auf die Optimierung des Netzwerks. Deshalb wird für die Vorhersage der Leistung des Netzwerks in der Applikation die dritte Teilmenge verwendet, dem Test-Datensatz. Für eine repräsentative Validierung des Netzwerks sollten die zweite und die dritte Teilmenge statistisch relevante Daten enthalten, was eine untere Schranke für die benötigte Menge an Daten darstellt.

Es ist zudem zu beachten, dass die Bilder repräsentativ sein sollten, das heißt denen der späteren Applikation entsprechen und nicht 'perfekten' Bildern. Ansonsten kann das Netzwerk Probleme auf nicht-'perfekten' Bildern haben.

Das Netzwerk und der Trainingsprozess

In DL erfolgen die Zuordnungen, indem das Eingabebild von einem Netzwerk verarbeitet wird. Die Ausgabe des gesamten Netzwerkes besteht aus einer Anzahl Vorhersagen. Solche Vorhersagen sind, z.B. für die Klassifikation, die Konfidenzwerte für jede Klasse, welche die Klassenzugehörigkeit des Bildes darstellen.

Die spezifisches Netzwerke variieren, insbesondere von Methode zu Methode. Einige Methoden wie z.B. die Objekterkennung verwenden ein Subnetzwerk, um Merkmalskarten zu erzeugen (siehe die Erklärungen unten und in Deep Learning / Objektdetektion). Hier wird ein grundlegendes Neuronales Netzwerk erklärt. Ein solches Netzwerk besteht aus einer Anzahl an Layer oder Filter, welche in einer bestimmten Art angeordnet und miteinander verbunden sind. Allgemein ist ein Layer ein Baustein, der besondere Aufgaben durchführt. Er kann wie eine Box gesehen werden, welche Eingaben erhält, diese anhand einer Funktion transformiert und die Ausgabe an den nächsten Layer weiter gibt. Dabei sind verschiedene Funktionen für die unterschiedlichen Typen von Layern möglich, einige Beispiele sind im „`Solution Guide on Classification`" gegeben. Viele der Layer haben Gewichte. Diese Filter-Gewichte sind die freien Parameter, welche beim Training verändert werden. Ausgabe der meisten Layer sind Merkmalskarten. Dabei hängt die Anzahl der Merkmalskarten (die Tiefe der Layer-Ausgabe) sowie die Größe der Merkmalskarte (Höhe und Breite) vom spezifischen Layer ab.

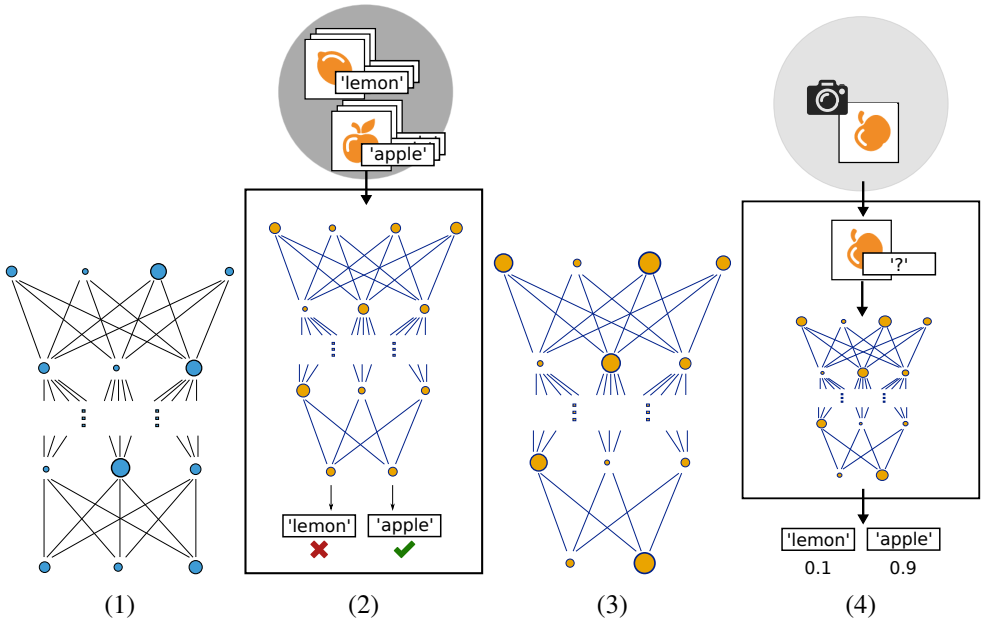


Schema eines möglichen Klassifikator-Ausschnitts. Unten sind zu den Layern gehörenden Merkmalskarten gezeigt, wobei diese auf eine einheitliche Größe skaliert wurden.

Um ein Netzwerk für eine spezifische Aufgabe zu trainieren wird eine Zielfunktion verwendet. Abhängig von der Aufgabe gibt es verschiedene Zielfunktionen, doch funktionieren sie alle nach dem gleichen Prinzip. Die Zielfunktion vergleicht die Vorhersage des Netzwerks mit der gegebenen Information, was es im Bild finden soll (und gegebenenfalls wo) und bestraft Abweichungen. Die Zielfunktion wird minimiert durch entsprechende Änderung der Filter-Gewichte. Um also das Netzwerk für eine spezielle Aufgabe zu trainieren, wird die Minimierung der Zielfunktion (eine Funktion des Fehlers) angestrebt, in der Hoffnung, damit auch das Leistungsmaß des Netzwerkes zu verbessern. In der Praxis wird diese Optimierung durch die Berechnung des Gradienten und der entsprechenden Anpassung der Parameter (Filter-Gewichte) bei mehrfacher Iteration über den Trainings-Datensatz erreicht.

Es gibt weitere Parameter, welche das Training beeinflussen aber nicht direkt während des regulären Trainings gelernt werden. Diese Parameter werden vor dem Training gesetzt. Um sie von den anderen Parametern abzuheben, werden sie als Hyperparameter bezeichnet. Siehe den obigen Abschnitt „Setzen der Trainingsparameter“.

Alle Gewichte von Grund auf zu optimieren benötigt eine große Menge an Ressourcen. Deshalb wird von der folgenden Erkenntnis Gewinn gezogen. Die ersten, tiefen Layer detektieren einfache Merkmale wie Kanten oder Kurven. Die Merkmalskarten der späteren Layer sind kleiner, doch zeigen sie komplexere Merkmale. Für ein großes Netzwerk sind die Merkmale der tiefen Layer generell genug, damit sich deren Gewichte zwischen verschiedenen Aufgaben nur gering ändern. Dies führt zu einer Technik, die Transferlernen genannt wird: Man nimmt ein bereits trainiertes Netzwerk inklusive seiner Gewichte und trainiert es für eine neue Aufgabe, wobei man von passenden Gewichten der tiefer liegenden Layer profitiert. Als Resultat werden viel weniger Ressourcen benötigt. Während generell gesagt ein Netzwerk mit größerem Datensatz zuverlässiger wird, hängt die Menge an notwendiger Daten auch mit der Komplexität der Aufgabe zusammen. Ein Schema für den Ablauf mit Transferlernen ist unten mit Hilfe der Klassifikation gezeigt.



Elementares Schema des Transferlernens anhand der Klassifikation. (1) Ein vortrainiertes Netzwerk wird eingelesen. (2) Trainingsphase, das Netzwerk wird auf dem Trainings-Datensatz trainiert. (3) Das trainierte Modell mit neuen Fähigkeiten. (4) Inferenzphase, das trainierte Netzwerk inferiert auf neuen Bildern.

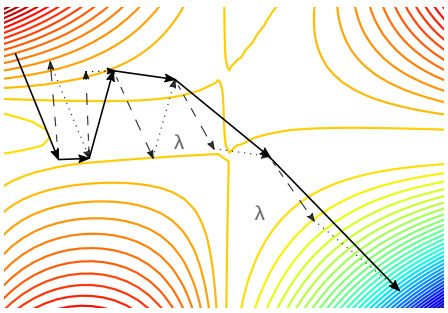
Setzen der Trainingsparameter

Die verschiedenen DL Methoden sind für unterschiedliche Aufgaben konzipiert und unterscheiden sich in der Art und Weise, wie sie zusammengesetzt sind. Allen ist gemeinsam, dass das Training ein Minimierungsproblem darstellt. Beim Trainieren des Netzwerkes oder Subnetzwerkes wird eine geeignete Zielfunktion minimiert, siehe dazu den obigen Abschnitt „Das Netzwerk und der Trainingsprozess“. Um dies zu tun, gibt es bestimmte Parameter, die vor dem Training gesetzt werden müssen und während des Trainings nicht optimiert werden. Diese Parameter werden als Hyperparameter bezeichnet. Für ein DL Modell kann über eine Veränderungsstrategie vorgegeben werden, wann und wie diese Hyperparameter während des Trainings geändert werden sollen. In diesem Abschnitt wird der Grundgedanke verschiedener Hyperparameter erklärt. Es ist zu beachten, dass die jeweiligen Methoden weitere Hyperparameter haben können, weitere Informationen dazu finden sich im jeweiligen Kapitel.

Wie zuvor erwähnt, vergleicht die Zielfunktion die Vorhersage des Netzwerks mit den Informationen zum Bildinhalt und bestraft Abweichungen. Das Netzwerk zu trainieren bedeutet, die Filter-Gewichte so zu ändern, dass die Zielfunktion weniger bestraft, d.h., die Zielfunktion optimiert wird. Um dies zu tun wird eine Menge Daten des Trainings-Datensatzes genommen. Für diese Teilmenge wird der Gradient der Zielfunktion berechnet und das Netzwerk angepasst, indem die Filter-Gewichte geändert werden. Dies wird mit der nächsten Teilmenge an Daten wiederholt bis der ganze Trainings-Datensatz verarbeitet wurde. Eine solche Teilmenge an Daten wird als Batch oder Minibatch bezeichnet. Die Batchgröße (*'batch_size'*) bestimmt die Anzahl der Daten, die einen Batch bilden und deshalb simultan verarbeitet werden.

Eine gesamte Iteration über den vollständigen Trainings-Datensatz wird Epoche genannt. Es ist vorteilhaft, mehrmals über den Trainings-Datensatz zu iterieren. Die Anzahl Iterationen wird dabei über die `'Epochen'` definiert. Das bedeutet, `'Epochen'` bestimmt, wie oft der Trainingsalgorithmus über den Trainings-Datensatz iteriert.

Gewisse Modelle (z.B. Anomalieerkennung) nutzen den gesamten Datensatz auf einmal. Andere Modellen verarbeiten den Datensatz batchweise, wobei der stochastische Gradientenabsenkungsalgorithmus SGD verwendet wird. Dies bringt weitere Parameter mit sich, welche im Folgenden erläutert werden. Nach jeder Berechnung des Zielfunktionsgradienten werden die Filter-Gewichte aktualisiert. Es gibt zwei Hyperparameter, die für die Anpassung der Filter-Gewichte wichtig sind: Die Lernrate *'learning_rate'* λ, die den Einfluss des Gradienten auf die Anpassung der Gewichte beschreibt und das Momentum (*'momentum'* μ, im Intervall $[0, 1)$), das den Einfluss früherer Anpassungen festlegt. Mehr Informationen dazu befinden sich in der Dokumentation von `train_dl_model_batch`. In einfachen Worten gesagt: Bei der Anpassung der Gewichte erinnert man sich an den Schritt, den man für die letzte Anpassung gemacht hat. Jetzt wird ein Schritt in die Richtung des Gradienten gemacht, wobei die Länge abhängig ist von der Lernrate. Zusätzlich wird der Schritt der letzten Anpassung wiederholt, aber diesmal nur mit einer Länge entsprechend dem Momentum $\mu = 0$. Dies wird in der folgenden Grafik illustriert. Eine zu hohe Lernrate kann dazu führen, dass der Algorithmus divergiert, eine zu kleine Lernrate führt zu unnötig vielen Schritten. Daher startet man üblicherweise mit einer höheren Lernrate und reduziert sie während dem Training. Mit einem Momentum $\mu = 0$ hat das Momentum Verfahren keinen Einfluss, so dass nur der Gradient den Anpassungsvektor bestimmt.



Skizze von *'learning_rate'* und *'momentum'* während eines Lernschrittes. Der Gradientenschritt: Die Lernrate λ multipliziert mit dem Gradienten g (λg - gestrichelte Linie). Der Momentumschritt: Das Momentum μ multipliziert mit dem vorherigen Anpassungsvektor v (μv - gepunktete Linie). Zusammen ergeben sie den eigentlichen Lernschritt: Der Anpassungsvektor v (v - durchgezogene Linie).

Regularisierung kann verwendet werden, um Überanpassung an die Trainingsdaten zu reduzieren (siehe auch den Abschnitt „Risiko von Überanpassung und Unteranpassung“ weiter unten). Mit dieser Technik wird der Zielfunktion ein zusätzlicher Term hinzugefügt. Eine mögliche Regularisierung ist die Gewichts-Dämpfung, siehe `train_dl_model_batch`. Einfach gesagt begünstigt die Regularisierung einfachere Modelle, die weniger anfällig sind, Rauschen in den Daten zu lernen. Dadurch generalisieren diese Modelle in der Regel besser. Regularisierung kann über den Hyperparameter `weight_prior` gesetzt werden. Die Wahl des Wertes ist ein Kompromiss zwischen der Fähigkeit des Modells zur Generalisierung, Überanpassung und Unteranpassung. Ist der Wert für *'weight_prior'* zu klein, kann Überanpassung auftreten, ist er zu groß, kann das Modell seine Fähigkeit, sich an die Daten anzupassen, verlieren, da alle Gewichte effektiv Null sind.

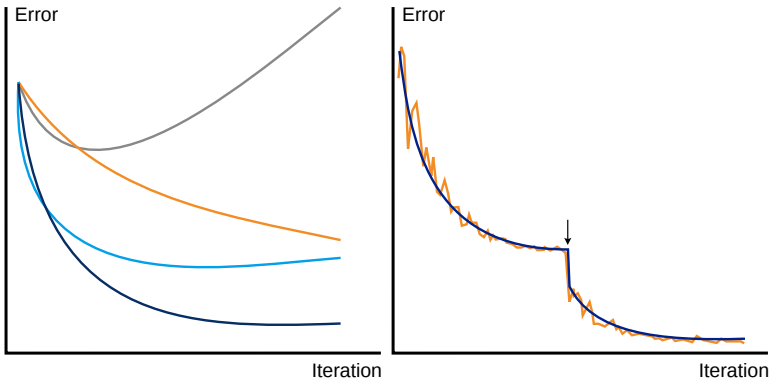
Mit den Daten und den Hyperparametern gibt es viele Einflussfaktoren auf das Resultat des Trainingsalgorithmus. Im Allgemeinen helfen mehr Daten, die Leistung des Netzwerks zu verbessern. Ob mehr Daten immer eine gute Lösung für Verbesserungen sind, hängt davon ab, wie einfach es ist, diese Daten zu sammeln. Üblicherweise hat eine kleine zusätzliche Menge an Daten kaum merklichen Einfluss auf die Gesamtleistung.

Das Training beaufsichtigen

Die verschiedenen DL Methoden geben verschiedene Resultate zurück. Entsprechend werden auch unterschiedliche Maße verwendet, um zu bestimmen 'wie gut' die Leistung eines Netzwerks ist. Dennoch gibt es gemeinsame Verhaltensmuster und mögliche Tücken während des Trainings, welche hier beschrieben werden.

Evaluierung während des Trainings Bei der Evaluierung der Leistung des Netzwerks muss beachtet werden, dass es sich nicht um ein reines Optimierungsproblem handelt (siehe die obigen Abschnitte „Das Netzwerk und der Trainingsprozess“ und „Setzen der Trainingsparameter“).

Um den Trainingsfortschritt zu beobachten ist es meistens nützlich, das Leistungsmaß zu visualisieren, z.B.den Fehler für die Elemente eines Batches im Falle der Klassifikation. Da sich die Elemente unterscheiden, kann die Schwierigkeit der Zuordnung variieren. Deshalb kann es sein, dass das Netzwerk bei den Elementen eines Batches besser oder schlechter abschneidet als bei den Elementen eines anderen Batches. Es ist also normal, dass sich das Leistungsmaß nicht glatt über die Iterationen verändert. Insgesamt sollte er aber sinken. Die Anpassung der Hyperparameter Lernrate *'learning_rate'* und Momentum *'momentum'* kann helfen, das Maß zu verbessern. Die folgende Grafik zeigt ein mögliches Beispiel.

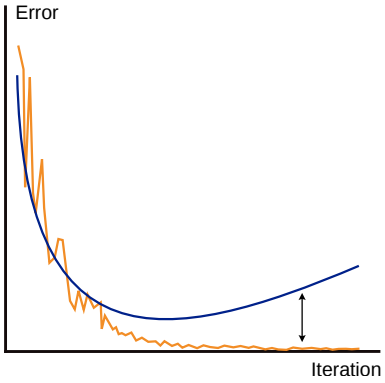


Schema eines Leistungsmaß während des Trainings, hier exemplarisch für die Klassifikation. (1) Generelle Tendenzen für mögliche Folgen verschiedener Werte für *'learning_rate'*, dunkelblau: angemessene Lernrate, grau: sehr hohe Lernrate, hellblau: hohe Lernrate, orange: tiefe Lernrate. (2) Ideal Fall eines Lernraten-Programms zum Reduzieren der Lernrate nach einer gewissen Anzahl Iterationen. In orange: Trainingsfehler, dunkelblau: Validierungsfehler. Der Pfeil markiert die Iteration, bei welcher die Lernrate angepasst, d.h., verringert wurde.

Risiko von Unteranpassung und Überanpassung Unteranpassung tritt auf, wenn das Modell die Komplexität der Aufgabe nicht wiedergeben kann. Das spiegelt sich direkt im Leistungsmaß wieder, welches in diesem Fall hoch bleibt.

Überanpassung tritt dann auf, wenn das Modell anfängt, sich die Trainingsdaten zu merken, statt generelle Regeln zur Aufgabe zu lernen. In diesem Fall bleibt das Leistungsmaß gleich oder verbessert sich, während sich das Leistungsmaß auf dem Validierungsdatensatz verschlechtert. In diesem Fall kann Regularisierung

helfen. Siehe die Erklärung zum Hyperparameter `weight_prior` im obigen Abschnitt „Setzen der Trainingsparameter“. Es ist zu beachten, dass ein ähnlicher Effekt auftritt, wenn die Kapazität des Models im Verhältnis zum Trainings-Datensatz zu hoch ist.



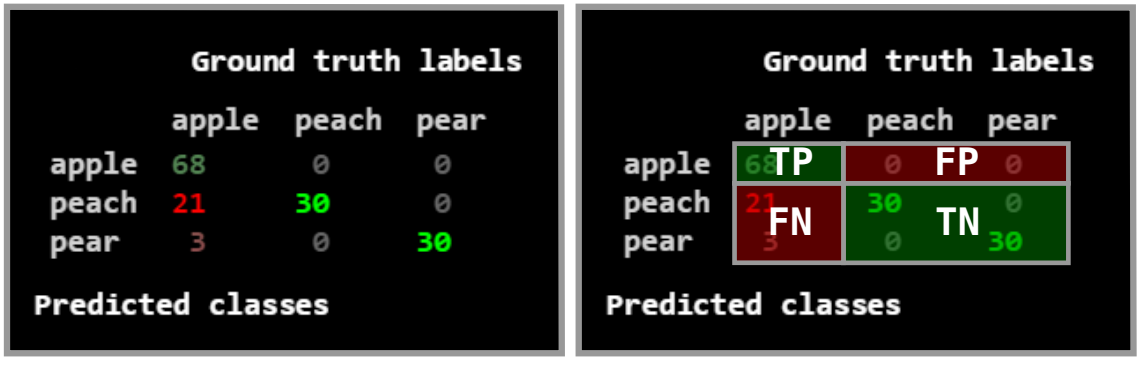
Skizze von möglicher Überanpassung, sichtbar durch die Generalisierungslücke (durch den Pfeil markiert). Der Fehler aus der Klassifikation dient als Beispiel für ein Leistungsmaß.

Konfusionsmatrix Für jede Instanz inferiert das Netzwerk eine top-Vorhersage, die Klasse für welche das Netzwerk die größte Affinität schlussfolgert. Ist die Ground Truth Klasse und damit die wahre Klasse bekannt, können die beiden Klassen miteinander verglichen werden: Die vorhergesagte mit der Ground Truth Klasse. Dabei unterscheiden sich die Instanzen zwischen den verschiedenen Methoden, z.B. sind die Instanzen Bilder in der Klassifikation, hingegen einzelne Pixel in der Semantischen Segmentierung.

Werden mehr als zwei Klassen unterschieden, kann man den Vergleich auf ein binäres Problem reduzieren. Das bedeutet, für eine gegebene Klasse wird nur verglichen, ob es sich um die selbe Klasse handelt (positiv) oder irgendeine andere Klasse (negativ). Somit reduziert sich der Vergleich für binäre Klassifikationsprobleme auf die folgenden vier möglichen Einheiten (die nicht alle bei jeder Methode Anwendung finden):

- ’True Positives’ (TP: positiv vorhergesagt, positiv gelabelt),
- ’True Negatives’ (TN: negativ vorhergesagt, negativ gelabelt),
- ’False Positives’ (FP: positiv vorhergesagt, negativ gelabelt),
- ’False Negatives’ (FN: negativ vorhergesagt, positiv gelabelt).

Eine Konfusionsmatrix ist eine Tabelle mit solchen Vergleichen. Sie hilft, die Leistung des Netzwerkes für die einzelnen Klassen darzustellen. Für jede Klasse listet sie, wie viele Instanzen das Netzwerk welcher Klasse zugeordnet hat. Z.B. für einen Klassifikator, der die drei Klassen ’apple’, ’peach’ und ’pear’ unterscheidet, zeigt die Tabelle wie viele Bilder mit der Ground Truth Klasse ’apple’ auch wirklich als ’apple’ erkannt wurden und wie viele als ’peach’, bzw. ’pear’ klassifiziert wurden. Es versteht sich, dass dies auch für die anderen Klassen angegeben wird. In HALCON repräsentiert jede Spalte die Ground Truth (wahren) Instanzen einer Klasse und jede Zeile die Instanzen der vom Netzwerk vorhergesagten Klasse.



(1) (2)

Ein Beispiel für eine Konfusionsmatrix aus der Klassifikation. In diesem Beispiel werden 68 Bilder der Klasse ’apple’ auch als solche erkannt (TP), ebenso werden 60 Bilder die keinen Vertreter der Klasse ’apple’ zeigen, auch korrekt als ’peach’ (30) oder ’pear’ (30) klassifiziert (TN). 0 Bilder gehören zur Klasse ’peach’

oder 'pear' und werden als 'apple' klassifiziert (FP), aber 24 Bilder der Klasse 'apple' werden als 'peach' (21) oder 'pear' (3) klassifiziert (FN). (1) Die Konfusionsmatrix mit allen drei zu unterscheidenden Klassen. Es scheint als ob der Klassifikator Äpfel und Pfirsiche öfters 'verwechselt' als andere Kombinationen. (2) Die Konfusionsmatrix des Binärproblems zur deutlicheren Darstellung der Klasse 'apple'.

Glossar

Im Folgenden beschreiben wir die wichtigsten Begriffe, welche wir im Kontext von Deep Learning verwenden:

Anchor Anchors sind festgesetzte umschließende Rechtecke. Diese dienen als Referenzrechtecke, aus denen das Netzwerk umschließende Rechtecke für die zu lokalisierenden Objekte vorschlägt.

Annotation Eine Annotation ist die Ground Truth Information darüber, was eine in den Daten gegebene Instanz repräsentiert. Dies ist z.B. das umschließende Rechteck und das zugehörige Label für eine Instanz der Objektdetektion. Dabei wird diese Information in einer dem Netzwerk zugänglichen Form gegeben.

Anomalie Eine Anomalie meint dabei eine Abweichung zum Normalen, etwas Unbekanntes.

Backbone Ein Backbone ist ein Teil eines vortrainiertes Klassifikationsnetzwerk. Die Aufgabe des Backbones ist das Generieren verschiedener Merkmalskarten, weshalb der klassifizierende Layer entfernt wurde.

Batchgröße - Hyperparameter *'batch_size'* Der Datensatz wird in kleinere Untergruppen, genannt Batches, unterteilt. Die Batchgröße bestimmt die Anzahl Bilder eines einzelnen Batches und somit die Anzahl Bilder, welche simultan verarbeitet werden.

COCO COCO ist eine Abkürzung aus dem Englischen "common objects in context"und bezeichnet einen großen Datensatz zur Objektdetektion, zur Segmentierung und zum Versehen von Überschriften. Dabei gibt es für die verschiedenen Arten von Annotationen jeweils ein gängiges Datenformat.

Convolutional Neural Networks (CNNs) Convolutional Neural Networks sind Netzwerke, die in Deep Learning verwendet werden und mindestens einen Convolutional Layer (Faltungs-Layer) beinhalten. CNNs sind besonders erfolgreich bei der Klassifizierung von Bildern.

Daten Im Kontext von Deep Learning verwenden wir den Begriff Daten für die zu erkennenden Instanzen (z.B. Bilder) mit ihren zugehörigen Information bezüglich der vorherzusagenden Eigenschaften (z.B. die Labels im Fall der Klassifikation).

Datenanreicherung Die Datenanreicherung ist die Erstellung von veränderten Kopien von vorhandenen Samples des Datensatzes zur Erhöhung der Vielfalt im Datensatz. Dies wird z.B. über eine Spiegelung oder Rotation erreicht.

Datensatz: Trainings-, Validierungs- und Test-Datensatz Der Datensatz bezeichnet die komplette Menge an Daten, welche für das Training verwendet werden. Der Datensatz wird in drei disjunkte Untermengen aufgeteilt:

- Der Trainings-Datensatz enthält die Daten, auf welchen der Algorithmus das Netzwerk trainiert.
- Der Validierungs-Datensatz enthält die Daten, auf welchen die Leistungsfähigkeit des Netzwerkes noch während des Trainings eingeschätzt wird.
- Der Test-Datensatz wird verwendet, um die Inferenzgenauigkeit des Netzwerks zu testen. D.h. die Fähigkeit der Inferenz (Vorhersage) wird auf neuen Daten getestet, also auf Daten ohne Einfluss auf die Optimierung des Netzwerkes.

Deep Learning Der Begriff "Deep Learning"wurde ursprünglich für das Training Neuronaler Netzwerke mit mehreren verborgenen Layer verwendet. Heutzutage wird der Begriff mehr als allgemeiner Begriff für verschiedene Methoden des maschinellen Lernens verwendet. In HALCON verwenden wir den Begriff als für Verfahren des maschinellen Lernens mit einem Neuronalen Netzwerk, welches mehrere verborgene Layer beinhaltet.

Ebene Mit Ebene wird bei innerhalb eines Merkmalspyramiden-Netzwerkes eine Gruppe von Layern bezeichnet, deren Merkmalskarten die selbe Höhe und Breite aufweisen. Dabei repräsentiert das Eingabebild die Ebene 0.

Epoche Im Kontext von Deep Learning bedeutet Epoche eine einzelne Trainingsiteration über den ganzen Trainings-Datensatz, d.h. über sämtliche Batches. Eine solche Trainingsiteration über Epochen ist nicht zu verwechseln mit einer Iteration über einen einzelnen Batch (d.h., innerhalb einer Epoche).

Fehler Im Kontext von Deep Learning verwenden wir den Begriff Fehler, wenn für eine Instanz die vom Netzwerk zugeordnete Klasse nicht mit der eigentlichen Klasse (z.b. dem Label des Bildes im Falle der Klassifikation) übereinstimmt. Dabei meinen wir jeweils den top-1 Fehler (siehe top-k Fehler).

Gewichte Gewichte sind die freien Parameter des Netzwerks, die während des Trainings zwecks Optimierung der Zielfunktion geändert werden. Ein Layer mit Gewichten multipliziert oder addiert diese mit seinen Eingabewerten. Im Gegensatz zu Hyperparametern werden Gewichte während des Trainings optimiert und werden somit geändert.

Head Heads sind Teil-Netzwerke, die aus einem oder mehreren Layer bestehen. Bei gewissen Architekturen setzen sie an den ausgewählten Pyramidenstufen an. Sie generieren die Ausgabe des Gesamt-Netzwerkes und erzeugen damit die Eingabe der Zielfunktionen.

Heads sind Teil-Netzwerke. Bei gewissen Architekturen setzen sie an den ausgewählten Pyramidenstufen an. Diese Teil-Netzwerke verarbeiten die Information aus vorherigen Teilen des Gesamt-Netzwerkes um daraus ortsaufgelöste Ausgaben für z.B. die Klassen-Vorhersagen zu generieren. Daraus generieren sie die Ausgabe des Gesamt-Netzwerkes und erzeugen damit die Eingabe der Zielfunktionen.

Hyperparameter Jede Methode des maschinellen Lernens, und somit auch CNNs, ist eine Umsetzung von Formeln mit mehreren Parametern. Während der Trainingsphase lernt das Modell von den Daten im Sinne von Optimierung der Parameter für die gegebene Aufgabe. Solche Modelle können aber auch noch weitere Parameter haben, deren Wert während der Trainingsphase nicht direkt erlernt wird. Die Werte dieser Parameter werden vor Trainingsbeginn festgelegt. Wir bezeichnen diese letzteren Parameter als Hyperparameter, um sie von jenen Netzwerkparametern (den Gewichten) zu unterscheiden, welche während der Trainingsphase optimiert werden. Aus einem anderen Blickpunkt betrachtet sind Hyperparameter Solver-spezifische Parameter.

Prominente Beispiele für Hyperparameter sind die initiale Lernrate oder die Batchgröße.

Inferenzphase Die Inferenzphase bezeichnet die Phase, in dem ein trainiertes Netzwerk angewandt wird, um die Klassenzugehörigkeit von Instanzen (welche das gesamte Eingabebild oder nur ein Teil davon sein können) und eventuell deren Lokalisierung zu folgern. Im Unterschied zur Trainingsphase wird das Netzwerk nicht mehr optimiert.

Intersection over Union Die Intersection over Union (IoU) bezeichnet ein Maß, welches die Überlappung zweier Bereiche quantifiziert. Von diesen zwei Bereichen oder Flächen wird die Schnittmenge, Intersection, gebildet, sowie die Vereinigungsmenge, Union. Die IoU ist das Verhältnis der beiden Flächen Intersection und Union.

Die Anwendung dieses Konzepts kann sich für verschiedene Methoden unterscheiden.

Klasse Klassen sind getrennte Kategorien, welche das Netzwerk unterscheidet (z.B. 'apple', 'peach', 'pear'). In HALCON wird die Klasse einer Instanz durch seine zugehörige Annotation gegeben.

Klassenagnostisch Klassenagnostisch bedeutet ohne Berücksichtigung der verschiedenen Klassen.

In HALCON wird der Term bei der Reduktion überlappender umschließender Rechtecke verwendet. Bei einer klassenagnostischen Unterdrückung umschließender Rechtecke geschieht die Unterdrückung der umschließenden Rechtecke ohne Beachtung ihrer zugeordneten Klasse, d.h., sich stark überlappende Instanzen werden unabhängig ihrer Klasse unterdrückt.

Klassifikator Im Kontext von Deep Learning bezeichnen wir mit Klassifikator folgendes: Der Klassifikator folgert für ein erhaltenes Bild die Konfidenz zu allen unterschiedenen Klassen. Als Beispiel seien die drei Klassen 'apple', 'peach', 'pear' unterschieden. Nun wenden wir den Klassifikator auf ein Bild eines Apfels an. Als Resultat erhalten wir Konfidenzen wie 'apple': 0.92, 'peach': 0.07, 'pear': 0.01.

Konfidenz Die Konfidenz ist eine Zahl, welche die Klassenzugehörigkeit einer Instanz darstellt. In HALCON ist die Konfidenz die Wahrscheinlichkeit, gegeben im Intervall [0,1]. Alternativer Name: Score

Konfusionsmatrix Eine Konfusionsmatrix ist eine Tabelle, in welcher die vorhergesagten Klassen des Netzwerks (top-1) mit der jeweiligen Ground Truth Klasse verglichen werden. Sie wird oft zur Visualisierung der Leistungsfähigkeit des Netzwerks auf dem Validierungs- oder Test-Datensatz verwendet .

Label Labels sind beliebige Zeichenketten (Strings), welche die Klasse eines Bildes angeben. In HALCON ist das Label eines Bildes gegeben durch seinen Namen (evt. gefolgt von einer Kombination aus Unterstrich und Ziffern) oder dem Namen seines direkten Ordners, z.B. 'apple_01.png', 'peach.png', 'pear/01.png'.

Layer und Verborgene Layer Layer sind Bausteine von Neuronalen Netzwerken und führen eine spezifische Funktion aus (z.B. Faltung, Vereinigung, etc., siehe „`Solution Guide Classification`"). Ein Layer kann man als eine Box betrachten, welche Eingabewerte erhält, diese transformiert und den Rückgabewert an den nächsten Layer weiter gibt. Eingabe- und Ausgabe-Layer sind jeweils mit den Daten verbunden, d.h. mit den Bilder, bzw. den Labels. Alle dazwischenliegenden Layers bezeichnet man als verborgenen Layer.

Alternative Namen: Schicht (Layer), Zwischenschicht (hidden Layer)

Lernrate - Hyperparameter *'learning_rate'* Die Lernrate ist die Gewichtung des Gradienten (siehe SGD) beim Anpassen der Argumente der Zielfunktion. In einfachen Worten: Der Gradient sagt uns, in welche Richtung optimiert werden soll und die Lernrate sagt, wie groß unser Schritt sein wird.

Merkmalskarte Eine Merkmalskarte ist die Ausgabe eines gegebenen Layers. Alternativer Name: Feature Map

Merkmalspyramide Die Merkmalspyramide ist eine Gruppe von Merkmalskarten, wobei jede Merkmalskarte aus einer anderen Ebene ist, d.h. kleiner als die vorhergehende Merkmalskarte ist.

Momentum - Hyperparameter *'momentum'* Das Momentum wird bei der Optimierung der Zielfunktionsargumente eingesetzt. Werden die Argumente der Zielfunktion aktualisiert (nach Berechnung des Gradienten), wird der Aktualisierungsvektor des vorherigen Aktualisierungsschrittes zu einem Anteil μ berücksichtigt. Dadurch werden Oszillationen gedämpft. Wir nennen den Hyperparameter μ Momentum. Wird μ auf 0 gesetzt, hat die Momentenmethode keinen Einfluss mehr. In einfachen Worten, wenn wir die Argumente aktualisieren, erinnern wir uns noch an den Schritt der letzten Aktualisierung. Wir gehen nun einen Schritt mit einer Länge entsprechend der Lernrate in Richtung des Gradienten. Zusätzlich wiederholen wir den Schritt, den wir bei der letzten Aktualisierung gingen, machen ihn jetzt aber nur noch μ mal so weit.

Nachtrainieren Wir verstehen unter Nachtrainieren das Anpassen der Gewichtungen eines bereits vortrainierten Netzwerkes, d.h. während des Nachtrainierens lernt ein Netzwerk eine spezifische Aufgabe.

Non-Maximum Suppression Bei der Objektdetektion wird die Non-Maximum Suppression verwendet um umschließende Rechtecke, die vom Netzwerk vorhergesagt wurden und sich dabei überlappen, zu unterdrücken. Überlappen sich verschiedene Instanzen stärker als durch einen Schwellenwert erlaubt, wird nur die Instanz mit höchstem Konfidenzwert beibehalten, während alle anderen Instanzen unterdrückt werden.

Regularisierung - Hyperparameter *'weight_prior'* Regularisierung ist eine Methode um Überanpassung von Neuronalen Netzwerken zu vermeiden. Dies geschieht durch Addition eines Strafterms zur Zielfunktion. Dieser Regularisierungsterm schiebt die Gewichte gegen Null und bestraft dabei insbesondere große Gewichte. Einfach gesagt führt dies dazu, das einfachere Modelle bevorzugt werden, die sich weniger an Rauschen in den Daten anpassen und besser generalisieren. In HALCON wird die Regularisierung durch den Parameter *'weight_prior'* gesteuert. Dieser Parameter wird in der Literatur auch Regularisierungsparameter, weight decay, oder einfach λ genannt (Wir vermerken, dass wir innerhalb von HALCON λ für die Lernrate verwenden und in Formeln das Symbol α für den Regularisierungsparameter).

Solver Der Solver optimiert das Modell, indem er die Gewichtungen so anpasst, dass die Zielfunktion optimiert (d.h. minimiert) wird.

Stochastisches Gradientenverfahren (engl.: stochastic gradient descent, SGD) Das stochastische Gradientenverfahren ist ein iterativer Optimierungsalgorithmus für differenzierbare Funktionen. Ein besonderes Merkmal bildet die Einschränkung der für einen einzelnen Iterationsschritt berücksichtigten Datenmenge, denn für eine Iteration wird nur ein einzelner Batch mit zufällig gewählten Bildern berücksichtigt. Bei Deep Learning verwenden wir den SGD zur Optimierung (d.h. Minimierung) der Zielfunktion.

Top-k Fehler Der Klassifikator ordnet einem Bild nicht eine einzelne Klasse zu, sondern bezüglich jeder Klasse eine Konfidenz. Für ein Bild gibt es somit k Klassen mit höchster vorhergesagter Konfidenz. Der top-k Fehler gibt nun an, bei wie vielen Vorhersagen diese k Klassen die Ground Truth Klasse (das Label) beinhalten. Der top-1 Fehler gibt somit an, ob die inferierte Klasse mit größter Wahrscheinlichkeit der Ground Truth, vom Benutzer gegebenen Klasse entspricht. Beim top-3 Fehler wird geprüft, ob das die Ground Truth Klasse einer der drei besten Vorhersagen entspricht (d.h. einer der drei Klassen, die der Klassifikator als am wahrscheinlichsten zutreffend betrachtet).

Transferlernen (engl.: transfer learning) Transferlernen meint eine Technik, bei der für ein neues Netzwerk das Wissen eines bereits existierenden und trainierten Netzwerkes die Basis bildet. Konkret wird ein bereits (vor)trainiertes Netzwerk inklusive seiner Gewichtungen übernommen und der Ausgabe-Layer für die spezifische Anwendung adaptiert. In HALCON betrachten wir auch den darauf folgenden Schritt des Nachtrainierens als Teil des Transferlernens.

Überanpassung (engl.: overfitting) Man spricht von Überanpassung, wenn das Netzwerk anfängt, sich die Trainingsdaten zu merken, anstatt generelle Regeln zur Klassifikationsaufgabe zu finden. Überanpassung wird ersichtlich, wenn das Modell den Fehler auf dem Trainings-Datensatz weiter minimiert, während der Fehler auf dem Validierungs-Datensatz steigt. Die meisten Neuronalen Netzwerke haben eine sehr große Anzahl Gewichtungsfaktoren, weshalb diese Netzwerke besonders anfällig für Überanpassung sind.

Umschließendes Rechteck Umschließende Rechtecke definieren einen bestimmten Teil innerhalb des Bildes. Sie werden auch dazu benutzt, ein Objekt im Bild zu lokalisieren.

Unteranpassung (engl.: underfitting) Unteranpassung passiert, wenn das Modell übergeneralisiert. D.h. das Modell ist nicht fähig, die Komplexität der Aufgabe zu erfassen. Dies wird ersichtlich, wenn der Wert der Zielfunktion oder der Fehler auf dem Trainings-Datensatz nicht signifikant gemindert wird.

Veränderungsstrategie Eine Veränderungsstrategie bestimmt, wann und wie die Hyperparameter während des Trainings eines DL Modells geändert werden.

Zielfunktion (engl.: loss) Eine Zielfunktion vergleicht die Vorhersage des Netzwerks mit der gegebenen Information, was es im Bild finden soll (und gegebenenfalls wo) und bestraft Abweichungen. Beim Training werden die Argumente dieser Zielfunktion optimiert, um das Netzwerk für seine spezifische Aufgabe zu trainieren.

```
get_dl_device_param ( : : DLDeviceHandle,
    GenParamName : GenParamValue )
```

Auslesen der Parameter einer Deep Learning Hardware-Einheit.

`get_dl_device_param` gibt die Parameterwerte von `GenParamName` für die Hardware-Einheit (im Folgenden Einheit genannt) `DLDeviceHandle` in `GenParamValue` aus. Siehe `query_available_dl_devices` für weitere Informationen zu den Handles der Einheiten.

Unterstützte Werte für `GenParamName` sind:

'calibration_precisions': Gibt die Datentypen der Einheit an, die für eine Kalibrierung eines Deep-Learning Modells verwendet werden können.

Mögliche Werte: *'int8'*.

'cast_precisions': Gibt die Datentypen der Einheit an, die für ein Cast-Prozess eines Deep-Learning Modells verwendet werden können.

Mögliche Werte: *'float32'*, *'float16'*.

'conversion_supported': Gibt den Wert *'true'* zurück, falls die Einheit entweder Datentypen für eine Kalibrierung oder für einen Cast-Prozess unterstützt, ansonsten *'false'*.

'id': ID der Einheit. Innerhalb einer AI²-Schnittstelle (siehe *'ai_accelerator_interface'*) sind die IDs der Einheiten eindeutig. Das Gleiche gilt für Einheiten die standardmäßig von HALCON unterstützt werden.

'ai_accelerator_interface': AI Accelerator Interface (AI²), die Schnittstelle über welche die Einheit `DLDeviceHandle` ausgeführt wird. Handelt es sich um eine standardmäßig von HALCON unterstützte Einheit, wird *'none'* zurückgegeben.

'inference_only': Gibt an, ob die Einheit nur zur Inferenz mit Deep Learning-Modellen (*'true'*) oder auch für das Training und Gradienten-erforderliche Operationen (*'false'*) verwendet werden kann.

'info': Dictionary mit zusätzlichen Informationen über die Einheit.

Restriktion: Nur für Einheiten, die über die AI²-Schnittstelle unterstützt werden.

'name': Name der Einheit.

'optimize_for_inference_params': Dictionary mit default benutzerdefinierten Konversions-Parametern für eine Kalibrierung oder einen Cast-Prozess eines Deep-Learnings Modells, dessen Einträge angepasst werden können *Restriktion*: Nur für Einheiten, die über die AI²-Schnittstelle unterstützt werden.

'precisions': Gibt die Datentypen an, welche die Einheit für die Gewichte und/oder Aktivierungen eines Deep-Learning Modells unterstützt.

Mögliche Werte: *'float32'*, *'float16'*, *'int8'*.

'settable_device_params': Dictionary mit setzbaren Werten für eine Hardware-Einheit. *Restriktion*: Nur für Einheiten, die über die AI²-Schnittstelle unterstützt werden.

'type': Typ der Einheit.

Parameter

▷ **DLDeviceHandle** (input_control) . . . . . dl_device ⇝ *handle*
Handle der Einheit.

▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'type'
Werteliste: `GenParamName` ∈ {'calibration_precisions', 'cast_precisions', 'conversion_supported', 'id', 'ai_accelerator_interface', 'inference_only', 'info', 'name', 'optimize_for_inference_params', 'precisions', 'settable_device_params', 'type'}

▷ **GenParamValue** (output_control) . . . . . attribute.name(-array) ⇝ *string* / real / integer
Wert des generischen Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_dl_device_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`query_available_dl_devices`

Nachfolger

`set_dl_model_param`

Modul

Foundation

```
optimize_dl_model_for_inference ( : : DLModelHandle,
    DLDeviceHandle, Precision, DLSamples,
    GenParam : DLModelHandleConverted, ConversionReport )
```

Optimiere ein Modell für die Inferenz auf einer Einheit über die AI²-Schnittstelle.

Der Operator `optimize_dl_model_for_inference` optimiert das Eingabe-Modell `DLModelHandle` für die Inferenz auf der Einheit `DLDeviceHandle` und gibt das optimierte Modell in `DLModelHandleConverted` zurück. Dieser Operator hat zwei unterschiedliche Funktionalitäten: Das Casten der Modell-Präzision in `Precision` und Kalibrieren des Modells basierend auf den gegebenen `DLSamples`. Zusätzlich kann in beiden Fällen die Modellarchitektur für das `DLDeviceHandle` optimiert werden.

Der Parameter `DLDeviceHandle` gibt die Deep-Learning-Einheit an, für welche das Modell optimiert wird. Ob die Einheit die Optimierung unterstützt, kann über `get_dl_device_param` mit *'conversion_supported'* ermittelt werden. Nach einer erfolgreichen Ausführung von `optimize_dl_model_for_inference` wird für das Ausgabe-Modell `DLModelHandleConverted` der Parameter *'precision_is_converted'* auf *'true'* gesetzt.

Der Parameter `Precision` gibt die Genauigkeit an, in die das Modell umgerechnet werden soll. Standardmäßig haben Modelle, die von HALCON geliefert werden die `Precision` *'float32'*. Die folgenden Werte werden für `Precision` unterstützt:

- *'float32'*
- *'float16'*
- *'int8'*

Der Parameter DLSamples gibt die Samples an, auf welchen die Kalibrierung durchgeführt wird. Folglich sollten sie repräsentativ sein. Es wird empfohlen, sie aus dem Trainingssplit bereitzustellen. Für die meisten Anwendungen reichen 10-20 Samples pro Klasse für gute Ergebnisse.

Beachten Sie, dass die Samples für einen reinen Cast-Prozess nicht benötigt werden. In diesem Fall kann für DLSamples ein leeres Tupel übergeben werden.

Der Parameter GenParam legt weitere, Einheitenspezifische Parameter und deren Werte fest. Welche Parameter für das gegebene DLDeviceHandle in GenParam zu setzen sind sowie deren Defaultwerte, kann über den Operator get_dl_device_param mit dem Parameter *'optimize_for_inference_params'* abgefragt werden.

Beachte, gewisse Einheiten erwarten auch nur ein leeres Dictionary.

Der Parameter ConversionReport gibt einen Report-Dictionary mit Informationen zur Konvertierung zurück.

Achtung

Dieser Operator kann nur über ein AI²-Interface verwendet werden. Außerdem können nach der Konvertierung keine Parameter mehr mit set_dl_model_param für DLModelHandleConverted gesetzt werden (außer *'runtime'* und *'device'*)

Parameter

- ▷ **DLModelHandle** (input_control) ..... dl_model ⇝ *handle*
  Eingabe-Modell.
- ▷ **DLDeviceHandle** (input_control) ..... dl_device(-array) ⇝ *handle*
  Einheit-Handle, das für die Optimierung verwendet wird.
- ▷ **Precision** (input_control) ..... string ⇝ *string*
  Präzision, in die das Modell konvertiert werden soll.
- ▷ **DLSamples** (input_control) ..... dict-array ⇝ *handle*
  Samples, die für die Optimierung benötigt werden.
- ▷ **GenParam** (input_control) ..... dict ⇝ *handle*
  Parameter Dictionary für Optimierung.
- ▷ **DLModelHandleConverted** (output_control) ..... dl_model ⇝ *handle*
  Ausgabe-Modell mit neuer Präzision.
- ▷ **ConversionReport** (output_control) ..... dict ⇝ *handle*
  Ausgabe-Report der Optimierung.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert optimize_dl_model_for_inference den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_dl_model_batch, query_available_dl_devices

Nachfolger

set_dl_model_param, apply_dl_model

Modul

Deep Learning Inference

query_available_dl_devices (: : GenParamName, GenParamValue : DLDeviceHandles)

Erzeugen einer Liste von Hardware-Einheiten für Deep Learning-basierte Anwendungen

query_available_dl_devices gibt eine Liste von Handles zurück. Jedes Handle entspricht einer Hardware-Einheit (im Folgenden Einheit genannt), welche für das Trainieren oder Inferieren von Deep Learning-Modellen zur Verfügung stehen.

Eine Einheit für Deep Learning ist entweder standardmäßig von HALCON oder über eine AI²-Schnittstelle unterstützt.

Die Einheiten, die standardmäßig von HALCON unterstützt werden, entsprechen jenen die mit set_dl_model_param über *'runtime' = 'cpu'* oder *'runtime' = 'gpu'* für ein Deep Learning-Modell gesetzt werden können. Für diese Einheiten wird eine HALCON interne Implementierung für die Inferenz oder das Training des Modells ausgeführt. Siehe Deep Learning für mehr Details.

Einheiten, die über die AI²-Schnittstelle unterstützt werden, können ebenfalls über set_dl_model_param gesetzt werden. Die Inferenz des Deep Learning Modells wird in diesem Fall jedoch nicht durch HALCON, sondern auf der Einheit selbst.

query_available_dl_devices gibt für jede Einheit, die von HALCON für Deep Learning-Anwendungen unterstützt wird, ein Handle zurück. Also sowohl für jene Einheiten auf denen HALCON läuft als auch für mögliche Inference Engines.

Mit GenParamName kann nach Einheiten gefiltert werden. Zum Filtern können alle Werte GenParamName, die von get_dl_device_param zur Verfügung gestellt werden und nicht ein Handle für GenParamValue zurückgeben, verwendet werden. Eine Liste von unterstützten Werten ist in der Operatorreferenz von get_dl_device_param zu finden. Zusätzlich sind noch folgende Werte für GenParamName unterstützt:

'runtime': Die Einheiten, die von HALCON für diesen Einheiten-Typ unterstützt werden.
Mögliche Werte (standardmäßig): *'cpu'*, *'gpu'*.

GenParamName kann mehrere gleichnamige Einträge enthalten. In diesem Fall werden die entsprechenden Werte bei der Filterung mit einem logischen *'oder'* verbunden. Beispiele wie der Filter verwendet werden kann sind im Beispielcode zu finden.

Parameter

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: []
Werteliste: GenParamName ∈ {'calibration_precisions', 'cast_precisions', 'conversion_supported', 'id', 'ai_accelerator_interface', 'inference_only', 'name', 'optimize_for_inference_params', 'precisions', 'runtime', 'settable_device_params', 'type'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Wert des generischen Parameters.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {}

▷ **DLDeviceHandles** (output_control) ..... dl_device(-array) ⇝ *handle*
Liste von DLDevice Handles

Beispiel

```
* Query all deep-learning-capable hardware devices
query_available_dl_devices ([], [], DLDeviceHandles)

* Query all GPUs with ID 0 or 2
query_available_dl_devices (['type', 'id', 'id'], ['gpu', 0, 2],\
                            DLDeviceHandles)

* Query the unique GPU with ID 1 supported by HALCON
query_available_dl_devices (['runtime', 'id'], ['gpu', 1], DLDeviceHandles)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert query_available_dl_devices den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`get_dl_device_param`

Modul

Foundation

`set_dl_device_param` `( : : DLDeviceHandle, GenParamName, GenParamValue : )`

Setzen der Parameter einer Deep Learning Hardware-Einheit.

`set_dl_device_param` setzt Parameterwerte für die Hardware-Einheit (im Folgenden Einheit genannt) `DLDeviceHandle`. Siehe `query_available_dl_devices` für weitere Informationen zu den Handles der Einheiten.

Parameter

- ▷ **`DLDeviceHandle`** (input_control) . . . . . dl_device ⇝ *handle*
  Handle der Einheit.
- ▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Name des generischen Parameters.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'settable_device_params'}
- ▷ **`GenParamValue`** (input_control) . . . . . attribute.name(-array) ⇝ *string* / real / integer / handle
  Wert des generischen Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_dl_device_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`query_available_dl_devices`

Nachfolger

`set_dl_model_param`

Siehe auch

`get_dl_device_param`

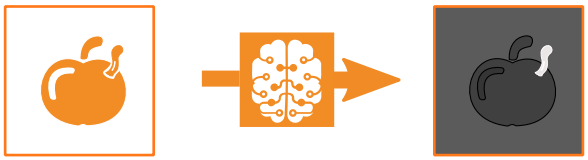
Modul

Foundation

8.1 Anomalieerkennung

Dieses Kapitel beschreibt, wie Deep Learning-basierte Anomalieerkennung genutzt wird.

Mit der Anomalieerkennung will man ermitteln, ob ein Bild Abweichung zum Normalen, d.h. etwas Unbekanntes enthält.



Ein mögliches Beispiel der Anomalieerkennung: Jedem Pixel des Eingabebildes wird ein Wert zugewiesen. Dieser gibt an, wie wahrscheinlich der Pixel ein unbekanntes Merkmal, eine Anomalie, zeigt. Der Wurm ist nicht Teil der während des Trainings gesehenen Äpfel und so erhalten seine Pixel einen höheren Score.

Während des Trainings lernt ein Anomalieerkennungsmodell die Gemeinsamkeiten von Trainingsbildern. Das trainierte Modell inferiert, wie wahrscheinlich ein Eingabebild nur erlernte Merkmale oder auch etwas davon Abweichendes, also Anomalien, zeigt. Dieses Inferenzergebnis gibt das Modell als Grauwertbild zurück. Die Pixelwerte dieses Grauwertbildes sind Scores, die angeben, wie wahrscheinlich die entsprechenden Pixel im Eingabebild eine Anomalie zeigen.

Genereller Ablauf

Dieser Abschnitt beschreibt den generellen Ablauf einer Anomalieerkennungsaufgabe mittels Deep Learning.

Vorverarbeitung des Datensatzes Dieser Abschnitt behandelt die notwendigen Anpassungen des Datensatzes.

1. Die Informationen des Datensatzes müssen eingelesen werden. Dazu kann die Prozedur

   - `read_dl_dataset_anomaly`

   verwendet werden. Dabei wird ein Dictionary `DLDataset` erstellt, welches als Datenbank dient und alle notwendigen Informationen über die Daten abspeichert. Für mehr Information zu den Daten und wie sie übergeben werden, wird auf den unteren Absatz „Daten“ und das Kapitel Deep Learning / Modell verwiesen.

2. Die Daten von `DLDataset` sollen in drei getrennte Datensätze aufgeteilt werden. Dazu kann die Prozedur

   - `split_dl_dataset`

   verwendet werden.

3. Das Netzwerk stellt mehrere Anforderungen an die Bilder (wie z.B. die Bilddimensionen und den Grauwertbereich). Diese Werte können mit

   - `get_dl_model_param`

   abgefragt werden. Dafür muss das Modell erst eingelesen werden mittels

   - `read_dl_model`.

4. Der Datensatz kann nun vorverarbeitet werden. Dazu kann die Prozedur

   - `preprocess_dl_dataset`

   verwendet werden. Für die Implementierung einer selbst erstellten Vorverarbeitung kann man sich an dieser Prozedur orientieren.

   Um diese Prozedur zu verwenden, müssen die Vorverarbeitungsparameter wie z.B. die Bildgröße festgelegt werden. Die Parameter und ihre Werte sind im Dictionary `DLPreprocessParam` zu speichern. Zur Erzeugung dieses Dictionaries kann die Prozedur

   - `create_dl_preprocess_param`

   verwendet werden. Es wird empfohlen, dieses Dictionary `DLPreprocessParam` abzuspeichern, um während der späteren Inferenz-Phase Zugang zu den Werten zu haben.

Trainieren des Modells Dieser Abschnitt behandelt das Trainieren eines Anomalieerkennungsmodells.

1. Setzen der Trainingsparameter und Abspeichern derselben im Dictionary `TrainParam`. Für dies kann die Prozedur

   - `create_dl_train_param`

   verwendet werden.

2. Trainieren des Modells. Dafür kann die Prozedur
   - `train_dl_model`

   verwendet werden. Diese Prozedur benötigt:
   - das Handle des Modells, `DLModelHandle`
   - das Dictionary mit den Informationen über den Datensatz, `DLDataset`
   - das Dictionary mit den Trainingsparametern, `TrainingParam`

Evaluierung des trainierten Modells Dieser Abschnitt behandelt das Evaluieren eines Anomalieerkennungsmodells.

1. Setzen aller Modellparameter, welche die Evaluieren beeinflussen können.
2. Die Evaluierung kann bequem mit der Prozedur
   - `evaluate_dl_model`

   durchgeführt werden. Diese Prozedur erwartet ein Dictionary `GenParamEval` mit den Evaluierungsparametern.
3. Das Dictionary `EvaluationResults` enthält die angefragten Evaluierungsmaße.

Inferenz auf neuen Bildern Dieser Abschnitt behandelt die Inferenz auf neuen Bildern durch ein Anomalieerkennungsmodell. Für ein trainiertes Modell sind die folgenden Schritte durchzuführen:

1. Abfragen der Anforderungen des Modells an die Bilder über den Operator
   - `get_dl_model_param`

   oder die Prozedur
   - `create_dl_preprocess_param_from_model`.
2. Setzen der Modellparameter, wie im Abschnitt „Modellparameter“ beschrieben, über den Operator
   - `set_dl_model_param`.
3. Generieren des Dictionaries `DLSample` für jedes zu inferierende Bild. Dafür kann die Prozedur
   - `gen_dl_samples_from_images`

   verwendet werden.
4. Jedes Bild muss auf dieselbe Art und Weise wie die Trainingsbilder vorverarbeitet werden. Es wird empfohlen, dafür die Prozedur
   - `preprocess_dl_samples`

   zu verwenden. Wurde während der Vorverarbeitung das Dictionary `DLPreprocessParam` gespeichert, kann es direkt als Eingabe verwendet werden, um die Parameterwerte festzulegen.
5. Anwenden des Modells über den Operator
   - `apply_dl_model`.
6. Die Resultate können aus dem Dictionary *'DLResult'* abgerufen werden.

Daten

Es wird zwischen den Daten für Training, Evaluierung und Inferenz auf neuen Bildern unterschieden.

Ein Grundgedanke der Datenhandhabung: Das Modell interagiert mit den Daten über Dictionaries. Dies bedeutet, das Modell erhält die Eingabedaten über das Dictionary `DLSample` und gibt die Ausgabe über das Dictionary `DLResult`, bzw. `DLTrainResult` zurück. Für weitere Informationen zur Datenhandhabung wird auf das Kapitel Deep Learning / Modell verwiesen.

Klassen Bei der Anomalieerkennung werden genau zwei Klassen unterschieden:

- *'ok'*, ohne Anomalie, Klassen-ID *0*.
- *'nok'*, mit Anomalie, Klassen-ID *1*. (auf Pixelwerten eine ID *>0*, siehe den unteren Abschnitt „Daten für die Evaluierung“).

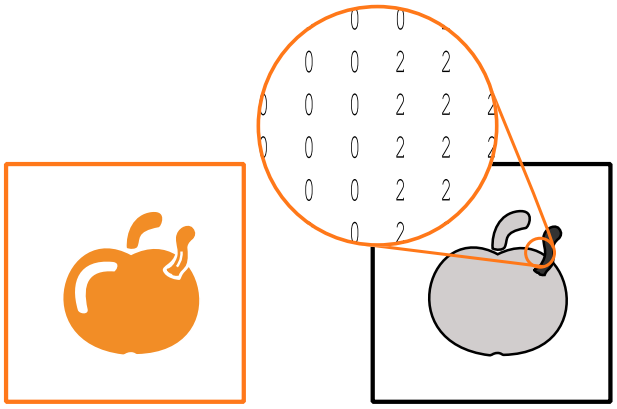
Diese Klassen treffen sowohl für das Bild als ganzes sowie auch für die einzelnen Pixel zu.

Daten für das Training Der Datensatz besteht nur aus Bildern ohne Anomalie und den dazugehörigen Informationen. Damit das Modell die Daten verarbeiten kann, müssen sie bestimmte Anforderungen erfüllen. Informationen zu den Bildanforderungen finden sich im unteren Abschnitt „Bilder“.

Die Trainingsdaten werden dazu benötigt, um das Modell für eine spezifische Aufgabe zu trainieren. Mit Hilfe dieser Daten kann das Modell lernen, welche Merkmale die Anomalie-freien Bilder gemeinsam haben.

Daten für die Evaluierung Dieser Datensatz muss Bilder ohne Anomalie enthalten, es können zusätzlich aber auch solche mit Anomalie darin sein. Jedes enthaltene Bild braucht ein Ground Truth Label `image_label`, welches die Klasse des Bildes bestimmt (siehe den obigen Abschnitt). Dadurch wird angegeben, ob auf dem Bild eine Anomalie ersichtlich ist (*'nok'*) oder nicht (*'ok'*).

Die Evaluierung kann visuell auch auf Pixelebene stattfinden, wenn ein entsprechendes Bild `anomaly_file_name` im Dictionary `DLSample` vorhanden ist. Auf dem Bild `anomaly_file_name` gibt jeder Pixel die Klassen-ID an und damit ob der entsprechende Pixel des Eingabebildes eine Anomalie zeigt (Pixelwert > *0*) oder nicht (Pixelwert *0*).

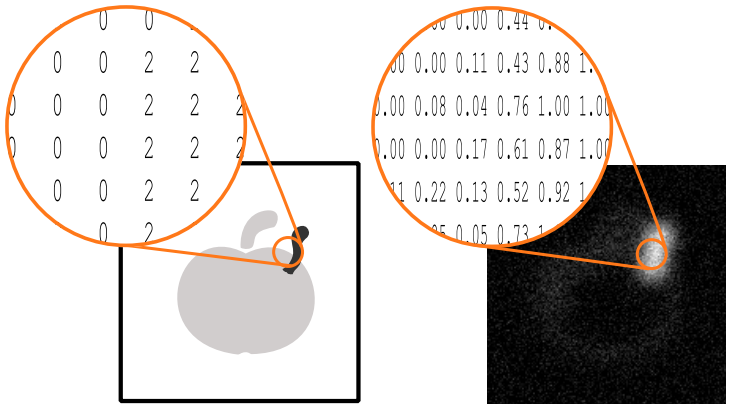


Schema des Bildes `anomaly_file_name`. Zur besseren Darstellung werden Grauwerte für die verschiedenen Zahlen verwendet. (1) Eingabebild. (2) Das entsprechende Bild `anomaly_file_name`, welches die Klassenannotationen zur Verfügung stellt, *0*: *'ok'* (weiß und hellgrau), *2*: *'nok'* (dunkelgrau).

Bilder Unabhängig von der Anwendung stellt das Modell Anforderungen an die Bilder, wie z.B. die Bilddimensionen. Die spezifischen Werte hängen vom Modell selbst ab, für die spezifischen Werte der verschiedenen Modelle siehe die Dokumentation von `read_dl_model`. Für ein eingelesenes Modell können die Werte mit `get_dl_model_param` abgefragt werden. Um diese Anforderungen zu erfüllen, müssen die Bilder eventuell vorverarbeitet werden. Die Standard-Vorverarbeitung für den ganzen Datensatz und damit auch der Bilder kann mit Hilfe der Prozedur `preprocess_dl_dataset` durchgeführt werden, bzw. `preprocess_dl_samples` für ein einzelnes Sample. Für eine selbst erstellte Vorverarbeitung liefern diese Prozeduren eine Anleitung für die Implementierung.

Modell-Ausgabe Als Trainingsausgabe gibt der Operator `train_dl_model_anomaly_dataset` ein Dictionary `DLTrainResult`. Für die Anomalieerkennung hat es die folgenden Einträge:

- `anomaly_score`: Ein Score, der angibt, wie wahrscheinlich das Bild als Ganzes eine Anomalie enthält. Die Berechnung dieses Scores basiert auf den Werten von `anomaly_image`.
- `anomaly_image`: Ein Bild, mit Scores als Pixelwerte, die angeben wie wahrscheinlich der entsprechende Pixel des Eingabebildes zu einer Anomalie gehört (siehe die untere Darstellung). Die Werte sind $\in [0, 1]$.



(1) (2)

Schema des Bildes `anomaly_image`. Zur besseren Darstellung werden Grauwerte für die verschiedenen Zahlen verwendet. (1) Das Bild `anomaly_file_name` liefert die Klassenannotationen *0*: *'ok'* (weiß und hellgrau), *2*: *'nok'* (dunkelgrau) (2) Das entsprechende Bild `anomaly_image`.

Spezifische Parameter

Bei einem Anomalieerkennungsmodell werden sowohl die Modellparameter als auch die Hyperparameter mit `set_dl_model_param` gesetzt. Für Informationen zu den Modellparametern wird auf die Dokumentation von `get_dl_model_param` verwiesen.

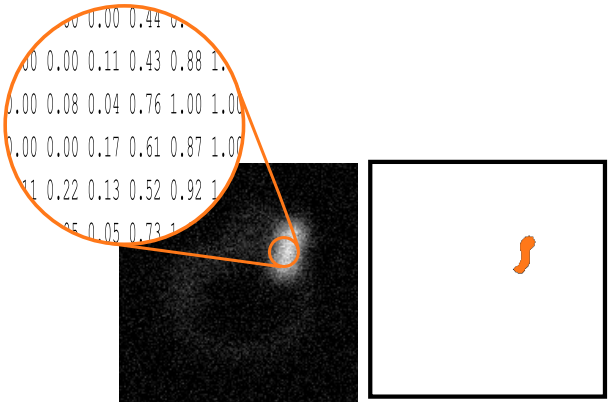
Während des Trainings wird der gesamte Datensatz auf einmal verwendet und nicht nur jeweils ein Batch. Infolgedessen haben bestimmte Parameter wie z.B. *'batch_size_multiplier'* keinen Einfluss.

Das Modell gibt Scores zurück, aber klassifiziert weder das Bild als ganzes noch die Pixel, d.h., es bestimmt nicht, ob diese eine Anomalie zeigen oder nicht. Für diese Klassifizierung müssen Schwellenwerte angegeben werden, welche den minimalen Score festlegen, damit ein Pixel oder Bild als anomal zählt. Mögliche Schwellenwerte können mit der Prozedur `compute_dl_anomaly_anomaly_thresholds` geschätzt werden. Die Schwellenwerte können mit der Prozedur `threshold_dl_anomaly_results` angewandt werden. Als Resultat fügt die Prozedur folgende (vom Schwellenwert abhängige) Einträge zum Dictionary `DLResultat` des Samples hinzu:

`anomaly_class`
: Vorhergesagte Klasse für das Bild als Ganzes (für den gegebenen Schwellenwert).

`anomaly_class_id`
: ID der vorhergesagten Klasse für das Bild als Ganzes (für den gegebenen Schwellenwert).

`anomaly_region`
: Regionen der Pixel welche als anomal bewertet wurden (für den gegebenen Schwellenwert, siehe die untere Darstellung).



(1) (2)

Schema von `anomaly_region`. Zur besseren Darstellung werden Grauwerte für die verschiedenen Zahlen verwendet. (1) Das Bild `anomaly_image` mit den erhaltenen pixelweisen Scores. (2) Die entsprechende Region `anomaly_region`.

train_dl_model_anomaly_dataset (: : DLModelHandle, DLSamples, DLTrainParam : DLTrainResult)

Trainieren eines Deep Learning-Modells zur Anomalieerkennung.

Der Operator train_dl_model_anomaly_dataset trainiert das in DLModelHandle enthaltenen Deep Learning-Modell mit *'type'='anomaly_detection'*.

Dieser Operator verarbeitet den ganzen Datensatz auf einmal. Dies ist anders als beim Operator train_dl_model_batch. Die Iterationen über den Datensatz werden vom Operator intern durchgeführt. Somit muss der Operator für das ganze Training des Modells nur einmal aufgerufen werden.

Der Trainings-Datensatz wird als Tupel von Dictionaries DLSamples übergeben. Weitere Informationen zu den verwendeten Dictionaries und zugehörigen Keys können im Kapitel Deep Learning / Modell gefunden werden. Der Operator erwartet im Trainings-Datensatz zum Trainieren des Modells nur Bilder ohne Anomalie.

Mit dem Dictionary DLTrainParam können die Hyperparameter verändert werden. Folgende Werte werden unterstützt:

- max_num_epochs: Dieser Parameter spezifiziert die maximale Anzahl an Trainingsepochen. Falls das durch error_threshold definierte Abbruchkriterium bereits in einer früheren Epoche erreicht wird, wird das Training entsprechend beendet.
  Restriktion: max_num_epochs >=*1*.
  Default: max_num_epochs = *30*.
- error_threshold: Dieser Parameter ist ein Abbruchskriterium für das Training. Ist der Trainingsfehler kleiner als der in error_threshold gesetzte Wert, wird das Training erfolgreich abgeschlossen.
  Restriktion: 0.0 <= error_threshold <= *1.0*.
  Default: error_threshold = *0.001*.
- domain_ratio: Dieser Parameter bestimmt den Anteil an Informationen jedes Bildes, die für das Training verwendet werden. Da Bilder meistens eine große Fülle von Informationen enthalten, kann es von Vorteil sein, die Menge zu reduzieren. Zusätzlich kann ein kleinerer Wert von domain_ratio die Trainingszeit reduzieren. Allerdings muss stets genügend Information übrig bleiben, weshalb dieser Parameter nicht zu klein gesetzt werden soll. Sonst kann passieren, dass das Trainingsergebnis nicht zufriedenstellend ist oder das Training gar fehlschlägt.
  Restriktion: 0.0 < domain_ratio <= *1.0*.
  Default: domain_ratio = *0.1*.
- regularization_noise: Dieser Parameter ist ein Regularisierungsfaktor, der verwendet wird, um das Training robuster zu machen.
  Restriktion: regularization_noise >=*0.0*.
  Default: regularization_noise = *0.0*.

Das Ausgabe-Dictionary DLTrainResult enthält die folgenden Werte:

- final_error: Der beste Fehler, der während des Trainings erreicht wurde.
- final_epoch Die Epoche, in welcher der Fehler final_error erreicht wurde.

Achtung

Der Operator train_dl_model_anomaly_dataset basiert auf Algorithmen, die in manchen Fällen nicht deterministisch sind. Daher können Ergebnisse von mehreren Aufrufen von train_dl_model_anomaly_dataset trotz gleicher Eingabewerte leicht voneinander abweichen.

Systemvoraussetzungen: Um diesen Operator auf der GPU auszuführen, indem *'runtime'* auf *'gpu'* gesetzt ist (siehe get_dl_model_param), werden cuDNN und cuBLAS benötigt. Für weitere Details wird auf den „Installation Guide", Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods", verwiesen. Alternativ kann dieser Operator auch auf der CPU ausgeführt werden, indem man *'runtime'* auf *'cpu'* setzt.

Parameter

- ▷ **DLModelHandle** (input_control) . . . . . dl_model ⇝ *handle*
  Handle des Deep Learning-Modells.
- ▷ **DLSamples** (input_control) . . . . . dict-array ⇝ *handle*
  Tupel von Dictionaries mit Eingabebildern und zugehörigen Informationen.
- ▷ **DLTrainParam** (input_control) . . . . . dict ⇝ *handle*
  Parameter zum Trainieren des Anomalieerkennungsmodells.
  Defaultwert: []
- ▷ **DLTrainResult** (output_control) . . . . . dict ⇝ *handle*
  Dictionary mit den Trainingsergebnissen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `train_dl_model_anomaly_dataset` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_dl_model`, `set_dl_model_param`, `get_dl_model_param`

Nachfolger

`apply_dl_model`

Siehe auch

`apply_dl_model`

Modul

Deep Learning Training

8.2 Framework

create_dl_layer_activation (: : DLLayerInput, LayerName,
ActivationType, GenParamName, GenParamValue : DLLayerActivation)

Create an activation layer.

The operator `create_dl_layer_activation` creates an activation layer whose handle is returned in `DLLayerActivation`.

The parameter `DLLayerInput` determines the feeding input layer and expects the layer handle as value.

The parameter `LayerName` sets an individual layer name. Note that if creating a model using `create_dl_model` each layer of the created network must have an unique name.

The parameter `ActivationType` sets the type of the activation. Supported activation types are:

'relu': Rectified linear unit (ReLU) activation, which is defined as follows.

$$ReLU(x) := \begin{cases} 0 & \text{if } x \leq 0, \\ x & \text{if } 0 < x \leq \beta, \\ \beta & \text{else.} \end{cases}$$

The *'upper_bound'* β can be set using the generic parameters, see below.

'sigmoid': Sigmoid activation, which is defined as follows.

$$Sigmoid(x) := \frac{1}{1 + e^{-x}}$$

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

'upper_bound': Float value defining an upper bound for a rectified linear unit. If the activation layer is part of a model which has been created using create_dl_model, the upper bound can be unset. To do so, use set_dl_model_layer_param and set an empty tuple for *'upper_bound'*.
Default: []

Certain parameters of layers created using this operator create_dl_layer_activation can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'activation_type' (ActivationType)	x	x
'input_layer' (DLLayerInput)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerActivation)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x
'upper_bound'	–	x

Parameter

▷ **DLLayerInput** (input_control) ..... dl_layer ⇝ *handle*
Feeding layer.

▷ **LayerName** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **ActivationType** (input_control) ..... string ⇝ *string*
Activation type.
Defaultwert: 'relu'
Werteliste: ActivationType ∈ {'relu', 'sigmoid'}

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: GenParamName ∈ {'is_inference_output', 'upper_bound'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'true', 'false'}

▷ **DLLayerActivation** (output_control) ..... dl_layer ⇝ *handle*
Activation layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_batch_normalization (: : DLLayerInput, LayerName, Momentum, Epsilon, Activation, GenParamName, GenParamValue : DLLayerBatchNorm)

Create a batch normalization layer.

The operator create_dl_layer_batch_normalization creates a batch normalization layer whose handle is returned in DLLayerBatchNorm. Batch normalization is used to improve the performance and stability of a neural network during training. The mean and variance of each input activation are calculated for each batch and the input values are transformed to have zero mean and unit variance. Moreover, a linear scale and shift transformation is learned. During training, to take all samples into account, the batch-wise calculated mean and variance values are combined with a Momentum into running mean and running variance, where i denotes the iteration index:

$$\begin{aligned} running_mean(i) &= (1 - \texttt{Momentum}) * mean(i) + \texttt{Momentum} * running_mean(i-1), \\ running_variance(i) &= (1 - \texttt{Momentum}) * variance(i) + \texttt{Momentum} * running_variance(i-1). \end{aligned}$$

To affect the mean and variance values you can set the following options for Momentum:

'auto': Combines mean and variance values by a cumulative moving average.

'freeze': Stops the adjustment of the mean and variance and their values stay fixed. This is usually done before fine-tuning a model.

Epsilon is a small offset to the variance and used to control the numerical stability. Usually its default value should be adequate.

The parameter DLLayerInput determines the feeding input layer.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The parameter Activation determines whether an activation is performed after the batch normalization in order to optimize the runtime performance.

- *'relu'*: perform a ReLU activation after the batch normalization.
  It is possible to specify an upper bound to the ReLU operation (see create_dl_layer_activation) via the generic parameter *'upper_bound'*.
- *'none'*: no activation operation is performed.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'bias_filler': See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
List of values: *'xavier'*, *'msra'*, *'const'*.
Default: 'xavier'

'bias_filler_const_val': Constant value if *'bias_filler'* has been set to *'const'*.
Default: 0

'bias_filler_variance_norm': See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
List of values: *'norm_out'*, *'norm_in'*, *'norm_average'*, or constant value (in combination with *'bias_filler'* set to *'msra'*).
Default: 'norm_out'

'bias_term': If set to *'false'*, the created batch normalization layer has no bias term.
Default: 'true'

'is_inference_output': Determines whether `apply_dl_model` will include the output of this layer in the dictionary `DLResultBatch` even without specifying this layer in `Outputs` (*'true'*) or not (*'false'*).
Default: 'false'

'learning_rate_multiplier': Multiplier for the learning rate for this layer that is used during training. If *'learning_rate_multiplier'* is set to *0*, the layer is skipped during training.
Default: 1.0

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

'upper_bound': Float value defining an upper bound for a rectified linear unit. If the activation layer is part of a model, which has been created using `create_dl_model`, the upper bound can be unset. To do so, use `set_dl_model_layer_param` and set an empty tuple for *'upper_bound'*.
Default: []

'weight_filler': See `create_dl_layer_convolution` for a detailed explanation of this parameter and its values.
List of values: *'xavier'*, *'msra'*, *'const'*.
Default: 'xavier'

'weight_filler_const_val': Constant value if *'weight_filler'* has been set to *'const'*. See `create_dl_layer_convolution` for a detailed explanation of this parameter and its values.
Default: 0.5

'weight_filler_variance_norm': See `create_dl_layer_convolution` for a detailed explanation of this parameter and its values.
List of values: *'norm_in'*, *'norm_out'*, *'norm_average'*, or constant value (in combination with *'weight_filler'* set to *'msra'*).
Default: 'norm_in'

Certain parameters of layers created using this operator `create_dl_layer_batch_normalization` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	set	get
'activation_mode' (`Activation`)	–	x
'epsilon' (`Epsilon`)	–	x
'input_layer' (`DLLayerInput`)	–	x
'momentum' (`Momentum`)	x	x
'name' (`LayerName`)	x	x
'output_layer' (`DLLayerBatchNorm`)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'bias_filler'	x	x
'bias_filler_const_val'	x	x
'bias_filler_variance_norm'	x	x
'bias_term'	x	x
'is_inference_output'	x	x
'learning_rate_multiplier'	x	x
'learning_rate_multiplier_bias'	x	x
'num_trainable_params'	–	x
'upper_bound'	x	x
'weight_filler'	x	x
'weight_filler_const_val'	x	x
'weight_filler_variance_norm'	x	x

Parameter

▷ **DLLayerInput** (input_control) . . . . . dl_layer ⇝ *handle*
Feeding layer.

▷ **LayerName** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **Momentum** (input_control) . . . . . string ⇝ *string* / real
Momentum.
Defaultwert: 'auto'
Werteliste: `Momentum` ∈ {'auto', 'freeze', 0.9, 0.99, 0.999}

▷ **Epsilon** (input_control) . . . . . number ⇝ *real*
Variance offset.
Defaultwert: 0.0001

▷ **Activation** (input_control) . . . . . string ⇝ *string*
Optional activation function.
Defaultwert: 'none'
Werteliste: `Activation` ∈ {'none', 'relu'}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'bias_filler', 'bias_filler_variance_norm', 'bias_filler_const_val', 'bias_term', 'is_inference_output', 'learning_rate_multiplier', 'learning_rate_multiplier_bias', 'upper_bound', 'weight_filler', 'weight_filler_variance_norm', 'weight_filler_const_val'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'xavier', 'msra', 'const', 'nearest_neighbor', 'bilinear', 'norm_in', 'norm_out', 'norm_average', 'true', 'false', 1.0, 0.9, 0.0}

▷ **DLLayerBatchNorm** (output_control) . . . . . dl_layer ⇝ *handle*
Batch normalization layer.

Beispiel

```
create_dl_layer_input ('input', [224,224,3], [], [], DLLayerInput)
* In practice, one typically sets ['bias_term'], ['false'] for a convolution
* that is directly followed by a batch normalization layer.
create_dl_layer_convolution (DLLayerInput, 'conv1', 3, 1, 1, 64, 1, 'none', \
                             'none', ['bias_term'], ['false'], \
                             DLLayerConvolution)
create_dl_layer_batch_normalization (DLLayerConvolution, 'bn1', 'auto', \
                                     0.0001, 'none', [], [], DLLayerBatchNorm)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_dl_layer_convolution

Nachfolger

create_dl_layer_activation, create_dl_layer_convolution

Literatur

Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,"Proceedings of the 32nd International Conference on Machine Learning, (ICML) 2015, Lille, France, 6-11 July 2015, pp. 448–456

Modul

Deep Learning Training

create_dl_layer_class_id_conversion (: : DLLayerInput, LayerName, ConversionMode, GenParamName, GenParamValue : DLLayerClassIdConversion)

Create a class ID conversion layer.

The operator create_dl_layer_class_id_conversion creates a class ID conversion layer whose handle is returned in DLLayerClassIdConversion. The layer converts between the IDs used internally by the network and the target / output class IDs.

The network internally uses consecutive integer values starting from 0 as IDs (the number of values depends on the model type). In case the target / output class IDs differ from the internal IDs, this layer can be used to convert between them. The target / output class IDs are stored in the model parameter *'class_ids'* (see get_dl_model_param for more information on this parameter). If no *'class_ids'* are set, this layer copies the input to the output.

The parameter ConversionMode specifies the conversion direction and accepts the following values:

- *'from_class_id'*: Convert class IDs into internal IDs. This mode is typically used after a target input layer.
- *'to_class_id'*: Convert internal IDs into class IDs. This mode is typically used after an inference output layer.

The parameter DLLayerInput determines the feeding input layer.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_class_id_conversion can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'input_layer' (DLLayerInput)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerClassIdConversion)	–	x
'shape'	–	x
'to_class_id' (ConversionMode)	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

- ▷ **DLLayerInput** (input_control) ..... dl_layer ⇝ *handle*
  Feeding layer.
- ▷ **LayerName** (input_control) ..... string ⇝ *string*
  Name of the layer.
- ▷ **ConversionMode** (input_control) ..... string ⇝ *string*
  Direction of the class ID conversion.
  Defaultwert: 'from_class_id'
  Werteliste: ConversionMode ∈ {'from_class_id', 'to_class_id'}
- ▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
  Generic input parameter names.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'is_inference_output'}
- ▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
  Generic input parameter values.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {'true', 'false'}
- ▷ **DLLayerClassIdConversion** (output_control) ..... dl_layer ⇝ *handle*
  Class IDs conversion layer.

Beispiel

```
* Example demonstrating the usage of
* create_dl_layer_class_id_conversion.
*
dev_update_off ()
set_system ('seed_rand', 42)
*
* Create simple segmentation model.
NumClasses := 3
InputShape := [32, 32, 3]
*
* Input feeding layers.
create_dl_layer_input ('image', InputShape, [], [], DLLayerInput)
create_dl_layer_input ('target', [InputShape[0],InputShape[1],1], [], [], \
                       DLLayerTarget)
create_dl_layer_class_id_conversion (DLLayerTarget, 'target_internal', \
                                     'from_class_id', [], [], \
                                     DLLayerTargetInternal)
* Feature extraction layers.
create_dl_layer_convolution (DLLayerInput, 'conv1', 3, 1, 1, 32, 1, \
                             'half_kernel_size', 'relu', [], [], \
                             DLLayerConv1)
create_dl_layer_convolution (DLLayerConv1, 'conv2', 3, 1, 1, 32, 1, \
```

```
                              'half_kernel_size', 'relu', [], [], \
                              DLLayerConv2)
* Output generation layers.
create_dl_layer_convolution (DLLayerConv2, 'conv_final', 1, 1, 1, \
                              NumClasses, 1, 'none', 'none', [], [], \
                              DLLayerConvFinal)
create_dl_layer_softmax (DLLayerConvFinal, 'softmax', [], [], \
                         DLLayerSoftMax)
create_dl_layer_depth_max (DLLayerSoftMax, 'output_internal', \
                           'argmax', [], [], DLLayerOutputInternal, _)
create_dl_layer_class_id_conversion (DLLayerOutputInternal, 'output', \
                                     'to_class_id', [], [], DLLayerOutput)
* Loss layer.
create_dl_layer_loss_cross_entropy (DLLayerSoftMax, DLLayerTargetInternal, \
                                    [], 'loss', 1.0, [], [], DLLayerLoss)
*
* Create the model.
create_dl_model ([DLLayerOutput, DLLayerLoss], DLModelHandle)
set_dl_model_param (DLModelHandle, 'type', 'segmentation')
set_dl_model_param (DLModelHandle, 'runtime', 'cpu')
*
* Test model on dummy example data.
read_image (Image, 'claudia')
zoom_image_size (Image, Image, InputShape[0], InputShape[1], 'constant')
convert_image_type (Image, Image, 'real')
*
* Fill target image with specific target class IDs.
ClassIDs := [42, 17, 5]
gen_image_const (Target, 'real', InputShape[0], InputShape[1])
paint_region (Target, Target, Target, ClassIDs[0], 'fill')
gen_rectangle1 (RectClass1, 1, 3, 16, 27)
paint_region (RectClass1, Target, Target, ClassIDs[1], 'fill')
gen_rectangle1 (RectClass2, 19, 1, 30, 30)
paint_region (RectClass2, Target, Target, ClassIDs[2], 'fill')
*
* Set class IDs in the model.
set_dl_model_param (DLModelHandle, 'class_ids', ClassIDs)
*
* Create test sample.
create_dict (DLSample)
set_dict_object (Image, DLSample, 'image')
set_dict_object (Target, DLSample, 'target')
*
* Train model for a few iterations. Note that training would not
* work without the first class ID conversion layer 'target_internal'.
for Idx := 1 to 100 by 1
    train_dl_model_batch (DLModelHandle, DLSample, DLTrainResult)
endfor
*
* Apply model on test image. With the second class ID conversion
* layer 'output', the image now contains values according to the
* target IDs in segmentation_image.
apply_dl_model (DLModelHandle, DLSample, [], DLApplyResult)
get_dict_object (SegmentationImage, DLApplyResult, 'output')
dev_display (SegmentationImage)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_concat (: : DLLayerInputs, LayerName, Axis, GenParamName, GenParamValue : DLLayerConcat)

Create a concatenation layer.

The operator create_dl_layer_concat creates a concatenation layer whose handle is returned in DLLayerConcat.

The parameter DLLayerInputs determines the feeding input layers.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

A concatenation layer concatenates the data tensors of the input layers in DLLayerInputs and returns a single data tensor DLLayerConcat. The parameter Axis specifies along which dimension the inputs should be concatenated. The supported options for Axis are:

'batch': Concatenation is applied along the batch-dimension.
Example: if you concatenate two inputs *A* and *B* of shape (b, d, h, w) = (2, 1, 1, 1), where *A* = [*A0*, *A1*] and *B* = [*B0*, *B1*], you obtain the output [*A0*, *A1*, *B0*, *B1*] with shape (4, 1, 1, 1).

'batch_interleaved': Concatenation is applied along the depth-dimension, but the output is reshaped as if the data was concatenated along the batch-dimension. For this axis, all inputs need to have exactly the same shape.
Note that when the input batch_size is *1*, the axes *'batch'* and *'batch_interleaved'* are identical.
Example: if you concatenate two inputs *A* and *B* of shape (b, d, h, w) = (2, 1, 1, 1), where *A* = [*A0*, *A1*] and *B* = [*B0*, *B1*], you obtain the output [*A0*, *A1*, *B0*, *B1*] with shape (4, 1, 1, 1).

'depth': Concatenation is applied along the depth-dimension.
Example: if you concatenate two inputs *A* and *B* of shape (b, d, h, w) = (2, 1, 1, 1), where *A* = [*A0*, *A1*] and *B* = [*B0*, *B1*], you obtain the output [*A0*, *A1*, *B0*, *B1*] with shape (2, 2, 1, 1).

'height': Concatenation is applied along the height-dimension.

'width': Concatenation is applied along the width-dimension.

Note that all non-concatenated dimensions must be equal for all input data tensors.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_concat can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'input_layer' (DLLayerInputs)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerConcat)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **DLLayerInputs** (input_control) . . . . . dl_layer(-array) ⇝ *handle*
Feeding input layers.

▷ **LayerName** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **Axis** (input_control) . . . . . string ⇝ *string*
Dimension along which the input layers are concatenated.
Defaultwert: 'depth'
Werteliste: `Axis` ∈ {'batch', 'batch_interleaved', 'depth', 'height', 'width'}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false'}

▷ **DLLayerConcat** (output_control) . . . . . dl_layer ⇝ *handle*
Concatenation layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

```
create_dl_layer_convolution ( : : DLLayerInput, LayerName,
    KernelSize, Dilation, Stride, NumKernel, Groups, Padding,
    Activation, GenParamName, GenParamValue : DLLayerConvolution )
```

Create a convolutional layer.

The operator `create_dl_layer_convolution` creates a convolutional layer with `NumKernel` kernels in `Groups` filter groups whose handle is returned in `DLLayerConvolution`.

The parameter `DLLayerInput` determines the feeding input layer and expects the layer handle as value.

The parameter `LayerName` sets an individual layer name. Note that if creating a model using `create_dl_model` each layer of the created network must have an unique name.

The parameter `KernelSize` specifies the filter kernel in the dimensions `width` and `height`.

The parameter `Dilation` specifies the factor of filter dilation in the dimensions `width` and `height`.

The parameter `Stride` specifies how the filter is shifted.

The values for `KernelSize`, `Dilation`, and `Stride` can be set as

- a single value which is used for both dimensions
- a tuple `[width, height]` and `[column, row]`, respectively.

The parameter Groups specifies the number of filter groups.

The parameter NumKernel specifies the number of filter kernels. NumKernel must be a multiple of groups.

The parameter Padding determines the padding, thus how many pixels with value 0 are appended on the boarder of the processed input image. Supported values are:

- *'half_kernel_size'*: The number of appended pixels depends on the specifies KernelSize. More precisely they are calculated as $\lfloor \texttt{KernelSize}/2 \rfloor$, where for the padding on the left / right bound the value of KernelSize in dimension width is regarded and for the padding on the upper / lower bound the value of KernelSize in height.
- *'none'*: No pixels are appended.
- Number of pixels: Specify the number of pixels appended on each border. To do so, the following tuple lengths are supported:
  - Single number: Padding in all four directions left/right/top/bottom.
  - Two numbers: Padding in left/right and top/bottom: [l/r, t,b].
  - Four numbers: Padding on left, right, top, bottom side: [l,r,t,b].
    Restriction: 'runtime' 'gpu' does not support asymmetric padding, i.e., the padding values for the left and right side must be equal, as well as the padding values for the top and bottom side.

  Restriction: The integer padding values must be smaller than the value set for KernelSize in the corresponding dimension.

The output dimensions of the convolution layer are given by

$$output_dim = \left\lfloor \frac{input_dim + 2(padding_begin + padding_end)}{stride} - \frac{kernel_size + (kernel_size - 1)(dilation - 1)}{stride} \right\rfloor + 1$$

Thereby we use the following values: $output_dim$: output width, $input_dim$: input width, $kernel_size$: corresponding entry in KernelSize, $dilation$: corresponding entry in Dilation, $stride$: corresponding entry in Stride, $padding_begin$: number of pixels added to the left/top of the input image, and $padding_end$: number of pixels added to the right/bottom of the input image.

The parameter Activation determines whether an activation is performed after the convolution in order to optimize the runtime performance. The following values are supported:

- *'relu'*: perform a ReLU activation after the convolution.
  It is possible to specify an upper bound to the ReLU operation (see create_dl_layer_activation) via the generic parameter *'upper_bound'*.
- *'none'*: no activation operation is performed.

We refer to the „Solution Guide on Classification" for more general information about the convolution layer and the reference given below for more detailed information about the arithmetic of the layer.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'bias_filler': See *'weight_filler'* for an explanation of the values.
List of values: *'xavier'*, *'msra'*, *'const'*.
Default: 'xavier'

'bias_filler_variance_norm': See *'weight_filler_variance_norm'* for an explanation of the values.
List of values: *'norm_in'*, *'norm_out'*, *'norm_average'*.
Default: 'norm_out'

'bias_filler_const_val': Specifies the constant weight initialization value if *'bias_filler'* has been set to *'const'*. Ignored for other values of *'bias_filler'*.
Default: 0

'bias_term': Determines whether if the created convolutional layer has a bias term (*'true'*) or not (*'false'*).
Default: 'true'

'is_inference_output': Determines whether `apply_dl_model` will include the output of this layer in the dictionary `DLResultBatch` even without specifying this layer in `Outputs` (*'true'*) or not (*'false'*).
Default: 'false'

'learning_rate_multiplier': Multiplier for the learning rate for this layer that is used during training. If *'learning_rate_multiplier'* is set to zero, the layer is skipped during training.
Default: 1.0

'learning_rate_multiplier_bias': Multiplier for the learning rate of the bias term. The total bias learning rate is the product of *'learning_rate_multiplier_bias'* and *'learning_rate_multiplier'*.
Default: 1.0

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

'upper_bound': Float value, which defines the upper bound for ReLU. To unset the upper bound, set *'upper_bound'* to an empty tuple.
Default: []

'weight_filler': This parameter defines the mode how the weights are initialized. The following values are supported:

- *'const'*: The weights are filled with constant values
- *'msra'*: The weights are drawn from a Gaussian distribution
- *'xavier'*: The weights are drawn from a uniform distribution.

Default: 'xavier'

'weight_filler_const_val': Specifies the constant weight initialization value if *'weight_filler'* has been set to *'const'*. Ignored for other values of *'weight_filler'*.
Default: 0.5

'weight_filler_variance_norm': This parameter determines the value range for *'weight_filler'*. The following values are supported:

- *'norm_average'*: the values are based on the average of the input and output size
- *'norm_in'*: the values are based on the input size
- *'norm_out'*: the values are based on the output size.

Default: 'norm_in'

Certain parameters of layers created using `create_dl_layer_convolution` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	**set**	**get**
'activation_mode' (`Activation`)	–	x
'dilation' (`Dilation`)	–	x
'groups' (`Groups`)	–	x
'input_depth'	–	x
'input_layer' (`DLLayerInput`)	–	x
'kernel_size' (`KernelSize`)	–	x
'name' (`LayerName`)	x	x
'num_kernels' (`NumKernel`)	–	x
'output_layer' (`DLLayerConvolution`)	–	x
'padding' (`Padding`)	–	x
'padding_type' (`Padding`)	–	x
'shape'	–	x
'stride' (`Stride`)	–	x
'type'	–	x

Generic Layer Parameters	set	get
'bias_filler'	x	x
'bias_filler_const_val'	x	x
'bias_filler_variance_norm'	x	x
'bias_term'	x	x
'is_inference_output'	x	x
'learning_rate_multiplier'	x	x
'learning_rate_multiplier_bias'	x	x
'num_trainable_params'	–	x
'upper_bound'	x	x
'weight_filler'	x	x
'weight_filler_const_val'	x	x
'weight_filler_variance_norm'	x	x

Parameter

▷ **DLLayerInput** (input_control) . . . . . dl_layer ⇝ *handle*
Feeding layer.

▷ **LayerName** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **KernelSize** (input_control) . . . . . number(-array) ⇝ *integer*
Width and height of the filter kernels.
Defaultwert: 3

▷ **Dilation** (input_control) . . . . . number(-array) ⇝ *integer*
Amount of filter dilation for width and height.
Defaultwert: 1

▷ **Stride** (input_control) . . . . . number(-array) ⇝ *integer*
Amount of filter shift in width and height direction.
Defaultwert: 1

▷ **NumKernel** (input_control) . . . . . number ⇝ *integer*
Number of filter kernels.
Defaultwert: 64

▷ **Groups** (input_control) . . . . . number ⇝ *integer*
Number of filter groups.
Defaultwert: 1

▷ **Padding** (input_control) . . . . . number(-array) ⇝ *string* / integer
Padding type or specific padding size.
Defaultwert: 'none'
Werteliste: `Padding` ∈ {'none', 'half_kernel_size', [all], [width,height], [left,right,top,bottom]}
Wertevorschläge: `Padding` ∈ {'none', 'half_kernel_size'}

▷ **Activation** (input_control) . . . . . number ⇝ *string*
Enable optional ReLU or sigmoid activations.
Defaultwert: 'none'
Werteliste: `Activation` ∈ {'none', 'relu', 'sigmoid'}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'weight_filler', 'weight_filler_variance_norm', 'weight_filler_const_val', 'bias_filler', 'bias_filler_variance_norm', 'bias_filler_const_val', 'bias_term', 'is_inference_output', 'learning_rate_multiplier', 'learning_rate_multiplier_bias', 'upper_bound'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'xavier', 'msra', 'const', 'nearest_neighbor', 'bilinear', 'norm_in', 'norm_out', 'norm_average', 'true', 'false', 1.0, 0.9, 0.0}

▷ **DLLayerConvolution** (output_control) . . . . . dl_layer ⇝ *handle*
Convolutional layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Literatur

V. Dumoulin, F. Visin: "A guide to convolution arithmetic for deep learning", 2018, http://arxiv.org/abs/1603.07285

Modul

Deep Learning Training

create_dl_layer_dense (: : DLLayerInput, LayerName, NumOut, GenParamName, GenParamValue : DLLayerDense)

Create a dense layer.

The operator create_dl_layer_dense creates a dense or fully connected layer (sometimes also called gemm) with NumOut output neurons whose handle is returned in DLLayerDense.

The parameter DLLayerInput determines the feeding input layer and expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'bias_filler': See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
List of values: *'xavier'*, *'msra'*, *'const'*.
Default: 'xavier'

'bias_filler_const_val': Constant value if *'bias_filler'* has been set to *'const'*.
Default: 0

'bias_filler_variance_norm': See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
List of values: *'norm_out'*, *'norm_in'*, *'norm_average'*, or constant value (in combination with *'bias_filler'* set to *'msra'*).
Default: 'norm_out'

'bias_term': if set to *'false'*, the created dense layer has no bias term.
Default: 'true'

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'learning_rate_multiplier': Multiplier for the learning rate for this layer that is used during training. If *'learning_rate_multiplier'* is set to zero, the layer is skipped during training.
Default: 1.0

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

'weight_filler': See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
List of values: *'xavier'*, *'msra'*, *'const'*.
Default: 'xavier'

'weight_filler_const_val': Constant value if *'weight_filler'* has been set to *'const'*. See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
Default: 0.5

'weight_filler_variance_norm': See `create_dl_layer_convolution` for a detailed explanation of this parameter and its values.

List of values: *'norm_in'*, *'norm_out'*, *'norm_average'*, or constant value (in combination with *'bias_filler'* set to *'msra'*).

Default: 'norm_in'

Certain parameters of layers created using `create_dl_layer_dense` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	set	get
'input_layer' (`DLLayerInput`)	–	x
'name' (`LayerName`)	x	x
'neurons_in'	–	x
'neurons_out' (`NumOut`)	–	x
'output_layer' (`DLLayerDense`)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'bias_filler'	x	x
'bias_filler_const_val'	x	x
'bias_filler_variance_norm'	x	x
'bias_term'	x	x
'is_inference_output'	x	x
'learning_rate_multiplier'	x	x
'learning_rate_multiplier_bias'	x	x
'num_trainable_params'	–	x
'weight_filler'	x	x
'weight_filler_const_val'	x	x
'weight_filler_variance_norm'	x	x

Parameter

▷ **`DLLayerInput`** (input_control) ..... dl_layer ⇝ *handle*
Feeding layer.

▷ **`LayerName`** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **`NumOut`** (input_control) ..... number ⇝ *integer*
Number of output neurons.
Defaultwert: 100

▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'weight_filler', 'weight_filler_variance_norm', 'weight_filler_const_val', 'bias_filler', 'bias_filler_variance_norm', 'bias_filler_const_val', 'bias_term', 'is_inference_output', 'learning_rate_multiplier', 'learning_rate_multiplier_bias'}

▷ **`GenParamValue`** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'xavier', 'msra', 'const', 'nearest_neighbor', 'bilinear', 'norm_in', 'norm_out', 'norm_average', 'true', 'false', 1.0, 0.9, 0.0}

▷ **DLLayerDense** (output_control) . . . . . dl_layer ⇝ *handle*
Dense layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_depth_max (: : DLLayerInput, LayerName, DepthMaxMode, GenParamName, GenParamValue : DLLayerDepthMaxArg, DLLayerDepthMaxValue)

Create a depth max layer.

The operator create_dl_layer_depth_max creates a depth max layer.

The parameter DLLayerInput determines the feeding input layer and expects the layer handle as value.

There are two possible output layers:

- DLLayerDepthMaxArg: Handle to a depth max layer with mode *'argmax'*.
- DLLayerDepthMaxValue: Handle to a depth max layer with mode *'value'*.

If they are returned depends on the mode DepthMaxMode. Note, these parameters only need to be set in case such an output layer is requested (see DepthMaxMode).

The parameter LayerName defines the name of the output layer(s) depending on DepthMaxMode:

- *'argmax'*: name of DLLayerDepthMaxArg.
- *'value'*: name of DLLayerDepthMaxValue.
- *'argmax_and_value'*: name of DLLayerDepthMaxArg, while the layer DLLayerDepthMaxValue receives the same name with the suffix string *'_value'* appended to it.

Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The mode DepthMaxMode indicates which depth max value is actually returned as output. The following values are supported:

'argmax': *newline* The depth index of the maximal value is returned in DLLayerDepthMaxArg.

'value': *newline* The maximal value itself is returned in DLLayerDepthMaxValue.

'argmax_and_value': *newline* Both are returned, the depth index of the maximal value in DLLayerDepthMaxArg, and the maximal value itself in the output layer DLLayerDepthMaxValue.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_depth_max can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'input_layer' (DLLayerInput)	–	x
'mode' (DepthMaxMode)	x	x
'name' (LayerName)	x	x
'output_layer' (DLLayerDepthMaxArg or/and DLLayerDepthMaxValue)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	–	x
'num_trainable_params'	–	x

Parameter

- ▷ **DLLayerInput** (input_control) . . . . . dl_layer ⇝ *handle*
  Feeding layer.
- ▷ **LayerName** (input_control) . . . . . string ⇝ *string*
  Name of the layer.
- ▷ **DepthMaxMode** (input_control) . . . . . string ⇝ *string*
  Mode to indicate type of return value.
  Defaultwert: 'argmax'
  Werteliste: DepthMaxMode ∈ {'argmax', 'value', 'argmax_and_value'}
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Generic input parameter names.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'is_inference_output'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
  Generic input parameter values.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {'true', 'false'}
- ▷ **DLLayerDepthMaxArg** (output_control) . . . . . dl_layer(-array) ⇝ *handle*
  Optional, depth max layer with mode *'argmax'*.
- ▷ **DLLayerDepthMaxValue** (output_control) . . . . . dl_layer(-array) ⇝ *handle*
  Optional, depth max layer with mode *'value'*.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_dropout (: : DLLayerInput, LayerName, Probability, GenParamName, GenParamValue : DLLayerDropOut)

Create a DropOut layer.

The operator create_dl_layer_dropout creates a DropOut layer with probability Probability and returns the handle DLLayerDropOut.

The parameter DLLayerInput determines the feeding input layer and expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

During training, activations within DLLayerInput are set to zero with probability Probability. All other activations are rescaled with (1 - Probability).

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_dropout can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'input_layer' (DLLayerInput)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerDropOut)	–	x
'probability' (Probability)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **DLLayerInput** (input_control) . . . . . dl_layer ⇝ *handle*
Feeding layer.

▷ **LayerName** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **Probability** (input_control) . . . . . number ⇝ *real*
Probability.
Defaultwert: 0.5

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: GenParamName ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'true', 'false'}

▷ **DLLayerDropOut** (output_control) . . . . . dl_layer ⇝ *handle*
DropOut layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_elementwise (: : DLLayerInputs, LayerName, Operation, Coefficients, GenParamName, GenParamValue : DLLayerElementWise)

Create an elementwise layer.

The operator create_dl_layer_elementwise creates an element-wise layer whose handle is returned in DLLayerElementWise.

An elementwise layer applies a certain operation to every data tensor of the input layers handles and to each element of the data tensor. As a consequence, all input data tensors should be of the same shape and the output tensor has the same shape as the first input tensor.

Depending on Operation, the layer supports implicit broadcasting. I.e., if one of the shape dimensions (batch_size, depth, height, width) of the second or any of the following inputs is *1*, the values are implicitly duplicated along that dimension to match the shape of the first input. See also the restrictions below.

The parameter DLLayerInputs determines the feeding input layers.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The parameter Operation specifies the operation that is applied. The supported operations are:

- *'maximum'*: Element-wise maximum. Broadcasting is fully supported for this operation.
- *'minimum'*: Element-wise minimum. Broadcasting is fully supported for this operation.
- *'product'*: Element-wise product. Broadcasting is supported for this operation, but all inputs following the second input must have the same shape as the second input.
- *'sum'*: Element-wise summation. Broadcasting is not supported for this operation.

The optional parameter Coefficients determines a weighting coefficient for every input tensor. The number of values in Coefficients must match the number of feeding layers in DLLayerInputs. Set Coefficients equal to *[]* if no coefficients shall be used in the element-wise operation.

Restriction: No coefficients can be set for Operation = *'product'*.

Example: for Operation = *'sum'*, the i-th element of the output data tensor is given by

$$output[i] = \sum_{n=0}^{N-1} \texttt{Coefficients}[n] \cdot \texttt{DLLayerInputs}_n[i],$$

where N is the number of input data tensors.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_elementwise can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'coefficients' (Coefficients)	–	x
'input_layer' (DLLayerInputs)	–	x
'name' (LayerName)	x	x
'operation' (Operation)	–	x
'output_layer' (DLLayerElementWise)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **DLLayerInputs** (input_control) ..... dl_layer(-array) ⇝ *handle*
Feeding input layers.

▷ **LayerName** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **Operation** (input_control) ..... string ⇝ *string*
Element-wise operations.
Defaultwert: 'sum'
Werteliste: Operation ∈ {'maximum', 'minimum', 'product', 'sum'}

▷ **Coefficients** (input_control) ..... number(-array) ⇝ *real*
Optional input tensor coefficients.
Defaultwert: []

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: GenParamName ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'true', 'false'}

▷ **DLLayerElementWise** (output_control) ..... dl_layer ⇝ *handle*
Elementwise layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_input (: : LayerName, Shape, GenParamName, GenParamValue : DLLayerInput)

Create an input layer.

The operator create_dl_layer_input creates an input layer with spatial dimensions given by Shape whose handle is returned in DLLayerInput.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

Shape must be a tuple of length three, containing width, height and depth of the input. The Tuple values must be given as integer. The batch size has to be set later with set_dl_model_param, once the model has been created by create_dl_model.

When the created model is applied using e.g., apply_dl_model or train_dl_model_batch, it must be possible to map an input with its corresponding input layer. Operators applying a model expect a feeding dictionary DLSample, see Deep Learning / Modell. The mentioned mapping is done using dictionary entries, where the key matches the input layer name. Thus, for an input of this layer a sample dictionary will need an entry with the key LayerName (except if the *'input_type'* is set to *'constant'*, see below).

The parameter Shape defines the shape of the input values (the values given in the feeding dictionary DLSample):

- for an input image the layer Shape defines the image size. Images shall be given with type real (for information on image types see Bild).
- for an input tuple its length will need to match the product of the individual values in Shape, i.e., width × height × depth. Tuple values are distributed along the column- (width), row- (height), and depth-axes in this order.

  Tuple values can be given either as integer or real.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'allow_smaller_tuple': For tuple inputs, setting *'allow_smaller_tuple'* to *'true'* allows to have an input tuple with less values than the total dimension given by Shape. E.g., this can be the case if an input corresponds to the number of objects within one image and the number of objects changes from image to image. If fewer than the maximum number of values given by the total dimension of Shape are present, the remaining values are set to zero.

Shape should be set such that it fits the maximum expected length. For the example above this would be the maximum number of objects within one image present in the whole dataset.

Default: 'false'.

'const_val': Constant output value, if the *'input_type'* is set to *'constant'* (see below).

Restriction: Only an integer or float is settable. This value is only settable or gettable if *'input_type'* is set to *'constant'*.

Default: 0.0.

'input_type': Defines the type of input that is expected. The following values are possible:

'region_to_bin': The layer expects an array of regions as input and internally converts it to a binary image where each region is encoded in one depth channel. Regions reaching out of the given dimensions are clipped to the width and height given by Shape. The maximum number of regions is defined by the depth of Shape. If fewer than the maximum number of regions are given, the output is filled up with empty (zero) images. For example, this can be the case if the regions are corresponding to objects within an image and the number of objects changes from image to image.

'constant': The layer does not expect any key value pair for this input layer inside the feeding dictionary. Instead all entries within the output of this layer are filled with the value given by *'const_val'*.

Default: 'default'.

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).

Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using create_dl_layer_input can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'input_layer'	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerInput)	–	x
'shape' (Shape)	–	x
'type'	–	x

Generic Layer Parameters	set	get
'allow_smaller_tuple'	–	x
'const_val'	(x)	(x)
'input_type'	–	x
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **LayerName** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **Shape** (input_control) . . . . . number-array ⇝ *integer*
Dimensions of the input (width, height, depth).
Defaultwert: [224,224,3]

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: GenParamName ∈ {'allow_smaller_tuple', 'const_val', 'input_type', 'is_inference_output'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0.0, 'constant', 'default', 'false', 'region_to_bin', 'true'}

▷ **DLLayerInput** (output_control) . . . . . dl_layer ⇝ *handle*
Input layer.

Beispiel

```
* Create a model for summation.
create_dl_layer_input ('input_a', [2, 3, 4], [], [], DLLayerInputA)
create_dl_layer_input ('input_b', [2, 3, 4], [], [], DLLayerInputB)
create_dl_layer_elementwise ([DLLayerInputA, DLLayerInputB], 'sum', \
                             'sum', [], [], [], DLLayerElementWise)
create_dl_model (DLLayerElementWise, DLModel)
set_dl_model_param (DLModel, 'runtime', 'cpu')
*
* Add 'input_a' as an inference model output.
set_dl_model_layer_param (DLModel, 'input_a', 'is_inference_output', 'true')
*
* Feed input data as tuple (a) or image (b).
create_dict (Sample)
set_dict_tuple (Sample, 'input_a', [1:(2*3*4)])
gen_empty_obj (InputB)
for I := 1 to 4 by 1
    gen_image_const (Channel, 'real', 2, 3)
    get_region_points (Channel, Rows, Cols)
    set_grayval (Channel, Rows, Cols, gen_tuple_const(|Rows|, I))
    append_channel (InputB, Channel, InputB)
endfor
set_dict_object (InputB, Sample, 'input_b')
*
```

```
* Apply the model for summation and get results.
set_dl_model_param (DLModel, 'batch_size', 2)
apply_dl_model (DLModel, [Sample,Sample], [], Result)
get_dict_object (Sum, Result[0], 'sum')
get_dict_object (TupleInputA, Result[1], 'input_a')
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

create_dl_layer_activation, create_dl_layer_batch_normalization, create_dl_layer_class_id_conversion, create_dl_layer_class_id_conversion, create_dl_layer_concat, create_dl_layer_convolution, create_dl_layer_dense, create_dl_layer_depth_max, create_dl_layer_dropout, create_dl_layer_elementwise, create_dl_layer_loss_cross_entropy, create_dl_layer_loss_ctc, create_dl_layer_loss_distance, create_dl_layer_loss_focal, create_dl_layer_loss_huber, create_dl_layer_lrn, create_dl_layer_pooling, create_dl_layer_reshape, create_dl_layer_softmax, create_dl_layer_transposed_convolution, create_dl_layer_zoom_factor, create_dl_layer_zoom_size, create_dl_layer_zoom_to_layer_size

Modul

Deep Learning Training

create_dl_layer_loss_cross_entropy (: : DLLayerInput, DLLayerTarget, DLLayerWeights, LayerName, LossWeight, GenParamName, GenParamValue : DLLayerLossCrossEntropy)

Create a cross entropy loss layer.

The operator create_dl_layer_loss_cross_entropy creates a cross entropy loss layer whose handle is returned in DLLayerLossCrossEntropy. This layer computes the two dimensional cross entropy loss on the input (provided by DLLayerInput) given the corresponding target (provided by DLLayerTarget) and weight (provided by DLLayerWeights).

Cross entropy is commonly used to measure the similarity between two vectors.

Example: Let us see an illustrative example, where we have a pixel-level classification problem with three classes.

The input vector for a single pixel is $\mathbf{x} = [0.7, 0.1, 0.2]$ (e.g., the output of a softmax layer) which means that the predicted value (e.g., probability) is 0.7 for the class at index 0, 0.1 for the class at index 1 and 0.2 for the class at index 2.

The target vector is $\mathbf{t} = [1.0, 0.0, 0.0]$ with a probability of 1.0 for the actual class and 0.0 else. Entropy is calculated by the dot product of these two vectors. Since the target vector has only one non-zero entry, it can be given by the index of the actual class instead of a vector, in this case $t = 0$.

The cross entropy is then simply the value of the input vector at the target class index, hence $\mathbf{x}[t] = 0.7$. Using this simplification, the cross entropy loss function over an input image can be defined by

$$L_{cross_entropy}(\mathbf{x}, \mathbf{t}, \mathbf{w}) \quad := \quad -\frac{1}{W} \sum_{i=0}^{N-1} w_i \cdot \mathbf{x}_i[t_i],$$

where the input $\mathbf{x}$ consists of one prediction vector $\mathbf{x}_i$ for each pixel, the target $\mathbf{t}$ and weight $\mathbf{w}$ consist of one value t_i and w_i for each input pixel, N is the number of pixels and $W = \sum_{i=0}^{N-1} w_i$ is the sum over all weights.

Hence, this layer expects multiple incoming layers:

- `DLLayerInput`: The feeding input layer which specifies the prediction (e.g., a softmax layer, commonly with logarithmized results). It expects the layer handle as value.
- `DLLayerTarget`: The feeding input layer which specifies the target sequences (originating from the ground truth information). It expects the layer handle as value.
- `DLLayerWeights`: The feeding input layer which specifies the weight sequences. It expects the layer handle as value.

The parameter `LayerName` sets an individual layer name. Note that if creating a model using `create_dl_model` each layer of the created network must have an unique name.

The parameter `LossWeight` determines the scalar weight factor with which the loss, calculated in this layer, is multiplied. This parameter can be used to specify the contribution of the cross entropy loss to the overall network loss in case multiple loss layers are used.

The following generic parameters `GenParamName` and the corresponding values `GenParamValue` are supported:

'is_inference_output': Determines whether `apply_dl_model` will include the output of this layer in the dictionary `DLResultBatch` even without specifying this layer in `Outputs` (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator `create_dl_layer_loss_cross_entropy` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	set	get
'input_layer' (`DLLayerInput`, `DLLayerTarget`, or/and `DLLayerWeights`)	–	x
'loss_weight' (`LossWeight`)	x	x
'name' (`LayerName`)	x	x
'output_layer' (`DLLayerLossCrossEntropy`)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **`DLLayerInput`** (input_control) ..... dl_layer ⇝ *handle*
Input layer (usually a softmax).

▷ **`DLLayerTarget`** (input_control) ..... dl_layer ⇝ *handle*
Target layer.

▷ **`DLLayerWeights`** (input_control) ..... dl_layer ⇝ *handle*
Weights layer.

▷ **`LayerName`** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **LossWeight** (input_control) . . . . . number ⇝ *real*
Overall loss weight if there are multiple losses in the network.
Defaultwert: 1.0

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false'}

▷ **DLLayerLossCrossEntropy** (output_control) . . . . . dl_layer ⇝ *handle*
Cross entropy loss layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_loss_ctc (: : DLLayerInput, DLLayerInputLengths, DLLayerTarget, DLLayerTargetLengths, LayerName, GenParamName, GenParamValue : DLLayerLossCTC)

Create a CTC loss layer.

The operator `create_dl_layer_loss_ctc` creates a Connectionist Temporal Classification (CTC) loss layer whose handle is returned in `DLLayerLossCTC`. See the reference cited below for information about the CTC loss.

With this loss layer it is possible to train sequence to sequence models (Seq2Seq). E.g., it can be used to train a model that is able to read text in an image. In order to do so, the sequences are compared, thus the determined network prediction `DLLayerInput` with sequence length `DLLayerInputLengths` to the given `DLLayerTarget` with sequence length `DLLayerTargetLengths`.

The following variables are important to understand the input shapes:

- *T:* Maximum input sequence length (i.e., `width` of `DLLayerInput`)
- *S:* Maximum output sequence length (i.e., `width` of `DLLayerTarget`)
- *C:* Number of classes including 0 as the blank class ID (i.e., `depth` of `DLLayerInput`)

This layer expects multiple layers as input:

- `DLLayerInput`: Input layer with the network prediction. It expects the layer handle as value.
  Shape: [*T*,1,*C*]
- `DLLayerInputLengths`: Input layer specifying the input sequence length of each item in the batch. It expects the layer handle as value.
  Shape: [1,1,1]
- `DLLayerTarget`: Input layer which specifies the target sequences. It expects the layer handle as value.
  Shape: [*S*,1,1]
- `DLLayerTargetLengths`: Input layer which specifies the target sequence length of each item in the batch. It expects the layer handle as value.
  Shape: [1,1,1]

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The CTC loss is typically applied in a CNN as follows. The input sequence is expected to be encoded in some CNN layer with the output shape [width: *T*, height: 1, depth: *C*]. Typically the end of a large fully convolutional classifier is pooled in height down to 1 with an average pooling layer. It is important that the last layer is wide enough to hold enough information. In order to obtain the sequence prediction in the output depth a 1x1 convolutional layer is added after the pooling with the number of kernels set to *C*. In this use case the CTC loss obtains this convolutional layer as input layer DLLayerInput. The width of the input layer determines the maximum output sequence of the model.

The CTC loss can be applied to a batch of input items with differing input and target sequence lengths. *T* and *S* are the maximum lengths. In DLLayerInputLengths and DLLayerTargetLengths the individual length of each item in a batch needs to be specified.

Restrictions

- A model containing this layer cannot be trained on a CPU.
- A model containing this layer cannot be trained with a *'batch_size_multiplier'* != 1.0.
- The input layer DLLayerInput must not be a softmax layer. The softmax calculation is done internally in this layer. For inference, there should be an extra softmax layer connected to the DLLayerInput (see create_dl_layer_softmax).

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_loss_ctc can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'input_layer' (DLLayerInput, DLLayerInputLengths, DLLayerTarget, and/or DLLayerTargetLengths)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerLossCTC)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **DLLayerInput** (input_control) ..... dl_layer ⇝ *handle*
Input layer.

▷ **DLLayerInputLengths** (input_control) ..... dl_layer ⇝ *handle*
Input layer which specifies the input sequence length of each item in the batch.

▷ **DLLayerTarget** (input_control) ..... dl_layer ⇝ *handle*
Input layer which specifies the target sequences.

▷ **DLLayerTargetLengths** (input_control) ..... dl_layer ⇝ *handle*
Input layer which specifies the target sequence length of each item in the batch.

▷ **LayerName** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {}

▷ **DLLayerLossCTC** (output_control) ..... dl_layer ⇝ *handle*
CTC loss layer.

Beispiel

```
* Create a simple Seq2Seq model which overfits to a single output sequence.

* Input sequence length
T := 6
* Number of classes including blank (blank is always class_id: 0)
C := 3
* Batch Size
N := 1
* Maximum length of target sequences
S := 3

* Model creation
create_dl_layer_input ('input', [T,1,1], [], [], Input)
create_dl_layer_dense (Input, 'dense', T*C, [], [], DLLayerDense)
create_dl_layer_reshape (DLLayerDense, 'dense_reshape', [T,1,C], [], [],\
                         ConvFinal)

* Training part

* Specify the shapes without batch-size
* (batch-size will be specified in the model).
create_dl_layer_input ('ctc_input_lengths', [1,1,1], [], [],\
                       DLLayerInputLengths)
create_dl_layer_input ('ctc_target', [S,1,1], [], [], DLLayerTarget)
create_dl_layer_input ('ctc_target_lengths', [1,1,1], [], [],\
                       DLLayerTargetLengths)
* Create the loss layer
create_dl_layer_loss_ctc (ConvFinal, DLLayerInputLengths, DLLayerTarget,\
                          DLLayerTargetLengths, 'ctc_loss', [], [],\
                          DLLayerLossCTC)

* Get all names so that users can set values
get_dl_layer_param (ConvFinal, 'name', CTCInputName)
get_dl_layer_param (DLLayerInputLengths, 'name', CTCInputLengthsName)
get_dl_layer_param (DLLayerTarget, 'name', CTCTargetName)
get_dl_layer_param (DLLayerTargetLengths, 'name', CTCTargetLengthsName)

* Inference part
create_dl_layer_softmax (ConvFinal, 'softmax', [], [], DLLayerSoftMax)
create_dl_layer_depth_max (DLLayerSoftMax, 'prediction', 'argmax', [], [],\
```

```
                              DLLayerDepthMaxArg, _)

* Setting a seed because the weights of the network are randomly initialized
set_system ('seed_rand', 35)

create_dl_model ([DLLayerLossCTC,DLLayerDepthMaxArg], DLModel)

set_dl_model_param (DLModel, 'batch_size', N)
set_dl_model_param (DLModel, 'runtime', 'gpu')
set_dl_model_param (DLModel, 'learning_rate', 1)

* Create input sample for training
InputSequence := [0,1,2,3,4,5]
TargetSequence := [1,2,1]
create_dict (InputSample)
set_dict_tuple (InputSample, 'input', InputSequence)
set_dict_tuple (InputSample, 'ctc_input_lengths', |InputSequence|)
set_dict_tuple (InputSample, 'ctc_target', TargetSequence)
set_dict_tuple (InputSample, 'ctc_target_lengths', |TargetSequence|)
Eps := 0.01

PredictedSequence := []
dev_inspect_ctrl ([InputSequence, TargetSequence, CTCLoss, PredictedValues,\
                   PredictedSequence])
MaxIterations:= 15
for I := 0 to MaxIterations by 1
  apply_dl_model (DLModel, InputSample, ['prediction','softmax'],
                  DLResultBatch)
  get_dict_object (Softmax, DLResultBatch, 'softmax')
  get_dict_object (Prediction, DLResultBatch, 'prediction')
  PredictedValues := []
  for t := 0 to T-1 by 1
      get_grayval (Prediction, 0, t, PredictionValue)
      PredictedValues := [PredictedValues, PredictionValue]
  endfor
  train_dl_model_batch (DLModel, InputSample, DLTrainResult)

  get_dict_tuple (DLTrainResult, 'ctc_loss', CTCLoss)
  if (CTCLoss < Eps)
      break
  endif
  stop()
endfor

* Rudimentary implementation of fastest path prediction
PredictedSequence := []
LastV := -1
for I := 0 to |PredictedValues|-1 by 1
  V := PredictedValues[I]
  if (V == 0)
      LastV := -1
      continue
  endif
  if (|PredictedSequence| > 0 and V == LastV)
      continue
  endif
  PredictedSequence := [PredictedSequence, V]
  LastV :=  PredictedSequence[|PredictedSequence|-1]
endfor
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Literatur

Graves Alex et al., "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks."Proceedings of the 23rd international conference on Machine learning. 2006.

Modul

Deep Learning Training

create_dl_layer_loss_distance (: : DLLayerInput, DLLayerTarget, DLLayerWeights, LayerName, DistanceType, LossWeight, GenParamName, GenParamValue : DLLayerLossDistance)

Create a distance loss layer.

The operator create_dl_layer_loss_distance creates a distance loss layer whose handle is returned in DLLayerLossDistance.

This layer expects multiple layers as input:

- DLLayerInput: The feeding input layer which specifies the prediction (e.g., a softmax layer). It expects the layer handle as value.
- DLLayerTarget: The feeding input layer which specifies the target sequences (originating from the ground truth information). It expects the layer handle as value.
- DLLayerWeights: The feeding input layer which specifies the weight sequences. It expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The parameter LossWeight is an overall loss weight if there are multiple losses in the network.

The parameter DistanceType determines which distance measure is applied. Currently only *'l2'* is implemented. Depending on the generic parameter *'reduce'* this results in

- L2 loss distance as a tensor:

$$\begin{aligned}\texttt{DLLayerLossDistance}[i] &= 0.5 \cdot \texttt{LossWeight} \cdot \texttt{DLLayerWeights}[i] \\ &\cdot(\texttt{DLLayerInput}[i] - \texttt{DLLayerTarget}[i])^2,\end{aligned}$$

  in this case loss is the tensor of the same size as DLLayerInput.

- L2 loss distance as a scalar:

$$\texttt{DLLayerLossDistance} = \frac{\sum_{i=0}^{N-1} \texttt{DLLayerLossDistance}[i]}{\sum_{i=0}^{N-1} \texttt{DLLayerWeights}[i]},$$

  where N is a number of elements in DLLayerInput. Thus DLLayerInput, DLLayerTarget and DLLayerWeights should have the same size. Setting the weights in DLLayerWeights to *1* will result in a loss normalized over the number of elements.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

'reduce': Determines whether the output of the layer is reduced.

- *'true'*: The output is reduced to a scalar.
- *'false'*: The output of the layer is a tensor, where each element is a 'per-pixel' loss (squared differences).

Default: 'true'.

Certain parameters of layers created using this operator `create_dl_layer_loss_distance` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	set	get
'input_layer' (`DLLayerInput`, `DLLayerTarget`, or/and `DLLayerWeights`)	–	x
'loss_weight' (`LossWeight`)	x	x
'name' (`LayerName`)	x	x
'output_layer' (`DLLayerLossDistance`)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x
'reduce'	x	x

Parameter

▷ **`DLLayerInput`** (input_control) ..... dl_layer ⇝ *handle*
Input layer.

▷ **`DLLayerTarget`** (input_control) ..... dl_layer ⇝ *handle*
Target layer.

▷ **`DLLayerWeights`** (input_control) ..... dl_layer ⇝ *handle*
Weights layer.

▷ **`LayerName`** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **`DistanceType`** (input_control) ..... string ⇝ *string*
Type of distance.
Defaultwert: 'l2'
Werteliste: `DistanceType` ∈ {'l2'}

▷ **`LossWeight`** (input_control) ..... number ⇝ *real*
Overall loss weight if there are multiple losses in the network.
Defaultwert: 1.0

▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output', 'reduce'}

▷ **`GenParamValue`** (input_control) ..... attribute.value(-array) ⇝ *string*
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false'}

▷ **`DLLayerLossDistance`** (output_control) ..... dl_layer ⇝ *handle*
Distance loss layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_loss_focal (: : DLLayerInput, DLLayerTarget, DLLayerWeights, DLLayerNormalization, LayerName, LossWeight, Gamma, ClassWeights, Type, GenParamName, GenParamValue : DLLayerLossFocal)

Create a focal loss layer.

The operator create_dl_layer_loss_focal creates a focal loss layer whose handle is returned in DLLayerLossFocal. See the reference cited below for further information about its definition and parameter meanings.

This layer expects multiple layers as input:

- DLLayerInput: The feeding input layer which specifies the prediction (e.g., a sigmoid or softmax layer). It expects the layer handle as value.
- DLLayerTarget: The feeding input layer which specifies the target sequences (originating from the ground truth information). It expects the layer handle as value.
- DLLayerWeights: The feeding input layer which specifies the weight sequences. It expects the layer handle as value.
- DLLayerNormalization: The feeding input layer which specifies the factor to normalize the loss. This parameter is optional, it can be given by the layer handle as value or ignored handing over an empty tuple *[]*.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The parameter LossWeight is a overall loss weight if there are multiple losses in the network.

Default: '1.0'.

The parameter Gamma is the exponent of the focal factor.

The parameter ClassWeights defines class specific weights. All loss contributions of a class are weighted with the given factor. Typically, this is set to *'1.0/(Number of samples of the class)'*. Note, the length of this array has to correspond to the number of classes.

Default: '1.0' for all classes.

The parameter Type sets the focal loss options:

'focal_binary': Focal loss.

'sigmoid_focal_binary': Focal loss fused with sigmoid.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator `create_dl_layer_loss_focal` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	set	get
'focal_type' (`Type`)	–	x
'gamma' (`Gamma`)	x	x
'has_weight'	x	x
'input_layer' (`DLLayerInput`, `DLLayerTarget`, `DLLayerWeights`, and/or `DLLayerNormalization`)	–	x
'loss_weight' (`LossWeight`)	x	x
'name' (`LayerName`)	x	x
'output_layer' (`DLLayerLossFocal`)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **`DLLayerInput`** (input_control) ..... dl_layer ⇝ *handle*
Input layer.

▷ **`DLLayerTarget`** (input_control) ..... dl_layer ⇝ *handle*
Target layer.

▷ **`DLLayerWeights`** (input_control) ..... dl_layer ⇝ *handle*
Weights layer.

▷ **`DLLayerNormalization`** (input_control) ..... dl_layer ⇝ *handle*
Normalization layer.
Defaultwert: []

▷ **`LayerName`** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **`LossWeight`** (input_control) ..... number ⇝ *real* / integer
Overall loss weight if there are multiple losses in the network.
Defaultwert: 1.0

▷ **`Gamma`** (input_control) ..... number ⇝ *real* / integer
Focal factor.
Defaultwert: 2.0

▷ **`ClassWeights`** (input_control) ..... number(-array) ⇝ *real* / integer
Class specific weight.
Defaultwert: []

▷ **`Type`** (input_control) ..... string ⇝ *string*
Focal loss type.
Defaultwert: 'focal_binary'
Werteliste: `Type` ∈ {'focal_binary', 'sigmoid_focal_binary'}

▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string*
Generic input parameter values.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'true', 'false'}

▷ **DLLayerLossFocal** (output_control) . . . . . dl_layer ⇝ *handle*
Focal loss layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Literatur

T. Lin, P. Goyal, R. Girshick, K. He and P. Dollar, "Focal Loss for Dense Object Detection,"in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 2, pp. 318-327, 1 Feb. 2020, doi: 10.1109/TPAMI.2018.2858826.

Modul

Deep Learning Training

create_dl_layer_loss_huber (: : DLLayerInput, DLLayerTarget, DLLayerWeights, DLLayerNormalization, LayerName, LossWeight, Beta, GenParamName, GenParamValue : DLLayerLossHuber)

Create a Huber loss layer.

The operator create_dl_layer_loss_huber creates a Huber loss layer whose handle is returned in DLLayerLossHuber. The Huber loss is defined by

$$\begin{aligned} L_{Huber}(x,t,w,n) &:= \frac{\alpha}{n}\sum_{i=0}^{N-1} w_i l(x_i - t_i), \quad \text{with} \\ l(y) &:= \begin{cases} 0.5y^2/\beta & \text{if } |y| < \beta \\ |y| - 0.5\beta & \text{else.} \end{cases} \end{aligned}$$

This layer expects multiple layers as input:

- DLLayerInput: The feeding input layer x (usually a softmax). It expects the layer handle as value.
- DLLayerTarget: The feeding input layer which specifies the targets t. It expects the layer handle as value.
- DLLayerWeights: The feeding input layer which specifies the weights w. It expects the layer handle as value.

The underlying data tensors are assumed to be of the same shape with a total number of N elements.

The parameter DLLayerNormalization can be used to determine the normalization factor n. If DLLayerNormalization is set to an empty tuple, the sum over all weights is used for the normalization n.

The parameter LossWeight determines the scalar weight factor α.

The parameter Beta sets the value for β in the formula. If Beta is set to *0*, the Huber loss is equal to an L1-loss.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator `create_dl_layer_loss_huber` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	set	get
'beta' (`Beta`)	x	x
'input_layer' (`DLLayerInput`, `DLLayerTarget`, `DLLayerWeights`, or/and `DLLayerNormalization`)	–	x
'loss_weight' (`LossWeight`)	x	x
'name' (`LayerName`)	x	x
'output_layer' (`DLLayerLossHuber`)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **`DLLayerInput`** (input_control) . . . . . dl_layer ⇝ *handle*
Input layer.

▷ **`DLLayerTarget`** (input_control) . . . . . dl_layer ⇝ *handle*
Target layer.

▷ **`DLLayerWeights`** (input_control) . . . . . dl_layer ⇝ *handle*
Weights layer.

▷ **`DLLayerNormalization`** (input_control) . . . . . dl_layer ⇝ *handle*
Normalization layer.
Defaultwert: []

▷ **`LayerName`** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **`LossWeight`** (input_control) . . . . . number ⇝ *real*
Scalar weight factor.
Defaultwert: 1.0

▷ **`Beta`** (input_control) . . . . . number ⇝ *real*
Beta value in the loss-defining formula.
Defaultwert: 1.1
Restriktion: Beta >= 0

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false'}

▷ **`DLLayerLossHuber`** (output_control) . . . . . dl_layer ⇝ *handle*
Huber loss layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_lrn (: : DLLayerInput, LayerName, LocalSize, Alpha, Beta, K, NormRegion, GenParamName, GenParamValue : DLLayerLRN)

Create a LRN layer.

The operator create_dl_layer_lrn creates a local response normalization layer which performs normalization over a local window and whose handle is returned in DLLayerLRN. Currently, only *'across_channels'* can be set for NormRegion, which results in a normalization across the channel dimension. More detailed, a value x_c located in a channel with index c is normalized with a scale factor depending on a local window,

$$LRN(x_c) = x_c \cdot \left(k + \frac{\alpha}{n} \sum_{c'=\max(0,c-n/2)}^{\min(N-1,c+n/2)} (x_{c'})^2 \right)^{-\beta},$$

where n is the size of the local window given by LocalSize, N is the total number of channels, α is the scaling parameter Alpha (used as a normalization constant), β the exponent Beta (used as a contrast constant), and k is a constant summand K, which is used to avoid any singularities.

The parameter DLLayerInput determines the feeding input layer and expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_lrn can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'alpha' (Alpha)	–	x
'beta' (Beta)	–	x
'input_layer' (DLLayerInput)	–	x
'k' (K)	–	x
'local_size' (LocalSize)	–	x
'name' (LayerName)	x	x
'norm_region' (NormRegion)	–	x
'output_layer' (DLLayerLRN)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

- ▷ **DLLayerInput** (input_control) . . . . . dl_layer ⇝ *handle*
  Feeding layer.
- ▷ **LayerName** (input_control) . . . . . string ⇝ *string*
  Name of the layer.
- ▷ **LocalSize** (input_control) . . . . . number ⇝ *integer*
  Size of the local window.
  Defaultwert: 5
- ▷ **Alpha** (input_control) . . . . . number ⇝ *real*
  Scaling factor in the LRN formula.
  Defaultwert: 0.0001
- ▷ **Beta** (input_control) . . . . . number ⇝ *real*
  Exponent in the LRN formula.
  Defaultwert: 0.75
- ▷ **K** (input_control) . . . . . number ⇝ *real*
  Constant summand in the LRN formula.
  Defaultwert: 1.0
- ▷ **NormRegion** (input_control) . . . . . string ⇝ *string*
  'across_channels'.
  Defaultwert: 'across_channels'
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Generic input parameter names.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'is_inference_output'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
  Generic input parameter values.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {'true', 'false'}
- ▷ **DLLayerLRN** (output_control) . . . . . dl_layer ⇝ *handle*
  LRN layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

```
create_dl_layer_pooling ( : : DLLayerInput, LayerName, KernelSize,
    Stride, Padding, Mode, GenParamName,
    GenParamValue : DLLayerPooling )
```

Create a pooling layer.

The operator create_dl_layer_pooling creates a pooling layer whose handle is returned in DLLayerPooling.

The parameter DLLayerInput determines the feeding input layer and expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The parameter KernelSize specifies the filter kernel in the dimensions width and height.

The parameter Stride specifies how the filter is shifted.

The values for KernelSize and Stride can be set as

- a single value which is used for both dimensions
- a tuple [width, height] and [column, row], respectively.

The parameter Padding determines the padding, thus how many pixels with value 0 are appended on the boarder of the processed input image. Supported values are:

- *'half_kernel_size'*: The number of appended pixels depends on the specifies KernelSize. More precisely they are calculated as $\lfloor \texttt{KernelSize}/2 \rfloor$, where for the padding on the left / right bound the value of KernelSize in dimension width is regarded and for the padding on the upper / lower bound the value of KernelSize in height.
- *'implicit'*: No pixels are appended on the left or on the top of the input image. The number of pixels appended on the right or lower bound of the input image is zero if the kernel size is a divisor of the input dimension or $stride - (input_dim - kernel_size)\%stride$ else. The values $input_dim$, $kernel_size$, and $stride$ stand for the input width or height and the according entries in KernelSize and Stride, respectively.
- *'none'*: No pixels are appended.
- Number of pixels: Specify the number of pixels appended on each border. To do so, the following tuple lengths are supported:
  - Single number: Padding in all four directions left/right/top/bottom.
  - Two numbers: Padding in left/right and top/bottom: [l/r, t,b].
  - Four numbers: Padding on left, right, top, bottom side: [l,r,t,b].
    Restriction: 'runtime' 'gpu' does not support asymmetric padding, i.e., the padding values for the left and right side must be equal, as well as the padding values for the top and bottom side.

  Restriction: The integer padding values must be smaller than the value set for KernelSize in the corresponding dimension.

The output dimensions of the pooling layer are given by

$$output_dim = \left\lfloor \frac{input_dim + padding_begin + padding_end - kernel_size}{stride} \right\rfloor + 1$$

Thereby we use the following values: $output_dim$: output width, $input_dim$: input width, $kernel_size$: corresponding entry in KernelSize, $stride$: corresponding entry in Stride, $padding_begin$: number of pixels added to the left/top of the input image, and $padding_end$: number of pixels added to the right/bottom of the input image.

The parameter Mode specifies the mode of the pooling operation. Supported modes are:

'average': The resulting pixel value is the average of all pixel values in the filter.

'maximum': The resulting pixel value is the maximum of all pixel values in the filter.

'global_average': Same as mode *'average'*, but the internally used kernel size is changed to $\lfloor input_dim/kernel_size \rfloor$ and the internally used stride is set to the new kernel size. Following the above formula, this can be used to control the desired output dimensions without the knowledge of the dimensions of the input dimensions. E.g., if the average over all pixel values of the input shall be returned, set the kernel_size to *1* and the output width and height is equal to *1*. As a consequence, the output dimension equals the kernel size if the kernel size is a divisor of the input dimension.

'global_maximum': Same as mode *'global_average'*, but the maximum is calculated instead of the average.

For more information about the pooling layer see the „Solution Guide on Classification".

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether `apply_dl_model` will include the output of this layer in the dictionary `DLResultBatch` even without specifying this layer in `Outputs` (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator `create_dl_layer_pooling` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	set	get
'global'	–	x
'input_layer' (`DLLayerInput`)	–	x
'kernel_size' (`KernelSize`)	–	x
'name' (`LayerName`)	x	x
'output_layer' (`DLLayerPooling`)	–	x
'padding' (`Padding`)	–	x
'padding_type' (`Padding`)	–	x
'pooling_mode' (`Mode`)	–	x
'shape'	–	x
'stride' (`Stride`)	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **`DLLayerInput`** (input_control) . . . . . dl_layer ⇝ *handle*
Feeding layer.

▷ **`LayerName`** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **`KernelSize`** (input_control) . . . . . number-array ⇝ *integer*
Width and height of the filter kernels.
Defaultwert: [2,2]

▷ **`Stride`** (input_control) . . . . . number-array ⇝ *integer*
Bi-dimensional amount of filter shift.
Defaultwert: [2,2]

▷ **`Padding`** (input_control) . . . . . number(-array) ⇝ *string* / integer
Type of padding.
Defaultwert: 'none'
Wertevorschläge: `Padding` ∈ {'none', 'half_kernel_size', 'implicit'}

▷ **`Mode`** (input_control) . . . . . number ⇝ *string*
Mode of pooling operation.
Defaultwert: 'maximum'
Werteliste: `Mode` ∈ {'maximum', 'average', 'global_maximum', 'global_average'}

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false', 1.0, 0.9, 0.0}

▷ **DLLayerPooling** (output_control) ..... dl_layer ⇝ *handle*
Pooling layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_reshape (: : DLLayerInput, LayerName, Shape, GenParamName, GenParamValue : DLLayerReshape)

Create a reshape layer.

The operator create_dl_layer_reshape creates a reshape layer whose handle is returned in DLLayerReshape.

The parameter DLLayerInput determines the feeding input layer and expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The parameter Shape determines the output shape, into which the input data is converted. The value of Shape has to be given in the form [width, height, depth, batch_size], where the fourth value for the batch size is optional. The overall size of the data has to remain constant, i.e., width * height * depth * batch_size has to be equal to width_in * height_in * depth_in * batch_size_in, where [width_in, height_in, depth_in, batch_size_in] is the shape of the input graph layer.

Optionally, one or several entries of Shape may be set to *0* in order to keep the value of the corresponding input dimension. Moreover, at most one of the shape dimensions might be set to *-1* in order to calculate its value automatically. The value will then be calculated such that the overall size of the data remains constant. Be aware that this can only be done successfully if the calculated value is an integer.

If the batch size is specified and it is not set to *0*, at least one dimension of Shape must be set to *-1*. This is necessary, because for a model created with create_dl_model, the model's batch size should always be settable with set_dl_model_param. Hence, either the output batch size of the reshape layer equals the batch size of the model (batch size in Shape set to *0*), or at least one reshape dimension should be calculated automatically (one value in Shape set to *-1*). In case the batch size is not specified it is set to *0*, which leads to an output batch size equal to the input one.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_reshape can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'input_layer' (DLLayerInput)	–	x
'name' (LayerName)	x	x
'output_depth' (Shape)	–	x
'output_height' (Shape)	–	x
'output_layer' (Shape)	–	x
'output_width' (Shape)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **DLLayerInput** (input_control) ..... dl_layer ⇝ *handle*
Feeding layer.

▷ **LayerName** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **Shape** (input_control) ..... number-array ⇝ *integer*
Shape of the output graph layer data.
Defaultwert: [224,224,3]

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: GenParamName ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'true', 'false'}

▷ **DLLayerReshape** (output_control) ..... dl_layer ⇝ *handle*
Reshape layer.

Beispiel

```
* Minimal example for reshape-layer.
create_dl_layer_input ('input', [64, 32, 10], [], [], DLLayerInput)
create_dl_layer_reshape (DLLayerInput, 'reshape_wh', [32, 64, 0], [], [], \
                         DLLayerReshapeWH)
create_dl_layer_reshape (DLLayerInput, 'reshape_bs', [64, 32, 1, -1], [], \
                         [], DLLayerReshapeBS)
* DLLayerReshapeBS has batch size 10 and depth 1.
get_dl_layer_param (DLLayerReshapeBS, 'shape', ShapeReshapeBS)
* Create a model and change the batch-size.
create_dl_model (DLLayerReshapeBS, DLModel)
set_dl_model_param (DLModel, 'batch_size', 2)
* DLLayerReshapeBS has batch size 20 now.
get_dl_model_layer_param (DLModel, 'reshape_bs', 'shape', ShapeReshapeBS)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_dl_layer_input, create_dl_layer_concat

Nachfolger

create_dl_layer_convolution, create_dl_layer_dense

Modul

Deep Learning Training

create_dl_layer_softmax (: : DLLayerInput, LayerName, GenParamName, GenParamValue : DLLayerSoftMax)

Create a softmax layer.

The operator create_dl_layer_softmax creates a softmax layer whose handle is returned in DLLayerSoftMax.

The parameter DLLayerInput determines the feeding input layer and expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The softmax layer applies the softmax function which is defined for each input x_i as follows:

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_{j=0}^{N-1} \exp(x_j)}$$

where N is the number of inputs. During training, the result of the softmax function is additionally shifted and transformed by a logarithm function, such that the values are suitable as input to e.g., a cross entropy loss layer. This behavior can be changed by setting the generic parameter *'output_mode'*, see below.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'output_mode': This parameter determines if and in which case the output is transformed by a logarithm function:

- *'default'*: During inference, the result of the softmax function is returned as output while during training, the softmax is further transformed by a logarithm function.
- *'no_log_training'*: The logarithm transformation is omitted during training and the result of the softmax function is directly returned.
- *'log_inference'*: The logarithm of the softmax is calculated during inference in the same way as during training.

Default: 'default'.

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_softmax can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'input_layer'(DLLayerInput)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerSoftMax)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x
'output_mode'	–	x

Parameter

▷ **DLLayerInput** (input_control) ..... dl_layer ⇝ *handle*
Feeding layer.

▷ **LayerName** (input_control) ..... string ⇝ *string*
Name of the layer.

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'output_mode', 'is_inference_output'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'default', 'no_log_training', 'log_inference', 'true', 'false'}

▷ **DLLayerSoftMax** (output_control) ..... dl_layer ⇝ *handle*
Softmax layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_transposed_convolution (: : DLLayerInput, LayerName, KernelSize, Stride, KernelDepth, Groups, Padding, GenParamName, GenParamValue : DLLayerTransposedConvolution)

Create a transposed convolution layer.

The operator `create_dl_layer_transposed_convolution` creates a transposed convolution layer whose handle is returned in `DLLayerTransposedConvolution`.

The parameter `DLLayerInput` determines the feeding input layer and expects the layer handle as value.

The parameter `LayerName` sets an individual layer name. Note that if creating a model using `create_dl_model` each layer of the created network must have an unique name.

The parameter `KernelSize` specifies the filter kernel in the dimensions `width` and `height`. So far, only quadratic kernels are supported. As a consequence, this value must be a tuple of length 1.

The parameter `Stride` determines how the filter is shifted in `row` and `column` direction. This value must be a tuple of length 1.

The parameter `KernelDepth` defines the depth of the output feature maps. This value must be a tuple of length 1.

The parameter `Groups` determines the amount of filter groups. So far, only a single filter group is supported and as a consequence this value must be a tuple of length 1.

The parameter `Padding` determines the padding, thus how many pixels with value 0 are appended on the boarder of the processed input image. Supported values are:

- *'half_kernel_size'*: The number of appended pixels depends on the specifies `KernelSize`. More precisely they are calculated as $\lfloor$`KernelSize`$/2\rfloor$, where for the padding on the left/right bound the value of `KernelSize` in dimension `width` is regarded and for the padding on the upper/lower bound the value of `KernelSize` in `height`.

- *'none'*: No pixels are appended.

So far, the transposed convolution layer does not support bias terms.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'bias_term': Determines, whether the layer has bias terms. As mentioned above, bias terms are not supported yet, so this value is only retrievable.
Default: 'false'

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'learning_rate_multiplier': Learning rate multiplier for this layer that is used during training. If *'learning_rate_multiplier'* is set to *0*, the layer is skipped during training.
Default: 1.0

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

'weight_filler': Defines the mode how the weights are. See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
List of values: *'xavier'*, *'msra'*, *'const'*
Default: 'xavier'

'weight_filler_const_val': Constant weight initialization value if *'weight_filler'* has been set to *'const'*. See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
Default: 0.5

'weight_filler_variance_norm': Value range for *'weight_filler'*. See create_dl_layer_convolution for a detailed explanation of this parameter and its values.
List of values: *'norm_average'*, *'norm_in'*, *'norm_out'*, constant value (in combination of *'weight_filler'* set to *'msra'*)
Default: 'norm_in'

Certain parameters of layers created using this operator create_dl_layer_transposed_convolution can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'groups' (Groups)	–	x
'input_depth'	–	x
'input_layer' (DLLayerInput)	–	x
'kernel_depth' (KernelDepth)	–	x
'kernel_size' (KernelSize)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerTransposedConvolution)	–	x
'padding_type' (Padding)	–	x
'shape'	–	x
'stride' (Stride)	–	x
'type'	–	x

Generic Layer Parameters	set	get
'bias_term'	x	x
'is_inference_output'	x	x
'learning_rate_multiplier'	x	x
'num_trainable_params'	–	x
'weight_filler'	x	x
'weight_filler_const_val'	x	x
'weight_filler_variance_norm'	x	x

Achtung

This layer cannot be run by setting *'runtime'* to *'cpu'* for training. It is only applicable for training with a GPU by setting *'runtime'* to *'gpu'*.

Parameter

- ▷ **DLLayerInput** (input_control) . . . . . dl_layer ⇝ *handle*
  Feeding layer.
- ▷ **LayerName** (input_control) . . . . . string ⇝ *string*
  Name of the layer.
- ▷ **KernelSize** (input_control) . . . . . number ⇝ *integer*
  Width and height of the filter kernels.
  Defaultwert: 3
- ▷ **Stride** (input_control) . . . . . number ⇝ *integer*
  Amount of filter shift.
  Defaultwert: 1
- ▷ **KernelDepth** (input_control) . . . . . number ⇝ *integer*
  Depth of filter kernels.
  Defaultwert: 64
- ▷ **Groups** (input_control) . . . . . number ⇝ *integer*
  Number of filter groups.
  Defaultwert: 1
- ▷ **Padding** (input_control) . . . . . number ⇝ *string*
  Type of the padding.
  Defaultwert: 'none'
  Werteliste: `Padding` ∈ {'none', 'half_kernel_size'}
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Generic input parameter names.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'bias_term', 'is_inference_output', 'learning_rate_multiplier', 'weight_filler', 'weight_filler_const_val', 'weight_filler_variance_norm'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
  Generic input parameter values.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {'xavier', 'msra', 'const', 'norm_in', 'norm_out', 'norm_average', 'true', 'false'}
- ▷ **DLLayerTransposedConvolution** (output_control) . . . . . dl_layer ⇝ *handle*
  Transposed convolutional layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

create_dl_layer_zoom_factor (: : DLLayerInput, LayerName, ScaleWidth, ScaleHeight, Interpolation, AlignCorners, GenParamName, GenParamValue : DLLayerZoom)

Create a zoom layer using a size factor.

The operator create_dl_layer_zoom_factor creates a zoom layer using a size factor and returns the layer handle in DLLayerZoom.

The parameter DLLayerInput determines the feeding input layer and expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The parameters ScaleWidth and ScaleHeight specify the ratio between the output and the corresponding input dimension. Together they define the output size of the zoom layer DLLayerZoom.

The parameter Interpolation defines the interpolation mode. Currently only *'bilinear'* interpolation is supported.

The parameter AlignCorners defines how coordinates are transformed from the output to the input image:

'true': The transformation is applied on the HALCON Non-Standard Cartesian coordinate system (edge-centered, with the origin in the upper left corner, see chapter Transformationen / 2D-Transformationen). Using the x axis as an example, this leads to:

$$x_{input} = x_{output} * (length_{input} - 1)/(length_{output} - 1)$$

'false': The transformation is applied in the HALCON standard coordinate system (pixel centered, with the origin in the center of the upper left pixel, see chapter Transformationen / 2D-Transformationen). Using the x axis as an example, this leads to:

$$x_{input} = (x_{output} + 0.5) * length_{input}/length_{output} - 0.5$$

Note that this parameter applies only to Interpolation *'bilinear'*.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_zoom_factor can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'align_corners' (AlignCorners)	x	x
'input_layer' (DLLayerInput)	–	x
'interpolation_mode' (Interpolation)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerZoom)	–	x
'scale_params' (ScaleWidth and ScaleHeight)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **DLLayerInput** (input_control) . . . . . dl_layer ⇝ *handle*
Feeding layer.

▷ **LayerName** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **ScaleWidth** (input_control) . . . . . number ⇝ *real* / integer
Ratio output/input width of the layer.
Defaultwert: 2.0

▷ **ScaleHeight** (input_control) . . . . . number ⇝ *real* / integer
Ratio output/input height of the layer.
Defaultwert: 2.0

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Mode of interpolation.
Defaultwert: 'bilinear'
Werteliste: `Interpolation` ∈ {'bilinear'}

▷ **AlignCorners** (input_control) . . . . . string ⇝ *string*
Pixel alignment for interpolation.
Defaultwert: 'false'
Werteliste: `AlignCorners` ∈ {'true', 'false'}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false'}

▷ **DLLayerZoom** (output_control) . . . . . dl_layer ⇝ *handle*
Zoom layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`create_dl_layer_zoom_size`, `create_dl_layer_zoom_to_layer_size`

Modul

Deep Learning Training

create_dl_layer_zoom_size (: : DLLayerInput, LayerName, Width, Height, Interpolation, AlignCorners, GenParamName, GenParamValue : DLLayerZoom)

Create a zoom layer using an absolute output size.

The operator `create_dl_layer_zoom_size` creates a zoom layer using an absolute output size and returns the layer handle in `DLLayerZoom`.

The parameter `DLLayerInput` determines the feeding input layer and expects the layer handle as value.

The parameter LayerName sets an individual layer name. Note that if creating a model using create_dl_model each layer of the created network must have an unique name.

The parameters Width and Height define the absolute output size of the zoom layer DLLayerZoom.

The parameter Interpolation defines the interpolation mode. Currently only *'bilinear'* interpolation is supported.

The parameter AlignCorners defines how coordinates are transformed from the output to the input image:

'true': The transformation is applied on the HALCON Non-Standard Cartesian coordinate system (edge-centered, with the origin in the upper left corner, see chapter Transformationen / 2D-Transformationen). Using the x axis as an example, this leads to:

$$x_{input} = x_{output} * (length_{input} - 1)/(length_{output} - 1)$$

'false': The transformation is applied in the HALCON standard coordinate system (pixel centered, with the origin in the center of the upper left pixel, see chapter Transformationen / 2D-Transformationen). Using the x axis as an example, this leads to:

$$x_{input} = (x_{output} + 0.5) * length_{input}/length_{output} - 0.5$$

Note that this parameter applies only to Interpolation *'bilinear'*.

The following generic parameters GenParamName and the corresponding values GenParamValue are supported:

'is_inference_output': Determines whether apply_dl_model will include the output of this layer in the dictionary DLResultBatch even without specifying this layer in Outputs (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator create_dl_layer_zoom_size can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using set_dl_model_layer_param and which ones can be retrieved using get_dl_model_layer_param or get_dl_layer_param. Note, the operators set_dl_model_layer_param and get_dl_model_layer_param require a model created by create_dl_model.

Layer Parameters	set	get
'align_corners' (AlignCorners)	x	x
'input_layer' (DLLayerInput)	–	x
'interpolation_mode' (Interpolation)	–	x
'name' (LayerName)	x	x
'output_layer' (DLLayerZoom)	–	x
'scale_params' (Width and Height)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **DLLayerInput** (input_control) . . . . . dl_layer ⇝ *handle*
Feeding layer.

▷ **LayerName** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **Width** (input_control) . . . . . number ⇝ *integer*
Absolute width of the output layer.
Defaultwert: 100

▷ **Height** (input_control) . . . . . number ⇝ *integer*
Absolute height of the output layer.
Defaultwert: 100

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Mode of interpolation.
Defaultwert: 'bilinear'
Werteliste: `Interpolation` ∈ {'bilinear'}

▷ **AlignCorners** (input_control) . . . . . string ⇝ *string*
Pixel alignment for interpolation.
Defaultwert: 'false'
Werteliste: `AlignCorners` ∈ {'true', 'false'}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false'}

▷ **DLLayerZoom** (output_control) . . . . . dl_layer ⇝ *handle*
Zoom layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`create_dl_layer_zoom_factor`, `create_dl_layer_zoom_to_layer_size`

Modul

Deep Learning Training

create_dl_layer_zoom_to_layer_size (: : DLLayerInput, DLLayerReference, LayerName, Interpolation, AlignCorners, GenParamName, GenParamValue : DLLayerZoom)

Create a zoom layer using the output size of a reference layer.

The operator `create_dl_layer_zoom_to_layer_size` creates a zoom layer using the output size of a reference layer and returns the layer handle in `DLLayerZoom`.

The parameter `DLLayerInput` determines the feeding input layer and expects the layer handle as value.

The parameter `DLLayerReference` is used to define the output size of the zoom layer `DLLayerZoom`: the size is adapted to the output size of `DLLayerReference`.

The parameter `LayerName` sets an individual layer name. Note that if creating a model using `create_dl_model` each layer of the created network must have an unique name.

The parameter `Interpolation` defines the interpolation mode. Currently only *'bilinear'* interpolation is supported.

The parameter `AlignCorners` defines how coordinates are transformed from the output to the input image:

'true': The transformation is applied on the HALCON Non-Standard Cartesian coordinate system (edge-centered, with the origin in the upper left corner, see chapter Transformationen / 2D-Transformationen). Using the x axis as an example, this leads to:

$$x_{input} = x_{output} * (length_{input} - 1)/(length_{output} - 1)$$

'false': The transformation is applied in the HALCON standard coordinate system (pixel centered, with the origin in the center of the upper left pixel, see chapter Transformationen / 2D-Transformationen). Using the x axis as an example, this leads to:

$$x_{input} = (x_{output} + 0.5) * length_{input}/length_{output} - 0.5$$

Note that this parameter applies only to `Interpolation` *'bilinear'*.

The following generic parameters `GenParamName` and the corresponding values `GenParamValue` are supported:

'is_inference_output': Determines whether `apply_dl_model` will include the output of this layer in the dictionary `DLResultBatch` even without specifying this layer in `Outputs` (*'true'*) or not (*'false'*).
Default: 'false'

'num_trainable_params': Number of trainable parameters (weights and biases) of the layer.

Certain parameters of layers created using this operator `create_dl_layer_zoom_to_layer_size` can be set and retrieved using further operators. The following tables give an overview, which parameters can be set using `set_dl_model_layer_param` and which ones can be retrieved using `get_dl_model_layer_param` or `get_dl_layer_param`. Note, the operators `set_dl_model_layer_param` and `get_dl_model_layer_param` require a model created by `create_dl_model`.

Layer Parameters	set	get
'align_corners' (`AlignCorners`)	x	x
'input_layer' (`DLLayerInput`)	–	x
'interpolation_mode' (`Interpolation`)	–	x
'name' (`LayerName`)	x	x
'output_layer' (`DLLayerZoom`)	–	x
'scale_params' (`DLLayerReference`)	–	x
'shape'	–	x
'type'	–	x

Generic Layer Parameters	set	get
'is_inference_output'	x	x
'num_trainable_params'	–	x

Parameter

▷ **`DLLayerInput`** (input_control) . . . . . dl_layer ⇝ *handle*
Feeding layer.

▷ **`DLLayerReference`** (input_control) . . . . . dl_layer ⇝ *handle*
Reference layer to define the output size.

▷ **`LayerName`** (input_control) . . . . . string ⇝ *string*
Name of the layer.

▷ **Interpolation** (input_control) ..... string ⇝ *string*
Mode of interpolation.
Defaultwert: 'bilinear'
Werteliste: `Interpolation` ∈ {'bilinear'}

▷ **AlignCorners** (input_control) ..... string ⇝ *string*
Pixel alignment for interpolation.
Defaultwert: 'false'
Werteliste: `AlignCorners` ∈ {'true', 'false'}

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Generic input parameter names.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'is_inference_output'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Generic input parameter values.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false'}

▷ **DLLayerZoom** (output_control) ..... dl_layer ⇝ *handle*
Zoom layer.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`create_dl_layer_zoom_size`, `create_dl_layer_zoom_factor`

Modul

Deep Learning Training

create_dl_model (: : OutputLayers : DLModelHandle)

Create a deep learning model.

The operator `create_dl_model` creates a deep learning model from a graph and returns its handle in `DLModelHandle`.

A deep learning model in HALCON mainly consists of a directed acyclic graph that defines the networks architecture. Further components of a deep learning model in HALCON are parameters as *'class_names'*, *'class_ids'*, and many others, or hyperparameters that are needed to train a model, as for example the *'learning_rate'*. While parameters and hyperparameters can be set after creation of the model using `set_dl_model_param`, the model itself can only be created using `create_dl_model` if its network architecture is given in form of a graph.

To build a graph that defines the models network architecture, one needs to put together the networks layers. In general, a graph starts with an input layer. A subsequent layer that follows after the input layer uses the input layer as feeding layer, and the new layer itself might be used as a feeding layer for the next layer, and so on. This is repeated until the graphs output layers (e.g., softmax or loss layers) are appended to the graph. To create a layer, use its specified creation operator, e.g., an input layer is created using `create_dl_layer_input`, a convolution layer is created using `create_dl_layer_convolution`, and so on.

When the graph is defined, a model can be created using `create_dl_model` by passing over the graphs output layer handles in `OutputLayers`. Note that the output layer handles save all other layers that directly or indirectly serve as feeding input layers for the output layers during their creation. This means that the output layer handles keep the whole network architecture necessary for the creation of the model using `create_dl_model`.

The type of the created model, hence the task the model is designed for (classification, object detection, segmentation), is only given by the networks architecture. However, if the networks architecture allows it, the type of the model, *'type'*, can be set using `set_dl_model_param`. A specified model type allows a more user friendly usage in the HALCON deep learning workflow. Supported types are:

'generic': This is the default model type. The task the models neuronal network can solve is defined by its architecture. When apply_dl_model is applied for inference, the operator returns the activations of the output layers. To train the model using train_dl_model_batch, the underlying graph requires loss layers.

'classification': The model is specified for classification and all layers required for training the model are adapted to the model. When apply_dl_model is applied for inference, the output is adapted according to the type, see apply_dl_model for more details. See Deep Learning / Klassifikation for further information.
In addition, the operator gen_dl_model_heatmap can be used to display the models heatmap.

'detection': The model is specified for detection and all layers and anchors required for training the model are adapted to the model. When apply_dl_model is applied for inference, the output is adapted according to the type, see apply_dl_model for more details. See Deep Learning / Objektdetektion for further information.

'segmentation': The model is specified for segmentation and all layers required for training the model are adapted to the model. When apply_dl_model is applied for inference, the output is adapted according to the type, see apply_dl_model for more details. See Deep Learning / Semantische Segmentierung for further information.

Furthermore, many deep learning procedures provide more functionality for the model if its type is specified. As an example, dev_display_dl_data can be used to display the inferred results more nicely.

Note that setting a model type requires that the graph fulfills certain structure conditions. We recommend to follow the architecture of our delivered neuronal networks if the model type should be set to one of these types.

Parameter

▷ **OutputLayers** (input_control) ..... dl_layer(-array) ⇝ *handle*
Output layers of the graph.

▷ **DLModelHandle** (output_control) ..... dl_model ⇝ *handle*
Handle of the deep learning model.

Ergebnis

If the parameters are valid, the operator create_dl_model returns the value TRUE. If necessary, an exception is raised.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

create_dl_layer_softmax, create_dl_layer_loss_cross_entropy, create_dl_layer_loss_focal, create_dl_layer_loss_huber

Nachfolger

set_dl_model_param

Modul

Deep Learning Training

get_dl_layer_param (: : DLLayer, GenParamName : GenParamValue)

Return the parameters of a deep learning layer.

The operator get_dl_layer_param returns the parameters GenParamName of the deep learning layer DLLayer in GenParamValue.

Depending on the type of the layer, different parameter names are valid. Which generic and layer-specific parameters can be queried is described in the specific operator references (create_dl_layer_*).

Parameter

- ▷ **DLLayer** (input_control) . . . . . dl_layer ⇝ *handle*
  Layer.
- ▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
  Parameter to query.
  Defaultwert: 'shape'
  Werteliste: GenParamName ∈ {'input_layer', 'name', 'shape', 'type'}
- ▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *real* / integer / string / handle
  Value of the queried parameter.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

get_dl_model_layer (: : DLModelHandle, LayerNames : DLLayers)

Create a deep copy of the layers and all of their graph ancestors in a given deep learning model.

The operator get_dl_model_layer creates a deep copy of every layer named in LayerNames and all their graph ancestors in the deep learning model DLModelHandle. You can retrieve the unique layer names using get_dl_model_param with its option *'summary'*.

You might use the output layers returned in DLLayers as inputs to the create_dl_layer_* and create_dl_model operators in order to create novel model architectures based on existing models.

Please note, that the output layers are copies. They contain the same weights and settings as in the given input model but they are unique copies. You cannot alter the existing model by changing the output layers.

Parameter

- ▷ **DLModelHandle** (input_control) . . . . . dl_model ⇝ *handle*
  Deep learning model.
- ▷ **LayerNames** (input_control) . . . . . string(-array) ⇝ *string*
  Names of the layers to be copied.
- ▷ **DLLayers** (output_control) . . . . . dl_layer(-array) ⇝ *handle*
  Copies of layers and all of their ancestors.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_dl_model

Nachfolger

create_dl_model, create_dl_layer_activation, create_dl_layer_batch_normalization, create_dl_layer_class_id_conversion, create_dl_layer_class_id_conversion, create_dl_layer_concat, create_dl_layer_convolution, create_dl_layer_dense, create_dl_layer_depth_max,

create_dl_layer_dropout, create_dl_layer_elementwise,
create_dl_layer_loss_cross_entropy, create_dl_layer_loss_ctc,
create_dl_layer_loss_distance, create_dl_layer_loss_focal,
create_dl_layer_loss_huber, create_dl_layer_lrn, create_dl_layer_pooling,
create_dl_layer_reshape, create_dl_layer_softmax,
create_dl_layer_transposed_convolution, create_dl_layer_zoom_factor,
create_dl_layer_zoom_size, create_dl_layer_zoom_to_layer_size

Modul

Deep Learning Training

get_dl_model_layer_activations (: Activations : DLModelHandle,
LayerName :)

Get the activations of a Deep Learning model layer.

The operator get_dl_model_layer_activations returns in Activations the activations of the specified LayerName of the model DLModelHandle.

Activations is a tuple of batch_size many objects, where every object is an image having the size (width, height, depth) of the given LayerName.

Parameter

- ▷ **Activations** (output_object) ..... image(-array) ⇝ *object :* real
  Output activations.
- ▷ **DLModelHandle** (input_control) ..... dl_model ⇝ *handle*
  Handle of the deep learning model.
- ▷ **LayerName** (input_control) ..... string ⇝ *string*
  Layer name.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

get_dl_model_layer_gradients (: Gradients : DLModelHandle,
LayerName :)

Get the gradients of a Deep Learning model layer.

The operator get_dl_model_layer_gradients returns in Gradients the gradients of the specified LayerName of the model DLModelHandle.

Gradients is a tuple of batch_size many objects, where every object is an image having the size (width, height, depth) of the given LayerName.

Parameter

- ▷ **Gradients** (output_object) ..... image(-array) ⇝ *object :* real
  Output gradients.
- ▷ **DLModelHandle** (input_control) ..... dl_model ⇝ *handle*
  Handle of the deep learning model.
- ▷ **LayerName** (input_control) ..... string ⇝ *string*
  Layer name.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

get_dl_model_layer_param (: : DLModelHandle, LayerName, ParamName : ParamValue)

Retrieve parameter values for a given layer.

The operator get_dl_model_layer_param returns for a layer the value of the parameter ParamName in ParamValue. The layer is referred by its name LayerName in the model DLModelHandle. You can retrieve the layer names using get_dl_model_param with its option *'summary'*.

Which generic and layer-specific parameters can be queried is described in the specific operator references (create_dl_layer_*).

Parameter

- ▷ **DLModelHandle** (input_control) ..... dl_model ⇝ *handle*
  Deep learning model.
- ▷ **LayerName** (input_control) ..... string ⇝ *string*
  Name of the layer.
- ▷ **ParamName** (input_control) ..... string ⇝ *string*
  Name of the queried parameter.
  Defaultwert: 'type'
  Werteliste: ParamName ∈ {'bias_filler', 'bias_filler_variance_norm', 'bias_filler_const_val', 'bias_term', 'input_layer', 'is_inference_output', 'learning_rate_multiplier', 'learning_rate_multiplier_bias', 'name', 'num_trainable_params', 'output_layer', 'shape', 'type', 'upper_bound', 'weight_filler', 'weight_filler_const_val', 'weight_filler_variance_norm'}
  Restriktion: length(ParamName) > 0
- ▷ **ParamValue** (output_control) ..... tuple(-array) ⇝ *string* / real / integer
  Value of the queried parameter.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

get_dl_model_param, set_dl_model_layer_param

Modul

Deep Learning Training

get_dl_model_layer_weights (: Weights : DLModelHandle, LayerName, WeightsType :)

Get the weights (or values) of a Deep Learning model layer.

The operator get_dl_model_layer_weights returns in Weights the values of a LayerName of the model DLModelHandle.

The parameter WeightsType determines which type of layer values are retrieved. The following values are supported for WeightsType:

- *'batchnorm_mean'*: Batch-wise calculated mean values to normalize the inputs. For further information, please refer to `create_dl_layer_batch_normalization`.
  Restriction: This value is only supported if the layer is of type *'batchnorm'*.
- *'batchnorm_mean_avg'*: Average of the batch-wise calculated mean values to normalize the inputs. For further information, please refer to `create_dl_layer_batch_normalization`.
  Restriction: This value is only supported if the layer is of type *'batchnorm'*.
- *'batchnorm_variance'*: Batch-wise calculated variance values to normalize the inputs. For further information, please refer to `create_dl_layer_batch_normalization`.
  Restriction: This value is only supported if the layer is of type *'batchnorm'*.
- *'batchnorm_variance_avg'*: Average of the batch-wise calculated variance values to normalize the inputs. For further information, please refer to `create_dl_layer_batch_normalization`.
  Restriction: This value is only supported if the layer is of type *'batchnorm'*.
- *'bias'*: Biases of the layer.
- *'bias_gradient'*: Gradients of the biases of the layer.
- *'bias_gradient_norm_l2'*: Gradients of the biases of the layer in terms of L2 norm.
- *'bias_norm_l2'*: Biases of the layer in terms of L2 norm.
- *'bias_update'*: Update of the biases of the layer. This is used in e.g., a solver which uses the last update.
- *'bias_update_norm_l2'*: Update of the biases of the layer in terms of L2 norm. This is used in a solver which uses the last update.
- *'weights'*: Weights of the layer.
- *'weights_gradient'*: Gradients of the weights of the layer.
- *'weights_gradient_norm_l2'*: Gradients of the weights of the layer in terms of L2 norm.
- *'weights_norm_l2'*: Weights of the layer in terms of L2 norm.
- *'weights_update'*: Update of the weights of the layer. This is used in a solver which uses the last update.
- *'weights_update_norm_l2'*: Update of the weights of the layer in terms of L2 norm. This is used in a solver which uses the last update.

The following tables give an overview, which parameters for `WeightsType` can be set using `set_dl_model_layer_weights` and which ones can be retrieved using `get_dl_model_layer_weights`.

Layer Parameters	set	get
'batchnorm_mean'	x	x
'batchnorm_mean_avg'	x	x
'batchnorm_variance'	x	x
'batchnorm_variance_avg'	x	x
'bias'	x	x
'bias_gradient'	–	x
'bias_gradient_norm_l2'	–	x
'bias_norm_l2'	–	x
'bias_update'	–	x
'bias_update_norm_l2'	–	x
'weights'	x	x
'weights_gradient'	–	x
'weights_gradient_norm_l2'	–	x
'weights_norm_l2'	–	x
'weights_update'	–	x
'weights_update_norm_l2'	–	x

Achtung

The operator `get_dl_model_layer_weights` is only applicable to self-created networks. For networks delivered by HALCON, the operator returns an empty tuple.

Parameter

- ▷ **`Weights`** (output_object) ..... image(-array) ⇝ *object :* real
  Output weights.
- ▷ **`DLModelHandle`** (input_control) ..... dl_model ⇝ *handle*
  Handle of the deep learning model.
- ▷ **`LayerName`** (input_control) ..... string ⇝ *string*
  Layer name.
- ▷ **`WeightsType`** (input_control) ..... string ⇝ *string*
  Selected type of layer values to be returned.
  Defaultwert: 'weights'
  Werteliste: `WeightsType` ∈ {'weights', 'weights_norm_l2', 'weights_update', 'weights_update_norm_l2', 'weights_gradient', 'weights_gradient_norm_l2', 'bias', 'bias_norm_l2', 'bias_update', 'bias_update_norm_l2', 'bias_gradient', 'bias_gradient_norm_l2', 'batchnorm_mean', 'batchnorm_variance', 'batchnorm_mean_avg', 'batchnorm_variance_avg'}

Beispiel

```
set_system ('seed_rand', 42)
* Create a small model network.
create_dl_layer_input ('input', [InputImageSize[0],InputImageSize[1],1], [],\
                       [], DLGraphNodeInput)
create_dl_layer_convolution (DLGraphNodeInput, 'conv', 3, 1, 1, 2, 1, 'none',\
                             'none', [], [], DLGraphNodeConvolution)
create_dl_layer_activation (DLGraphNodeConvolution, 'relu', 'relu', [], [],\
                            DLGraphNodeActivation)
create_dl_layer_dense (DLGraphNodeActivation, 'dense', 3, [], [],\
                       DLGraphNodeDense)
create_dl_layer_softmax (DLGraphNodeDense, 'softmax', [], [],\
                         DLGraphNodeSoftMax)
create_dl_model (DLGraphNodeSoftMax, DLModelHandle)
*
set_dl_model_param (DLModelHandle, 'type', 'classification')
set_dl_model_param (DLModelHandle, 'batch_size', 1)
set_dl_model_param (DLModelHandle, 'runtime', 'gpu')
set_dl_model_param (DLModelHandle, 'runtime_init', 'immediately')
*
* Train for 5 iterations.
for TrainIterations := 1 to NumTrainIterations by 1
    train_dl_model_batch (DLModelHandle, DLSample, DLTrainResult)
endfor
*
* Get the gradients, weights, and activations.
get_dl_model_layer_gradients (GradientsSoftmax, DLModelHandle, 'softmax')
get_dl_model_layer_gradients (GradientsDense, DLModelHandle, 'dense')
get_dl_model_layer_gradients (GradientsConv, DLModelHandle, 'conv')
*
get_dl_model_layer_weights (WeightsDense, DLModelHandle, 'dense',\
                        'weights_gradient')
get_dl_model_layer_weights (WeightsConv, DLModelHandle, 'conv',\
                        'weights_gradient')
*
get_dl_model_layer_activations (ActivationsDense,  DLModelHandle, 'dense')
get_dl_model_layer_activations (ActivationsConv,  DLModelHandle, 'conv')
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_dl_model`, `train_dl_classifier_batch`, `set_dl_model_layer_weights`

Nachfolger

`set_dl_model_layer_weights`

Alternativen

`get_dl_model_layer_activations`, `get_dl_model_layer_gradients`

Modul

Deep Learning Training

load_dl_model_weights (: : DLModelHandleSource, DLModelHandleTarget : ChangesByLayer)

Load the weights of a source model into a target model.

The operator `load_dl_model_weights` loads weights of a source model `DLModelHandleSource` into a target model `DLModelHandleTarget`. Thereby applies for every layer in the target model: Its weights are only changed if there is a layer in the source model having the same name and the same weight-shape. Note that `DLModelHandleSource` must be different from `DLModelHandleTarget`, i.e., you cannot use the same model handle as source and target.

`ChangesByLayer` is a tuple indicating for every target layer how many weights changed. Its entries are sorted by ascending layer IDs. The layer IDs can be queried via the operator `get_dl_model_param` with the parameter *'summary'*.

Note, that 'weights' means all weights and biases for all layers which can have such values (e.g., convolution, batchnorm, etc.).

Achtung

The operator `load_dl_model_weights` is only applicable to self-created networks. For networks delivered by HALCON, the operator is not working.

Parameter

- ▷ **DLModelHandleSource** (input_control) ..... dl_model ⇝ *handle*
  Handle of the source deep learning model.
- ▷ **DLModelHandleTarget** (input_control) ..... dl_model ⇝ *handle*
  Handle of the target deep learning model.
- ▷ **ChangesByLayer** (output_control) ..... integer(-array) ⇝ *integer*
  Indicates for every target layer how many weights changed.

Ergebnis

If the parameters are valid, the operator `load_dl_model_weights` returns the value TRUE. If necessary, an exception is raised.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Deep Learning Training

set_dl_model_layer_param (: : DLModelHandle, LayerName, ParamName, ParamValue :)

Set parameter values of a given layer.

The operator set_dl_model_layer_param sets the value ParamValue of the parameter ParamName for a layer. The layer is referred to by its name LayerName in the model DLModelHandle. You can retrieve the layer names using get_dl_model_param with its option *'summary'*.

Which generic and layer-specific parameters can be set is described in the specific operator references (create_dl_layer_*).

Parameter

- ▷ **DLModelHandle** (input_control) ..... dl_model ⇝ *handle*
  Deep learning model.
- ▷ **LayerName** (input_control) ..... string ⇝ *string*
  Name of the layer.
- ▷ **ParamName** (input_control) ..... attribute.name ⇝ *string*
  Name of the set parameter.
  Defaultwert: []
  Werteliste: ParamName ∈ {'bias_filler', 'bias_filler_variance_norm', 'bias_filler_const_val', 'bias_term', 'is_inference_output', 'learning_rate_multiplier', 'learning_rate_multiplier_bias', 'name', 'upper_bound', 'weight_filler', 'weight_filler_const_val', 'weight_filler_variance_norm'}
- ▷ **ParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / integer / real
  Value of the set parameter.
  Defaultwert: []
  Werteliste: ParamValue ∈ {'xavier', 'msra', 'const', 'norm_in', 'norm_out', 'norm_average', 'true', 'false'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

set_dl_model_param, get_dl_model_param, get_dl_model_layer_param

Modul

Deep Learning Training

set_dl_model_layer_weights (Weights : : DLModelHandle, LayerName, WeightsType :)

Set the weights (or values) of a Deep Learning model layer.

The operator set_dl_model_layer_weights sets for the model DLModelHandle the given Weights in the specified LayerName.

The parameter WeightsType determines which type of layer values are set. Which values can be set, please refer to the get_dl_model_layer_weights documentation.

Achtung

The operator set_dl_model_layer_weights is only applicable to self-created networks. For networks delivered by HALCON, the operator does have no impact.

Parameter

- ▷ **Weights** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* real
  Input weights.
- ▷ **DLModelHandle** (input_control) ..... dl_model ⇝ *handle*
  Handle of the deep learning model.
- ▷ **LayerName** (input_control) ..... string ⇝ *string*
  Layer name.

▷ **WeightsType** (input_control) . . . . . string ⇝ *string*
Selected type of layer values to be set.
Defaultwert: 'weights'
Werteliste: WeightsType ∈ {'weights', 'bias', 'batchnorm_mean', 'batchnorm_variance', 'batchnorm_mean_avg', 'batchnorm_variance_avg'}

Beispiel

```
* Create weights for a convolution layer.
gen_image_const (Weights, 'real', 1, 1)
paint_region (Weights, Weights, Weights, 1, 'fill')
gen_empty_obj (WeightsArray)
for Index := 0 to 10 by 1
    concat_obj (WeightsArray, Weights, WeightsArray)
endfor
*
* Input image with rows consisting of 1s to 10s.
gen_image_const (Image, 'real', 10, 10)
for Index := 0 to 9 by 1
    gen_rectangle1 (Rectangle, Index, 0, Index, 9)
    paint_region (Rectangle, Image, Image, Index + 1, 'fill')
endfor
*
* Create a small model network.
create_dl_layer_input ('image', [10, 2, 1], [], [], ImageNode)
create_dl_layer_convolution (ImageNode, 'conv', 1, 1, 2, 11, 1, 'none', \
                             'none', [], [], ConvNode)
create_dl_layer_zoom_factor (ConvNode, 'zoom', 2, 2, 'bilinear', 'true', [],\
                             [], ZoomNode)
create_dl_model (ZoomNode, DLModelHandle)
set_dl_model_param (DLModelHandle, 'runtime', 'cpu')
*
* Set the weights to the convolution layer.
set_dl_model_layer_weights (WeightsArray, DLModelHandle, 'conv', 'weights')
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_dl_model, get_dl_model_layer_weights

Nachfolger

get_dl_model_layer_weights

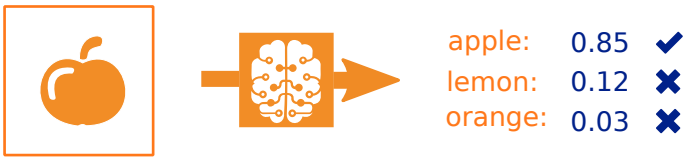
Modul

Deep Learning Training

8.3 Klassifikation

Dieses Kapitel beschreibt, wie Deep Learning-basierte Klassifikation in der Trainings- und Inferenzphase genutzt wird.

Deep Learning-basierte Klassifikation ist eine Methode, bei der einem Bild eine Menge an Konfidenzwerten zugeordnet werden. Diese Konfidenzwerte geben an, wie wahrscheinlich das Bild zu jeder der zu unterscheidenden Klassen gehört. Folglich bedeutet Klassifikation, soweit nur die beste Vorhersage berücksichtigt wird, die Zuordnung einer spezifischen Klasse aus einer Menge an Klassen zu einem Bild. Dies wird im folgenden Schema dargestellt.



Ein mögliches Beispiel zur Klassifikation in dem das Netzwerk zwischen drei Klassen unterscheidet. Für das Eingabebild werden Konfidenzwerte für jede der drei zu unterscheidenden Klassen bestimmt: 'apple' 0.85, 'lemon' 0.03, und 'orange' 0.12. Die beste Vorhersage sagt uns, dass das Bild als 'apple' erkannt wird.

Um Ihre spezifische Aufgabe zu erledigen, also Ihre vorliegende Daten in die zu unterscheidenden Klassen einzuteilen, muss der Klassifikator entsprechend trainiert werden. In HALCON wird eine Technik verwendet, die Transferlernen genannt wird (siehe auch das Kapitel Deep Learning). Dabei werden vortrainierte Netzwerke (Klassifikatoren) zur Verfügung gestellt, welche mit einer riesigen Anzahl gelabelter Bilddaten trainiert wurden. Diese Klassifikatoren wurden trainiert und getestet um industrielle Bildklassifikationsaufgaben gut zu lösen. Ein solcher Klassifikator, der bereits für allgemeine Klassifikation trainiert wurde, wird nun für die geforderte Aufgabe nachtrainiert. Dafür muss der Klassifikator wissen, welche Klassen unterschieden werden müssen und wie Beispiele dieser Klassen aussehen. Diese Informationen erhält er aus Ihrem Datensatz, d.h. aus Ihren Bildern und den von Ihnen vergebenen Ground Truth Labels, den Ground Truth Klassenzugehörigkeiten. Weitere Informationen zu den Anforderungen an die Daten sind im Abschnitt „Daten“ zu finden.

Die spezifischen Systemvoraussetzungen zur Anwendung von Deep Learning-basierter Klassifikation können im HALCON „`Installation Guide`“ eingesehen werden.

Die folgenden Abschnitte sind Einführungen zum generellen Ablauf einer Klassifikations-Anwendung mittels Deep Learning, Informationen zu den verwendeten Daten sowie Erklärungen zu den Evaluierungsmaßen.

Genereller Ablauf

Dieser Abschnitt beschreibt den generellen Ablauf einer Klassifikation mittels Deep Learning. Er ist unterteilt in die vier Bereiche Vorverarbeitung des Datensatzes, Trainieren des Modells, Evaluierung des trainierten Modells und Inferenz auf neuen Bildern. Dabei wird angenommen, dass der Datensatz bereits gelabelt ist, siehe den Absatz „Daten“ unten. Die HDevelop Beispiels-Reihe `classify_pill_defects_deep_learning` zeigt eine mögliche Anwendung unterteilt in die einzelnen Bereiche.

Vorverarbeitung des Datensatzes Dieser Abschnitt behandelt die notwendigen Anpassungen des Datensatzes. Die einzelnen Schritte sind im HDevelop Beispiel `classify_pill_defects_deep_learning_1_preprocess.hdev` gezeigt.

1. Die Information, was auf welchem Bild des Trainings-Datensatzes gefunden werden soll, muss eingelesen und konvertiert werden. Dazu kann die Prozedur
   - `read_dl_dataset_classification`

   verwendet werden. Dabei wird ein Dictionary `DLDataset` erstellt, welches als Datenbank dient und alle notwendigen Informationen über die Daten abspeichert. Für mehr Information zu den Daten und wie sie übergeben werden, wird auf den unteren Absatz „Daten“ und das Kapitel Deep Learning / Modell verwiesen.
2. Die Daten von `DLDataset` sollen in drei getrennte Datensätze aufgeteilt werden. Dazu kann die Prozedur
   - `split_dl_dataset`

   verwendet werden. Die resultierende Aufteilung wird für jeden Sample Eintrag von `DLDataset` jeweils über den Schlüssel `split` gespeichert.
3. Einlesen eines vortrainierten Klassifikators mit dem Operator
   - `read_dl_model`.

   Dieser Operator wird auch dazu verwendet, um fertig trainierte Klassifikatoren, die mit `write_dl_model` geschrieben wurden, einzulesen.
   Der Klassifikator stellt mehrere Anforderungen an die Bilder, wie die Bilddimensionen und den Grauwertbereich. Die Defaultwerte sind in `read_dl_model` aufgelistet. Mit diesen Werten wurden die Netzwerke vortrainiert. Die Netzwerkarchitekturen erlauben verschiedene Bilddimensionen, welche über `set_dl_model_param` gesetzt werden können. Eine solche Änderung erfordert für gewisse Netzwerke aber ein Nachtrainieren, siehe `read_dl_model`. Die aktuellen Werte können mit

- get_dl_model_param

abgefragt werden.

4. Der Datensatz kann nun vorverarbeitet werden. Dazu kann die Prozedur

   - preprocess_dl_dataset

   verwendet werden. Für die Implementierung einer selbst erstellten Vorverarbeitung kann man sich an dieser Prozedur orientieren.
   Um diese Prozedur zu verwenden, müssen die Vorverarbeitungsparameter wie z.B. die Bildgröße festgelegt werden. Die Parameter und ihre Werte sind im Dictionary DLPreprocessParam zu speichern. Zur Erzeugung dieses Dictionaries kann die Prozedur

   - create_dl_preprocess_param

   verwendet werden. Es wird empfohlen, dieses Dictionary DLPreprocessParam abzuspeichern, um während der späteren Inferenz-Phase Zugang zu den Werten zu haben.

Trainieren des Modells Dieser Abschnitt behandelt das Trainieren des Klassifikators. Die einzelnen Schritte sind im HDevelop Beispiel classify_pill_defects_deep_learning_2_train.hdev gezeigt.

1. Setzen der Trainingsparameter und Abspeichern derselben im Dictionary *'TrainParam'*. Diese Parameter beinhalten:

   - die Hyperparameter. Für eine Übersicht wird auf das Kapitel Deep Learning verwiesen.
   - Parameter zur eventuellen Datenanreicherung (optional).
   - Parameter zur Evaluierung während des Trainings.
   - Parameter zur Visualisierung von Trainingsergebnissen.
   - Parameter zur Serialisierung.

   Für dies kann die Prozedur

   - create_dl_train_param

   verwendet werden.
2. Trainieren des Modells. Dafür kann die Prozedur

   - train_dl_model

   verwendet werden. Diese Prozedur benötigt:

   - das Handle des Modells, DLModelHandle
   - das Dictionary mit den Informationen über den Datensatz, DLDataset
   - das Dictionary mit den Trainingsparametern, *'TrainingParam'*
   - die Angabe, über wie viele Epochen trainiert werden soll.

   Wird die Prozedur train_dl_model verwendet, werden die Gesamt-Zielfunktion sowie optionale Evaluierungsmaße visualisiert.

Evaluierung des trainierten Modells Dieser Abschnitt behandelt das Evaluieren eines Klassifikators. Die einzelnen Schritte sind im HDevelop Beispiel classify_pill_defects_deep_learning_3_evaluate.hdev gezeigt.

1. Die Evaluierung kann bequem mit der Prozedur

   - evaluate_dl_model

   durchgeführt werden.
2. Das Dictionary EvaluationResults enthält die angefragten Evaluierungsmaße. Die Ergebnisse der Evaluierung können mit der Prozedur

   - dev_display_classification_evaluation

   visualisiert werden.
3. Eine Heatmap kann für angegebene Samples generiert werden mit

   - dem Operator gen_dl_model_heatmap
   - der Prozedur gen_dl_model_classification_heatmap

Inferenz auf neuen Bildern Dieser Abschnitt behandelt die Inferenz auf neuen Bildern durch einen Klassifikator. Die einzelnen Schritte sind im HDevelop Beispiel `classify_pill_defects_deep_learning_4_infer.hdev` gezeigt.

1. Setzen der Parameter, wie z.B. *'batch_size'* über den Operator
   - `set_dl_model_param`.
2. Generieren des Dictionaries `DLSample` für jedes zu inferierende Bild. Dafür kann die Prozedur
   - `gen_dl_samples_from_images`

   verwendet werden.
3. Jedes Bild muss auf dieselbe Art und Weise wie die Trainingsbilder vorverarbeitet werden. Es wird empfohlen, dafür die Prozedur
   - `preprocess_dl_samples`

   zu verwenden. Wurde während der Vorverarbeitung das Dictionary `DLPreprocessParam` gespeichert, kann es direkt als Eingabe verwendet werden, um die Parameterwerte festzulegen.
4. Anwenden des Modells über den Operator
   - `apply_dl_model`.
5. Die Resultate können aus dem Dictionary *'DLResultBatch'* abgerufen werden.

Daten

Es wird zwischen den Daten für Training und Evaluierung versus Daten für Inferenz unterschieden. Letztere bestehen ausschließlich aus Bildern. Für Erstere ist jedoch bereits bekannt zu welcher Klasse die Bilder gehören. Die entsprechenden Informationen werden über die Labels zur Verfügung gestellt.

Ein Grundgedanke der Datenhandhabung: Das Modell interagiert mit den Daten über Dictionaries. Dies bedeutet, das Modell erhält die Eingabedaten über das Dictionary `DLSample` und gibt die Ausgabe über das Dictionary `DLResult`, bzw. `DLTrainResult` zurück. Für weitere Informationen zur Datenhandhabung wird auf das Kapitel Deep Learning / Modell verwiesen.

Daten für das Training und die Evaluierung Der Datensatz besteht aus Bildern und den dazugehörigen Informationen. Damit das Modell diese verarbeiten kann, müssen sie bestimmte Anforderungen erfüllen. Informationen zu den Bildanforderungen finden sich im unteren Abschnitt „Bilder“.

Die Trainingsdaten werden dazu benötigt, um den Klassifikator für eine spezifische Aufgabe zu trainieren. Mit Hilfe dieser Daten kann der Klassifikator lernen, welche Klassen zu unterscheiden sind und wie deren Vertreter aussehen. Bei der Klassifikation wird das Bild als Ganzes klassifiziert. Daher bestehen die Trainingsdaten aus Bildern und deren Ground Truth Labels, also die Klasse zu dem das Bild gehört. Zu beachten ist hierbei, dass die Bilder möglichst repräsentativ für die spätere Aufgabe sein sollten. Es gibt verschiedene Möglichkeiten, wie die Ground Truth Klassen als Label gespeichert und abgerufen werden können. Auf folgende Weisen können die Ground Truth Labels eines Bildes von der Prozedur `read_dl_dataset_classification` gelesen werden:

- Als Name des letzten Ordners, der das Bild enthält
- Als Dateiname.

Zum Trainieren eines Klassifikators wird eine Technik namens Transferlernen verwendet (siehe das Kapitel Deep Learning). Die Anzahl benötigter Bilder ist dabei geringer. Während der Klassifikator generell verlässlicher ist, wenn er mit einem größeren Datensatz trainiert wurde, hängt die Anzahl benötigter Bilder auch von der Komplexität der Aufgabe ab. Es sollten außerdem genug Trainingsdaten vorliegen um diese in die drei Untermengen aufzuteilen, welche idealerweise von einander unabhängig und gleich verteilt sind (siehe Abschnitt „Daten“ im Kapitel Deep Learning).

Bilder Unabhängig von der Anwendung stellt das Netzwerk Anforderungen an die Bilder, wie z.B. die Bilddimensionen. Die spezifischen Werte hängen vom Netzwerk selbst ab, für die spezifischen Werte der verschiedenen Netzwerke siehe die Dokumentation von `read_dl_model`. Für ein eingelesenes Netzwerk können die Werte mit `get_dl_model_param` abgefragt werden. Um diese Anforderungen zu erfüllen, müssen die Bilder eventuell vorverarbeitet werden. Die Standard-Vorverarbeitung für den ganzen Datensatz und damit auch der Bilder kann mit Hilfe der Prozedur `preprocess_dl_dataset` durchgeführt werden, bzw. `preprocess_dl_samples` für ein einzelnes Sample. Für eine selbst erstellte Vorverarbeitung liefert diese Prozedur eine Anleitung für die Implementierung.

Netzwerk Ausgabe Als Trainingsausgabe gibt das Modell ein Dictionary `DLTrainResult` mit dem aktuellen Wert der Gesamt-Zielfunktion sowie den Werten aller weiteren im Modell enthaltenen Zielfunktionen.

Als Inferenz- und Evaluierungs-Ausgabe gibt das Netzwerk für jedes Bild ein Dictionary `DLResult` zurück. Im Falle der Klassifikation beinhaltet dieses Dictionary für jedes Bild ein Tupel mit den Konfidenzwerten zu jeder unterschiedenen Klasse in absteigender Reihenfolge sowie ein Tupel mit den dazugehörigen Klassen-IDs.

Interpretation der Klassifikationsergebnisse

Bei der Klassifikation eines Bildes wird eine Menge von Konfidenzwerten erhalten, welche die Zugehörigkeit des Bildes zu jeder Klasse angibt. Es können außerdem die folgenden Werte berechnet werden.

Konfusionsmatrix, Precision, Recall, und F-score Bei der Klassifikation werden ganze Bilder klassifiziert. Daher sind die Instanzen der Konfusionsmatrix Bilder. Für weitere Informationen zur Konfusionsmatrix wird auf das Kapitel Deep Learning verwiesen.

Eine Konfusionsmatrix kann mit Hilfe der Prozeduren `gen_confusion_matrix` und `gen_interactive_confusion_matrix` generiert werden. Die interaktive Prozedur ermöglicht es, Beispiele einer spezifischen Kategorie auszuwählen, funktioniert aber nicht mit exportiertem Code.

Mehrere Bewertungsmaße können aus der Konfusionsmatrix abgeleitet werden: Die Precision ist das Verhältnis aller korrekt vorhergesagten Positivinstanzen zu allen positiv vorhergesagten Instanzen. Es stellt ein Maß dafür dar, wie viele positive Vorhersagen tatsächlich positiv sind.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Der Recall (auch als "True-Positive-Rate"bezeichnet) ist das Verhältnis aller korrekt vorhergesagten Positivinstanzen zu allen Ground Truth Positivinstanzen. Es ist also ein Maß dafür, wie viele tatsächlich positive Instanzen korrekt vorhergesagt wurden.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Ein Klassifikator mit hohem Recall, aber geringer Precision findet die meisten Elemente der Positivinstanzen einer Klasse (also die tatsächlichen Instanzen der Klasse), klassifiziert für diese Klasse allerdings auch viele falsche Instanzen. Ein Klassifikator mit hoher Precision und geringem Recall ist genau das Gegenteil, er klassifiziert nur wenige Instanzen als positiv, die meisten dieser Vorhersagen sind aber korrekt. Ein idealer Klassifikator mit hoher Precision und hohem Recall wird viele Instanzen als positiv mit hoher Genauigkeit klassifizieren.

Um dies mit einer einzigen Zahl auszudrücken, wird der F1-Score berechnet, das harmonische Mittel von Precision und Recall. Dies ist ein Maß für die Genauigkeit des Klassifikators.

$$\text{F1-score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Für das Beispiel mit der im Kapitel Deep Learning gezeigten Konfusionsmatrix ergibt sich für die Klasse 'apple' die Precision: 1.00 (= 68/(68+0+0)), der Recall: 0.74 (= 68/(68+21+3)) und der F1-Score: 0.85 (=2*(1.0*0.74)/(1.0+0.74)).

8.4 Modell

Dieses Kapitel erklärt das generelle Konzept des Deep Learning-Modells in HALCON und die Handhabung der Daten.

Ein Deep Learning-Modell ist in HALCON eine interne Repräsentation eines Neuronalen Netzwerkes. Jedes Neuronale Netzwerk hat seine Architektur, welche seine Funktion definiert und damit bestimmt, für welche Aufgaben es verwendet werden kann. Dabei sind verschiedene Netzwerk-Architekturen für eine Funktionalität möglich. Momentan sind folgende Funktionalitäten in HALCON als Modell implementiert:

- Anomalieerkennung, siehe Deep Learning / Anomalieerkennung.
- Klassifikation, siehe Deep Learning / Klassifikation.
- Objektdetektion, siehe Deep Learning / Objektdetektion.
- Semantische Segmentierung, siehe Deep Learning / Semantische Segmentierung.

Jede Funktionalität ist über ihren Modelltypen bezeichnet. Für die implementierten Methoden können weitere Informationen zum spezifischen Ablauf, den Daten-Anforderungen und Evaluierungsmöglichkeiten in den jeweiligen Kapiteln gefunden werden. Informationen zu Deep Learning (DL) im Allgemeinen befinden sich im Kapitel Deep Learning.

In diesem Kapitel finden sich Informationen, welche Daten ein DL Modell benötigt und zurück gibt sowie auch wie diese Daten übergeben werden.

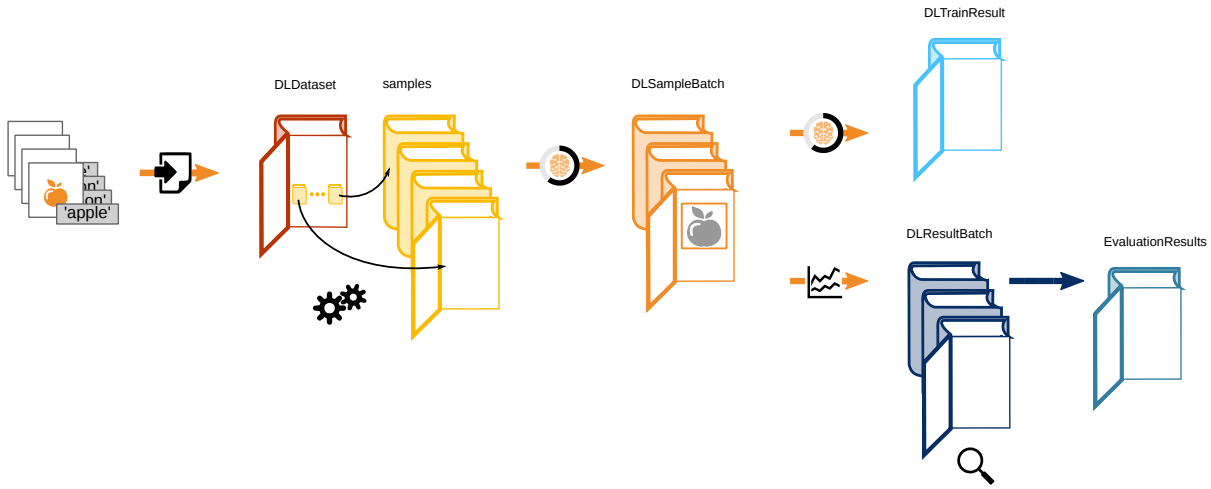
Daten

Bei Deep Learning Anwendungen werden Daten verschiedener Art unterschieden. Grob gesagt sind dies die Rohbilder mit möglichen Annotationen, für das Modell passend verarbeitete Daten und Ausgabedaten.

Bevor die verschiedenen Daten und die Einträge spezifischer Dictionaries erläutert werden, wird ein grober Überblick gegeben, wie die Daten zusammenhängen. Dabei beziehen sich Farben und Symbole auf die darunter gegebenen schematischen Darstellungen.

Grob gesagt, beginnt die Datenstruktur beim Training und der Evaluierung mit den rohen Bildern und ihren Ground Truth Annotationen (graue Rahmen). Mit den eingelesenen Daten werden die folgenden Dictionaries erstellt: Ein Dictionary `DLDataset` (rot) dient als Datenbank und verweist für jedes Eingabebild auf ein spezifisches Dictionary (gelb). Jedes Dictionaries `DLSample` (orange) enthält die Daten eines Samples in der Art wie das Modell sie als Eingabe benötigt. Ein Batch von `DLSamples` wird dem Modell bei der Evaluierung als `DLSampleBatch` übergeben. Bei der Evaluierung wird ein `DLResultBatch` zurückgegeben, ein Tupel von Dictionaries `DLResult` (dunkelblau), jeweils eines pro Sample. Diese Resultate werden für die Evaluierungsresultate in `EvaluationResults` (türkis) benötigt. Beim Training werden die Trainingsresultate (z.B. Losswerte) im Dictionary `DLTrainResult` (hellblau) zurückgegeben. Die wichtigsten Schritte, in welchen die Dictionaries erstellt oder modifiziert werden:

- Einlesen der Rohdaten (Symbol: Papier mit Pfeil)
- Vorverarbeitung der Daten (Symbol: Zahnräder)
- Training des Netzwerkes (Symbol: transparentes Gehirn in Kreissegment)
- Evaluierung des Modells (Symbol: Graph)
- Evaluierung des Samples (Symbol: Lupe)



Schematische Darstellung der Datenstruktur beim Training und der Evaluierung.

Bei der Inferenz sind keine Annotationen notwendig. Somit startet die Datenstruktur mit den rohen Bilder (grauer Rahmen). Das Dictionary `DLSample` (orange) enthält die Daten eines Samples in der Art wie das Modell sie als Eingabe benötigt. Die Resultate für ein Sample werden im Dictionary `DLResult` (dunkelblau) zurückgegeben. Die wichtigsten Schritte, in welchen die Dictionaries erstellt oder modifiziert werden:

- Einlesen der Rohdaten (Symbol: Papier mit Pfeil)
- Vorverarbeitung der Daten (Symbol: Zahnräder)
- Inferenz (Symbol: Gehirn in Kreis)
- Evaluierung des Samples (Symbol: Lupe)



Schematische Darstellung der Datenstruktur bei der Inferenz.

Damit das Modell die Daten verarbeiten kann, müssen diese gewisse Konventionen einhalten bezüglich welche Informationen benötigt werden und wie die Daten dem Modell übergeben werden. Wie aus den obigen Schemata ersichtlich, werden die Daten in HALCON über Dictionaries übergeben.

Die einzelnen Dictionaries, wie sie erstellt werden können und ihre Einträge werden im Folgenden genauer erläutert. Um zu markieren, welche Methoden (M) den jeweiligen Eintrag nutzen, verwenden wir folgende Abkürzungen:

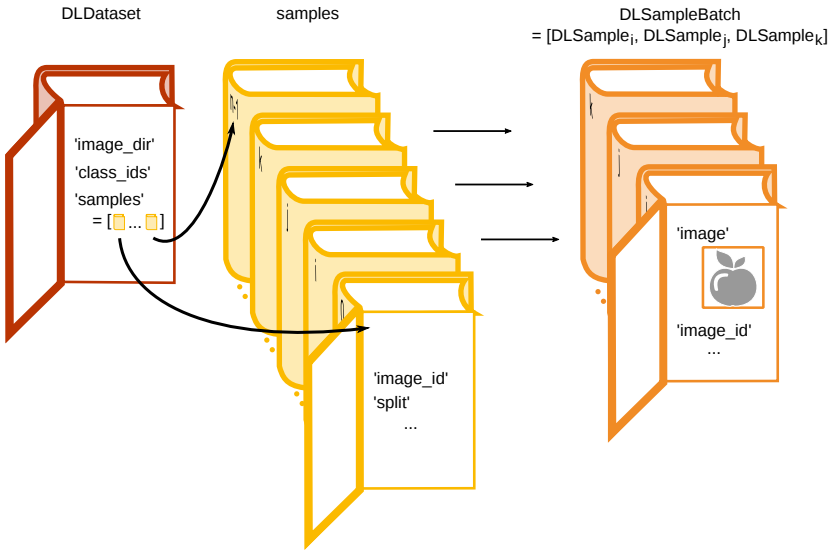
- 'Any': Jede Methode
- 'AD': Anomalieerkennung
- 'CL': Klassifikation
- 'OD': Objektdetektion

  Falls der Eintrag nur auf gewisse *'instance_type'* zutrifft, ist dieser spezifiziert mit 'r1': *'rectangle1'*, 'r2': *'rectangle2'*
- 'SE': Semantische Segmentierung

Die Einträge, die nur auf einzelne Methoden zutreffen, werden in den jeweiligen Kapiteln genauer beschrieben.

Eingabedaten beim Training und der Evaluierung Der Datensatz besteht aus Bildern und den dazugehörigen Informationen. Damit das Modell diese verarbeiten kann, müssen sie bestimmte Anforderungen erfüllen. Informationen zu den Bildanforderungen finden sich im unteren Abschnitt „Bilder".

Die Informationen zu den Bildern und dem Datensatz werden im Dictionary `DLDataset` vermerkt. Dabei dient dieses Dictionary als Datenbank. Genauer gesagt, werden darin die generellen Informationen über den Datensatz gespeichert sowie die Dictionaries der einzelnen Samples unter dem Schlüssel `samples` gesammelt. Werden die jeweiligen Bilddaten benötigt, wird für jedes erforderte Bild ein Dictionary `DLSample` erstellt (oder gelesen, falls dieses bereits existiert). Der Zusammenhang zwischen diesen Dictionaries ist in der folgenden Darstellung illustriert.



Schematische Darstellung der verschiedenen Dictionaries des Datensatzes, welche während des Trainings und der Evaluierung verwendet werden. Zu Darstellungszwecken ist die `BatchSize` auf drei gesetzt und werden nur wenige Einträge aufgelistet. In diesem Beispiel werden aus den insgesamt n Samples drei zufällig ausgewählt, jene mit den Indizes i,j und k. Die entsprechenden `DLSample` werden erstellt und im Tupel `DLSampleBatch` vereint.

Im Folgenden werden die einzelnen Dictionaries mit ihren Schlüssel/Wert-Paaren genauer erklärt.

`DLDataset` Das Dictionary `DLDataset` dient als Datenbank. Es speichert die generellen Informationen zum Datensatz und sammelt die Dictionaries der individuellen Samples. Dabei sind keine ikonischen Daten in `DLDataset` enthalten, aber die Pfade zu den entsprechenden Bildern. Das Dictionary `DLDataset` von den Trainings- und Evaluierungs- Prozeduren verwendet. Es ist für das Modell selbst nicht notwendig, aber seine Erstellung wird stark empfohlen. Dieses Dictionary wird direkt erzeugt, wenn die Daten mit Hilfe des MVTec Deep Learning Tools gelabelt werden. Alternativ wird es erstellt, wenn zum Einlesen der Daten eine der folgenden Prozeduren verwendet wird:

- `read_dl_dataset_anomaly` (Anomalieerkennung)
- `read_dl_dataset_classification` (Klassifikation)
- `read_dl_dataset_from_coco` (Objektdetektion mit *'instance_type' = 'rectangle1'*)
- `read_dl_dataset_segmentation` (Semantische Segmentierung).

Bezüglich der genauen Datenanforderungen um diese Prozeduren zu verwenden wird auf die jeweilige Prozedurendokumentation verwiesen. Wird `DLDataset` auf anderen Wegen erstellt, muss es zumindest die Einträge beinhalten, die in der unteren Erklärung nicht mit einer Zahl markiert sind. Während der Vorverarbeitung des Datensatzes werden dem Dictionary `DLDataset` die weiteren Einträge von den jeweiligen Prozeduren hinzugefügt.

Vom Modell abhängig kann das Dictionary die folgenden Einträge haben:

`image_dir`: Any
Basispfad zu allen Bildern.
Format: String

`dlsample_dir`: Any [1]
Basispfad zu allen Sample Dateien (sofern vorhanden).
Format: String

`class_names`: Any
Namen aller zu unterscheidenden Klassen.
Format: Tupel von Strings

`class_ids`: Any
IDs aller zu unterscheidenden Klassen (Wertebereich: 0-65534).
Format: Tupel von ganzzahligen Werten

`preprocess_param`: Any [1]
Alle während der Vorverarbeitung verwendeten Parameter.
Format: Dictionary

anomaly_dir: AD
Basispfad zu allen Anomalieregionen (Regionen, die eine Anomalie auf dem Bild kennzeichnen).
Format: String

class_weights: CL, SE [1]
Gewichte der verschiedenen Klassen.
Format: Tupel von Gleitkommazahlen

segmentation_dir: SE
Basispfad zu allen Segmentierungsbildern.
Format: String

Dieses Dictionary wird direkt erzeugt, wenn die Daten mit Hilfe des MVTec Deep Learning Tools gelabelt werden. Ebenso wird es von den oben erwähnten Prozeduren erstellt. Die mit [1] markierten Einträge werden von den Vorverarbeitungsprozeduren eingetragen.

samples
Der Schlüssel `samples` des Dictionary `DLDataset` hat als Wert ein Tupel von Dictionaries, eines für jedes Sample des Datensatzes. Diese Dictionaries beinhalten die Information der jeweiligen Samples und haben die folgenden Einträge:

image_file_name: Any
Dateiname des Bildes und sein Pfad relativ zu `image_dir`.
Format: String

image_id: Any
Eindeutige Bild-ID (Zeichenkodierungsformat: UINT8).
Format: Ganzzahl

split: Any [2]
Gibt den bei der Aufteilung zugeordneten Datensatz an (*'train'*,*'validation'*,*'test'*).
Format: String

dlsample_file_name: Any [3]
Dateiname des entsprechenden Dictionary `DLSample` und sein Pfad relativ zu `dlsample_dir`.
Format: String

anomaly_file_name: AD
Optional. Pfad zu den Dateien mit den Ground Truth Anomalieregionen (relativ zu `anomaly_dir`).
Format: String

anomaly_label: AD
Ground Truth Anomalie Label (als Name aus `class_names`).
Format: String

image_label_id: CL
Ground Truth Label des Bildes (als ID aus `class_ids`).
Format: Tupel von Ganzzahlen

bbox_label_id: OD
Ground Truth Label für den Bildausschnitt innerhalb des umschließenden Rechtecks (als ID aus `class_ids`).
Format: Tupel von Ganzzahlen

bbox_row1: OD:r1 [4]
Ground Truth umschließendes Rechteck: Obere linke Ecke, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col1: OD:r1 [4]
Ground Truth umschließendes Rechteck: Obere linke Ecke, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_row2: OD:r1 [4]
Ground Truth umschließendes Rechteck: Untere rechte Ecke, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col2: OD:r1 [4]
Ground Truth umschließendes Rechteck: Untere rechte Ecke, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

coco_raw_annotations: OD:r1
Optional. Für jede `bbox_label_id` innerhalb dieses Bildes enthält dieser Wert ein Dictionary mit allen unverarbeiteten COCO Annotations-Informationen.

Format: Tupel von Dictionaries

bbox_row: OD:r2 [4]
Ground Truth umschließendes Rechteck: Mittelpunkt, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col: OD:r2 [4]
Ground Truth umschließendes Rechteck: Mittelpunkt, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_phi: OD:r2 [4]
Ground Truth umschließendes Rechteck: Winkel phi.
Format: Tupel von Gleitkommazahlen

bbox_length1: OD:r2 [4]
Ground Truth umschließendes Rechteck: Halbe Kantenlänge 1.
Format: Tupel von Gleitkommazahlen

bbox_length2: OD:r2 [4]
Ground Truth umschließendes Rechteck: Halbe Kantenlänge 2.
Format: Tupel von Gleitkommazahlen

segmentation_file_name: SE
Dateiname des Ground Truth Segmentierungsbildes und sein Pfad relativ zu `segmentation_dir`.
Format: String

Diese Dictionaries sind Teil von `DLDataset` und werden als solche zugleich erstellt. Ausnahme sind dabei die markierten Einträge der Tabelle, [2]: Die Prozedur `split_dl_dataset` fügt den Eintrag `split` hinzu, [3]: Die Prozedur `preprocess_dl_samples` fügt den Eintrag `dlsample_file_name` hinzu. [4]: Für die Parameter der umschließenden Ground Truth Rechtecke (BBoxGT) werden pixel-zentrierte, subpixel-genaue Koordinaten verwendet.

DLSample Das Dictionary `DLSample` dient als Eingabewert des Modells. Für einen Batch werden sie als Einträge des Tupels `DLSampleBatch` übergeben.

Ein solches Dictionary `DLSample` wird für jedes Bild von der Prozedur `gen_dl_samples` aus den Daten in `DLDataset` erstellt. Es enthält alle Ground Truth Annotationen eines Bildes. Wird die Vorverarbeitung mit den Standard-Prozedur `preprocess_dl_samples` durchgeführt, werden sie automatisch von dieser erstellt. Es gilt zu beachten, dass die Vorverarbeitung zu Aktualisierungen des entsprechenden `DLSample` führen kann.

Das Dictionary `DLSample` hat folgende Einträge:

image: Any
Eingabebild
Format: Bild

image_id: Any
Eindeutige Bild-ID (wie in `DLDataset`).
Format: Ganzzahl

anomaly_ground_truth: AD
Bild oder Region, aus `anomaly_file_name`.
Format: Bild oder Region

anomaly_label: AD
Ground Truth Anomalie Label auf Bildebene (als Name aus `class_names`).
Format: String

anomaly_label_id: AD
Ground Truth Anomalie Label auf Bildebene (als ID aus `class_ids`).
Format: Ganzzahl

image_label_id: CL
Ground Truth Label des Bildes (als ID aus `class_ids`).
Format: Tupel von Ganzzahlen

bbox_label_id: OD
Ground Truth Label für den Bildausschnitt innerhalb des umschließenden Rechtecks (als ID aus `class_ids`).
Format: Tupel von Ganzzahlen

bbox_row1: OD:r1 [4]
Ground Truth umschließendes Rechteck: Obere linke Ecke, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col1: OD:r1 [4]
Ground Truth umschließendes Rechteck: Obere linke Ecke, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_row2: OD:r1 [4]
Ground Truth umschließendes Rechteck: Untere rechte Ecke, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col2: OD:r1 [4]
Ground Truth umschließendes Rechteck: Untere rechte Ecke, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_row: OD:r2 [4]
Ground Truth umschließendes Rechteck: Mittelpunkt, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col: OD:r2 [4]
Ground Truth umschließendes Rechteck: Mittelpunkt, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_phi: OD:r2 [4]
Ground Truth umschließendes Rechteck: Winkel phi.
Format: Tupel von Gleitkommazahlen

bbox_length1: OD:r2 [4]
Ground Truth umschließendes Rechteck: Halbe Kantenlänge 1.
Format: Tupel von Gleitkommazahlen

bbox_length2: OD:r2 [4]
Ground Truth umschließendes Rechteck: Halbe Kantenlänge 2.
Format: Tupel von Gleitkommazahlen

segmentation_image: SE
Bild mit den Ground Truth Segmentierungen, eingelesen von `segmentation_file_name`.
Format: Bild

weight_image: SE [5]
Bild mit den Pixel-Gewichten.
Format: Bild

Diese Dictionaries werden von der Prozedur `gen_dl_samples` erzeugt. Eine Ausnahme stellt der markierte Eintrag dar, [5]: wird von der Prozedur `gen_dl_segmentation_weights` erstellt. [4]: Für die Parameter der Ground Truth umschließenden Rechtecke werden pixel-zentrierte, subpixelgenaue Koordinaten verwendet.

Falls die Dictionaries `DLSample` gespeichert werden sollen, kann die Prozedur `write_dl_samples` verwendet werden. Eingelesen können die gespeicherten `DLSample` Dictionaries über die Prozedur `read_dl_samples`.

Eingabedaten bei der Inferenz Die Eingabedaten bei der Inferenz bestehen aus den blanken Bildern. Damit das Modell diese verarbeiten kann, müssen sie bestimmte Anforderungen erfüllen. Informationen zu den Bildanforderungen finden sich im unteren Abschnitt „Bilder“.

Das Modell ist so konzipiert, dass alle Daten über ein Dictionary `DLDataset` übergeben wird. Für die Inferenz beinhaltet ein solches Dictionary nur das Bild und wird von der Prozedur `gen_dl_samples_from_images` erstellt. Diese Dictionaries können einzeln oder als Einträge des Tupels übergeben werden `DLSampleBatch`.

Ausgabedaten beim Training Die Trainingsresultate werden im Dictionary `DLTrainResult` gesammelt. Die spezifischen Einträge hängen vom Modell und damit vom verwendeten Operator ab (für weitere Informationen, siehe die Dokumentation des entsprechendes Operators):

CL, OD, SE:
Der Operator `train_dl_model_batch` gibt folgende Einträge zurück:

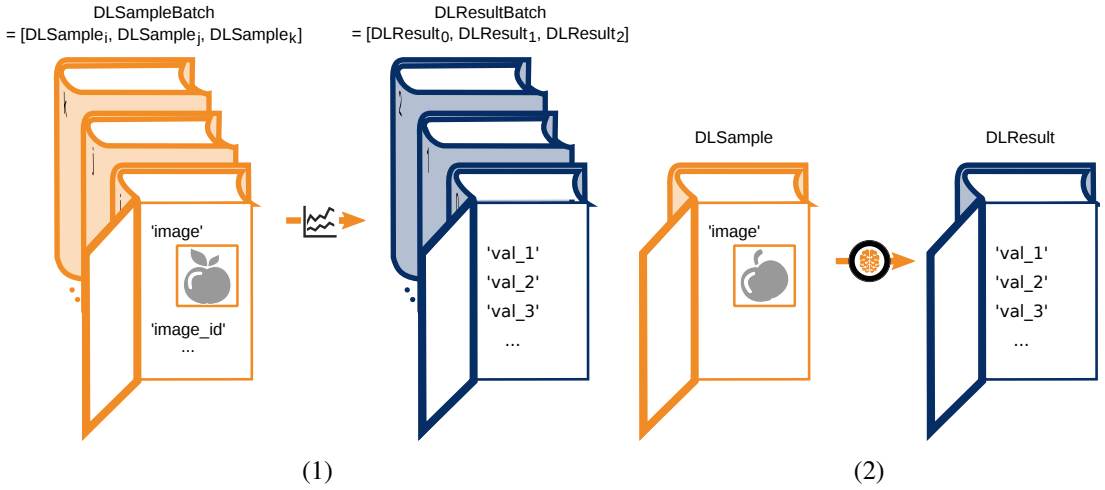
- `total_loss`
- weitere mögliche Zielfunktionen des Modells

AD:
Der Operator `train_dl_model_anomaly_dataset` gibt folgende Einträge zurück:

- `final_error`
- `final_epoch`

Als Ausgabe des Operators `train_dl_model_batch` gibt das Modell ein Dictionary `DLTrainResult` zurück. In diesem Dictionary befinden sich der aktuelle Wert der Gesamt-Zielfunktion unter dem Schlüssel `total_loss` sowie die Werte aller weiteren im Modell enthaltenen Zielfunktionen.

Ausgabedaten bei der Inferenz und der Evaluierung Als Ausgabe des Operators `apply_dl_model` gibt das Modell für jedes Sample ein Dictionary `DLResult`, siehe die unten folgende Darstellung. Die Evaluierung basiert auf diesen Resultaten und den dem Bild zugehörigen Annotationen. Die Evaluierungsresultate werden im Dictionary `EvaluationResults` gespeichert.



Schematische Darstellung der Dictionaries, welche als Modell-Eingabe dienen: (1) Evaluierung: `DLSample` beinhaltet das Bild sowie Informationen über das Bild und seinen Inhalt. Diese Daten dienen als Basis der Evaluierung. Zu Darstellungszwecken ist die `BatchSize` auf drei gesetzt, zudem sind nur wenige Einträge aufgelistet. (2) Inferenz: `DLSample` beinhaltet das Bild. Diese Dictionaries können einzeln oder im Tupel übergeben werden.

Abhängig vom Modell-Typ kann dieses Dictionary die folgenden Einträge haben:

`anomaly_image`: AD
Grauwertbild mit Scores als Pixelwerte, die angeben wie wahrscheinlich der entsprechende Pixel im Eingabebild zu einer Anomalie gehört.
Format: Bild

`anomaly_score`: AD
Score der Anomalie aufgrund der Werte in `anomaly_image`.
Format: Gleitkommazahl

`classification_class_ids`: CL
IDs der vorhergesagten Klassen des Bildes sortiert nach Konfidenzwerten.
Format: Tupel von Ganzzahlen

`classification_class_names`: CL
Vorhergesagte Klassennamen des Bildes sortiert nach Konfidenzwerten.
Format: Tupel von Strings

`classification_confidences`: CL
Konfidenzwerte für die Vorhersage des Bildes für jede Klasse.
Format: Tupel von Gleitkommazahlen

`bbox_class_id`: OD
Vorhergesagte Klasse für das umschließende Rechteck (als ID aus `class_ids`).
Format: Tupel von Ganzzahlen

bbox_confidence: OD
Konfidenzwerte für die Vorhersage des umschließenden Rechtecks.
Format: Tupel von Gleitkommazahlen

bbox_row1: OD:r1 [6]
Inferiertes umschließendes Rechteck: Obere linke Ecke, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col1: OD:r1 [6]
Inferiertes umschließendes Rechteck: Obere linke Ecke, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_row2: OD:r1 [6]
Inferiertes umschließendes Rechteck: Untere rechte Ecke, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col2: OD:r1 [6]
Inferiertes umschließendes Rechteck: Untere rechte Ecke, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_row: OD:r2 [6]
Inferiertes umschließendes Rechteck: Mittelpunkt, Zeilenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_col: OD:r2 [6]
Inferiertes umschließendes Rechteck: Mittelpunkt, Spaltenkoordinate.
Format: Tupel von Gleitkommazahlen

bbox_phi: OD:r2 [6]
Inferiertes umschließendes Rechteck: Winkel phi.
Format: Tupel von Gleitkommazahlen

bbox_length1: OD:r2 [6]
Inferiertes umschließendes Rechteck: Halbe Kantenlänge 1.
Format: Tupel von Gleitkommazahlen

bbox_length2: OD:r2 [6]
Inferiertes umschließendes Rechteck: Halbe Kantenlänge 2.
Format: Tupel von Gleitkommazahlen

segmentation_image: SE
Bild mit dem Segmentierungsresultat.
Format: Bild

segmentation_confidence: SE
Bild mit den Konfidenzwerten des Segmentierungs-Resultats.
Format: Bild

[6]: Für die Parameter der inferierten umschließenden Rechtecke werden pixel-zentrierte, subpixel-genaue Koordinaten verwendet.

Für weitere Informationen zu den Ausgabewerten wird auf das Kapitel der entsprechenden Methode verwiesen, z.B. Deep Learning / Semantische Segmentierung.

Bilder Unabhängig von der Anwendung stellt das Netzwerk Anforderungen an die Bilder. Die spezifischen Werte hängen vom Netzwerk selbst ab und können mit `get_dl_model_param` abgefragt werden. Um diese Anforderungen zu erfüllen, müssen die Bilder evt. vorverarbeitet werden. Die Standard-Vorverarbeitung für den ganzen Datensatz und damit auch der Bilder kann mit Hilfe der Prozedur `preprocess_dl_samples` durchgeführt werden. Für eine selbst erstellte Vorverarbeitung liefert diese Prozedur eine Anleitung für die Implementierung.

add_dl_pruning_batch (: : DLModelHandleToPrune, DLPruningHandle, DLSampleBatch :)

Berechnen von Scores für das Pruning eines Deep Learning-Modells.

add_dl_pruning_batch berechnet Scores für das Pruning des Deep Learning-Modells DLModelHandleToPrune. Genauer gesagt werden die Scores für den Batch gegebener Bilder, DLSampleBatch, berechnet und bei jedem Aufruf von add_dl_pruning_batch im Pruning-Daten-Handle DLPruningHandle intern akkumuliert.

Der Parameter DLModelHandleToPrune gibt das Deep Learning-Modell an, das für die Berechnung verwendet wird. Es ist zu beachten, dass add_dl_pruning_batch nur Deep Learning-Modelle vom Typ *'classification'* unterstützt.

Der Parameter DLPruningHandle gibt das Handle zu den Pruning-Daten an. Dieses Handle wird verwendet um Informationen wie z.B. die Scores oder den Pruning Modus zu übergeben. Siehe create_dl_pruning für weitere Informationen zu den implementierten Pruning Modi.

Der Parameter DLSampleBatch gibt den Batch mit Eingabebildern an, auf welchen die Scores berechnet werden. Beachten Sie, dass die Anzahl von Bildern in einem DLSampleBatch Tupel gleich der *'batch_size'* des Modells sein muss.

Das Konzept von Deep Learning ist in der Einleitung zu Kapitel Deep Learning beschrieben.

Parameter

▷ **DLModelHandleToPrune** (input_control) ..... dl_model ⇝ *handle*
Handle des zu reduzierenden Deep Learning-Modells.

▷ **DLPruningHandle** (input_control) ..... dl_pruning ⇝ *handle*
Handle mit den Pruning-Daten.

▷ **DLSampleBatch** (input_control) ..... dict-array ⇝ *handle*
Tupel von Dictionaries mit Eingabebildern.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_dl_model, create_dl_pruning, set_dl_pruning_param

Nachfolger

get_dl_pruning_param, gen_dl_pruned_model

Modul

Deep Learning Training

apply_dl_model (: : DLModelHandle, DLSampleBatch, Outputs : DLResultBatch)

Anwenden des Deep Learning-basierten Netzwerkes zur Inferenz auf einer Menge an Bildern.

apply_dl_model wendet das durch DLModelHandle gegebene Deep Learning-basierte Netzwerk auf ein Batch von Eingabebildern an, die über das Tupel von Dictionaries DLSampleBatch übergeben werden. Der Operator gibt DLResultBatch zurück, ein Tupel mit einem Ergebnis-Dictionary DLResult pro Eingabebild.

Für weitere Informationen zum Konzept und den Dictionaries des Deep Learning-Modells in HALCON siehe das Kapitel Deep Learning / Modell.

Um das Netzwerk auf Bilder anwenden zu können, müssen diese über ein Tupel von Dictionaries DLSampleBatch übergeben werden. Dabei bezieht sich jedes Dictionary auf ein einzelnes Bild. Mit Hilfe der Prozedur gen_dl_samples_from_images kann solch ein Dictionary passend erstellt werden. Das Tupel DLSampleBatch kann eine beliebige Anzahl Dictionaries enthalten. Der Operator apply_dl_model verarbeitet stets einen Batch mit bis zu *'batch_size'* Bildern simultan. Falls das Tupel mehr Bilder enthält, iteriert apply_dl_model intern über die notwendige Anzahl an Batches um alle Bilder zu verarbeiten. Für ein DLSampleBatch mit weniger als *'batch_size'* Bildern wird das Tupel zu einem vollen Batch aufgefüllt, was also bedeutet, dass die Laufzeit zur Verarbeitung eines solchen Batches unabhängig davon ist, ob dieser vollständig gefüllt ist oder nur ein einzelnes Bild enthält. Dies heißt auch, dass das Netzwerk stets den vollen Speicher

benötigt, selbst wenn in einem Operatorenaufruf weniger als *'batch_size'* Bilder verarbeitet werden sollen. Der aktuelle Wert von *'batch_size'* kann mittels `get_dl_model_param` abgefragt werden.

Es ist zu beachten, dass die Bilder eventuell vorverarbeitet werden müssen um die die Netzwerkanforderungen zu erfüllen, bevor sie dem Operator `apply_dl_model` übergeben werden können. Mit `get_dl_model_param` können die aktuellen Anforderungen des Netzwerks wie z.B. die Bilddimensionen abgefragt werden. Die Prozedur `preprocess_dl_dataset` gibt eine Anleitung für die Implementierung dieser Vorverarbeitung.

Die Resultate werden in `DLResultBatch` zurückgegeben, einem Tupel mit einem Dictionary `DLResult` für jedes Eingabebild. Für weitere Informationen zu den Ausgabe-Dictionaries `DLResultBatch` und dessen Schlüssel siehe das Kapitel Deep Learning / Modell. In `Outputs` kann spezifiziert werden, welche Daten vom Netzwerk in den Ergebnis-Dictionaries `DLResultBatch` ausgegeben werden. Dabei kann `Outputs` ein einzelner String, ein Tupel von Strings, oder ein leeres Tupel, mit dem alle möglichen Ausgaben abgerufen werden können, sein. Die Werte sind abhängig vom Modelltyp des verwendeten Netzwerks:

Modell mit *'type'='anomaly_detection'*

- `Outputs`=*'[]'*: `DLResult` enthält ein Bild, in dem jedes Pixel den Score des entsprechenden Pixels des Eingabebildes erhält. Zusätzlich enthält es einen Score für das Bild als Ganzes.

Modell mit *'type'='classification'*

- `Outputs`=*'[]'*: `DLResult` enthält ein Tupel mit den Konfidenzwerten in absteigender Reihenfolge sowie weitere Tupel mit den Klassennamen und Klassen-IDs in entsprechender Sortierung.

Modell mit *'type'='detection'*

- `Outputs`=*'[]'*: `DLResult` enthält für alle Modell-Ebenen die Koordinaten der umschließenden Rechtecke, sowie die geschätzten Klassen und deren Konfidenzwerte.
- `Outputs`= *'[bboxhead + level + _predictIon, classhead + level + _prediction]'*, wobei *'level'* die ausgewählte Ebene angibt und einen Wert zwischen *'min_level'* und *'max_level'* annehmen kann: `DLResult` enthält für die ausgewählten Modell-Ebenen die Koordinaten der umschließenden Rechtecke, sowie die geschätzten Klassen und deren Konfidenzwerte.

Modell mit *'type'='segmentation'*

- `Outputs`=*'segmentation_image'*: `DLResult` enthält ein Bild, in dem jedes Pixel einen Wert entsprechend der Klasse erhält, welcher das entsprechende Pixel zugeordnet wurde.
- `Outputs`=*'segmentation_confidence'*: `DLResult` enthält ein Bild, in dem jedes Pixel den Konfidenzwert der Klassifikation des entsprechenden Pixels erhält.
- `Outputs`=*'[]'*: `DLResult` enthält alle Ausgabewerte.

Achtung

Systemvoraussetzungen: Um diesen Operator auf der GPU auszuführen, indem *'device'* auf *'gpu'* gesetzt ist (siehe `get_dl_model_param`), werden cuDNN und cuBLAS benötigt. Für weitere Details wird auf den „`Installation Guide`", Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods", verwiesen.

Parameter

▷ **`DLModelHandle`** (input_control) ..... dl_model ⇝ *handle*
Handle des Deep Learning-Modells.

▷ **`DLSampleBatch`** (input_control) ..... dict-array ⇝ *handle*
Eingabedaten.

▷ **`Outputs`** (input_control) ..... string-array ⇝ *string*
Angeforderte Ausgabedaten.
Defaultwert: []
Werteliste: `Outputs` ∈ {[], 'segmentation_image', 'segmentation_confidence', 'bboxhead2_prediction', 'classhead2_prediction'}

▷ **`DLResultBatch`** (output_control) ..... dict-array ⇝ *handle*
Dictionary mit Ergebnisdaten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `apply_dl_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_dl_model`, `train_dl_model_batch`, `train_dl_model_anomaly_dataset`, `set_dl_model_param`

Modul

Deep Learning Inference

clear_dl_model (: : DLModelHandle :)

Löschen eines Deep Learning-Modells.

`clear_dl_model` löscht das Handle des Deep Learning-Modells, das durch `DLModelHandle` angegeben wird, und gibt den entsprechenden Speicherplatz frei. Das Modell kann nach dem Aufruf von `clear_dl_model` nicht mehr verwendet werden. Das Handle `DLModelHandle` ist nach dem Aufruf ungültig.

Für weitere Informationen zu Deep Learning-Modellen in HALCON siehe das Kapitel Deep Learning / Modell.

Parameter

▷ **DLModelHandle** (input_control) . . . . . dl_model(-array) ⇝ *handle*
Handle des Deep Learning-Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `clear_dl_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DLModelHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`read_dl_model`, `apply_dl_model`, `train_dl_model_batch`, `train_dl_model_anomaly_dataset`

Modul

Deep Learning Inference

create_dl_pruning (: : DLModelHandle, Mode, GenParam : DLPruningHandle)

Erstellen eines Handles für Pruning-Daten.

Der Operator create_dl_pruning erstellt ein Handle DLPruningHandle. Dieses Handle wird beim Pruning eines Deep Learning-Modells verwendet um Informationen zu übergeben.

Mode bestimmt die Pruning-Methode. Die Methode impliziert was im Modell reduziert wird und damit auch welche Scores benötigt werden. Momentan ist nur der folgende *'mode'* implementiert:

'oracle': Kernels werden aus Convolutional-Layern entfernt. Dafür wird für jeden (möglicherweise entfernbaren) Kernel ein Score berechnet, welcher angibt, wie wichtig der Kernel im verwendeten Netzwerk für die gegebenen Bilder ist. Für weitere Informationen wird auf den Artikel im Abschnitt „Literatur" verwiesen.

GenParam ist ein Dictionary zum Setzen generischer Parameter. Zur Zeit werden keine generischen Parameter unterstützt.

Für eine Erklärung zum Konzept von Deep Learning wird auf die Einleitung des Kapitels Deep Learning verwiesen.

Parameter

▷ **DLModelHandle** (input_control) . . . . . dl_model ⇝ *handle*
Handle des Deep Learning-Modells.

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Pruning-Methode.
Defaultwert: 'oracle'
Werteliste: Mode ∈ {'oracle'}

▷ **GenParam** (input_control) . . . . . dict ⇝ *handle*
Dictionary mit generischen Parameter.
Defaultwert: []

▷ **DLPruningHandle** (output_control) . . . . . dl_pruning ⇝ *handle*
Handle mit den Pruning-Daten.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_dl_model

Nachfolger

get_dl_pruning_param, set_dl_model_param

Literatur

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, Jan Kautz "Pruning Convolutional Neural Networks for Resource Efficient Inference", ICLR 2017, 5th International Conference on Learning Representations, Toulon, France.

Modul

Deep Learning Training

deserialize_dl_model (: : SerializedItemHandle : DLModelHandle)

Deserialisiert ein Deep Learning-Modell.

Der Operator deserialize_dl_model deserialisiert ein durch das Handle SerializedItemHandle definiertes Deep Learning-Modell, das mit dem Operator serialize_dl_model serialisiert wurde.

Der Operator verhält sich wie read_dl_model mit dem Unterschied, dass keine Datei, sondern ein serialisiertes Element als Eingabe dient. Für eine detailliertere Beschreibung sei auf die Dokumentation des Operators read_dl_model verwiesen.

Für weitere Informationen zu Deep Learning-Modellen in HALCON siehe das Kapitel Deep Learning / Modell.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **DLModelHandle** (output_control) ..... dl_model ⇝ *handle*
Handle des Deep Learning-Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_dl_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_dl_model`

Nachfolger

`apply_dl_model`, `train_dl_model_batch`, `train_dl_model_anomaly_dataset`, `set_dl_model_param`, `get_dl_model_param`

Siehe auch

`serialize_dl_model`

Modul

Deep Learning Inference

gen_dl_model_heatmap (: : DLModelHandle, DLSample, HeatmapMethod, TargetClasses, GenParam : DLResult)

Inferieren eines Samples und generieren einer Heatmap.

Der Operator `gen_dl_model_heatmap` inferiert das Sample `DLSample` und generiert eine Heatmap der in `TargetClasses` angegebenen Klasse. Dabei kann `DLSample` ein einzelnes Dictionary oder ein Tupel von Dictionaries verschiedener Samples sein. Dafür wird das in `DLModelHandle` gegebene Modell verwendet. Der Parameter `HeatmapMethod` bestimmt, mit welcher Methode die Heatmap berechnet wird. Die Resultate werden in `DLResult` zurückgegeben mit einem Ergebnis-Dictionary für jedes übergebene Sample. Es gilt zu bemerken, dass `DLResult` jeweils neu angelegt wird und somit bereits existierende Parameter dieses Namens überschrieben werden.

Eine Heatmap kann nützlich sein bei der Analyse, welche Teile eines Bildes bei seiner Inferenz in eine bestimmte Klasse einen starken Einfluss haben. Form und Größe dieser Bereiche und somit die Heatmaps können sich für verschiedene Netzwerke stark von einander unterscheiden, insbesondere wenn sich die Architekturen unterscheiden. Es gibt verschiedene Methoden um eine Heatmap zu berechnen. Die folgenden Methoden können über den Parameter `HeatmapMethod` ausgewählt werden:

'grad_cam': Steht für "Gradient-weighted Class Activation Mapping (Grad-CAM)". Für weitere Informationen zu dieser Methode wird auf die unten zitierte Referenz verwiesen.

Der Eingabeparameter `TargetClasses` bestimmt die Zielklasse, die Klasse, für welche die Heatmap generiert wird. Als Wert wird die Klassen-ID gesetzt. Alternativ kann auch ein leeres Tupel angegeben werden. In diesem Fall wird die Heatmap für die Klasse mit höchstem Konfidenzwert generiert, also die Klasse der besten Vorhersage.

Folgende Werte können über das Dictionary `GenParam` gesetzt werden:

target_conv_layer: Legt fest, von welchem Faltungs-Layer die Aktivierung und der Gradient zur Erstellung der Heatmap verwendet werden sollen. Der Faltungs-Layer kann über seinen Namen `NAME` bestimmt werden, wie er von *'summary'* in `get_dl_model_param` zurückgegeben wird, z.B. *'conv10'*. Als Standardwert

wird der letzte (tiefste) Faltungs-Layer mit einer nicht-trivialen Aktivierung (Weite und Höhe ungleich 1) verwendet. Es wird empfohlen den Standardwert zu verwenden, da sich die Aktivierung des tiefsten Faltungs-Layer in der Regel am besten zum Erzeugen der Heatmap eignet.

use_conv_only: Falls der ausgewählte Faltungs-Layer (mittels `target_conv_layer`) mit einem direkt folgenden ReLU-Layer fusioniert ist, kann mit diesem Parameter angegeben werden, von welchem dieser beiden Layer die Aktivierung und die Gradienten für die Heatmap genutzt werden sollen. Die folgenden Werte werden unterstützt:

'true': Faltungs-Layer

'false': ReLU-Layer

Der Standardwert für `use_conv_only` ist *'true'*.

scaling: Bestimmt die Art der Skalierung der Heatmap. Die folgenden Werte werden unterstützt:

'scale_after_relu': Alle negativen Werte der Heatmap werden auf 0 gesetzt. Anschließend werden alle Werte auf den Wertebereich [0,1] skaliert. Dadurch können auch Bereiche mit schwacher Aktivierung sehr hohe Werte nahe 1 erhalten.

'scale_before_relu': All Werte der Heatmap werden durch den maximalen Wert der Absolutbeträge dividiert und anschließend alle negativen Werte auf 0 gesetzt. Dadurch werden die Werte in den Bereich [0,1] skaliert. Im Vergleich zu *'scale_after_relu'* müssen schwach aktivierte Bereiche jedoch nicht hohe Werte nahe der 1 bekommen.

'none': Die Werte der Heatmap werden nicht skaliert.

Der Standardwert für `scaling` ist *'scale_after_relu'*.

Jedes Ergebnis-Dictionary in `DLResult` beinhaltet die Inferenz-Resultate wie sie mit `apply_dl_model` erhalten werden. Zusätzlich beinhaltet es ein verschachteltes Dictionary unter dem Schlüssel `heatmap_method`, wobei `method` der Name der Methode zur Berechnung der Heatmap ist, wie in `HeatmapMethod` gegeben. In diesem verschachtelten Dictionary ist die Heatmap unter dem Schlüssel `heatmap_image_class_classID` abgespeichert, wobei `classID` die ID der Zielklasse ist.

Achtung

Systemvoraussetzungen: Um diesen Operator auf der GPU auszuführen, indem *'runtime'* auf *'gpu'* gesetzt ist (siehe `get_dl_model_param`), werden cuDNN und cuBLAS benötigt. Für weitere Details wird auf den „`Installation Guide`", Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods", verwiesen. Alternativ kann dieser Operator auch auf der CPU ausgeführt werden, indem man *'runtime'* auf *'cpu'* setzt.

Die Heatmap dient als Hilfe um Klassifikationsergebnisse besser zu verstehen und ist nicht zur Segmentierung geeignet. Außerdem sei bemerkt, dass `gen_dl_model_heatmap` derzeit nur Modelle mit *'type'='classification'* unterstützt.

Parameter

▷ **DLModelHandle** (input_control) ..... dl_model ⇝ *handle*
Handle des Deep Learning-basierten Modells.

▷ **DLSample** (input_control) ..... dict-array ⇝ *handle*
Dictionaries mit den Eingabedaten der Samples.

▷ **HeatmapMethod** (input_control) ..... string ⇝ *string*
Methode nach der die Heatmap berechnet wird.
Defaultwert: 'grad_cam'

▷ **TargetClasses** (input_control) ..... tuple(-array) ⇝ *integer*
IDs der Zielklasse.
Defaultwert: []

▷ **GenParam** (input_control) ..... dict ⇝ *handle*
Dictionary für generische Parameter.
Defaultwert: []

▷ **DLResult** (output_control) ..... dict(-array) ⇝ *handle*
Dictionaries mit den Resultaten.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

apply_dl_model

Literatur

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra: "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization,"2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 618-626. doi: 10.1109/ICCV.2017.74

Modul

Deep Learning Inference

gen_dl_pruned_model (: : DLModelHandleToPrune,
DLPruningHandle : DLModelHandlePruned)

Pruning eines Deep Learning-Modells.

Der Operator gen_dl_pruned_model kopiert das Eingabe-Modell DLModelHandleToPrune und gibt die reduzierte Kopie in DLModelHandlePruned zurück. Es ist zu beachten, dass add_dl_pruning_batch nur Deep Learning-Modelle vom Typ *'type'='classification'* unterstützt.

Der Parameter DLPruningHandle liefert alle notwendigen Informationen, wie z.B. den ausgewählten Pruning Modus, den Prozentsatz oder die Kernel Scores. Siehe create_dl_pruning für weitere Informationen zu den implementierten Pruning Modi.

Verhalten in speziellen Fällen:

- Falls die angegebene Pruning-Menge *'percentage'* zu klein ist, um zu einer Reduzierung zu führen, ist DLModelHandlePruned eine Kopie des Eingabe-Modells DLModelHandleToPrune.
- Falls das Modell nicht erfolgreich reduziert werden konnte, wird ein leeres Handle zurückgegeben.

Parameter

▷ **DLModelHandleToPrune** (input_control) ..... dl_model ⇝ *handle*
Eingabe-Modell.

▷ **DLPruningHandle** (input_control) ..... dl_pruning ⇝ *handle*
Handle mit den Pruning-Daten.

▷ **DLModelHandlePruned** (output_control) ..... dl_model ⇝ *handle*
Reduziertes Modell.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_dl_pruning_batch, set_dl_pruning_param

Modul

Deep Learning Training

get_dl_model_param (: : DLModelHandle,
GenParamName : GenParamValue)

Auslesen der Parameter des Deep Learning-Modells.

get_dl_model_param gibt die Parameterwerte von GenParamName für das Deep Learning-Modell DLModelHandle in GenParamValue aus.

Für ein Deep Learning-Modell können die Parameter `GenParamName`, je nach Parameter und Modelltyp, mit Hilfe von `set_dl_model_param` oder `create_dl_model_detection` gesetzt werden.

Mit diesem Operator, `get_dl_model_param`, können die Parameterwerte `GenParamValue` abgefragt werden. Nachstehend ist ein Überblick über die verschiedenen Parameter gegeben sowie eine Erklärung zu den Parametern, mit Ausnahme jener, die nur gesetzt werden können. Für eine Erklärung letztgenannter Parameter wird auf die Dokumentation der entsprechenden Operatoren verwiesen.

GenParamName	**Ano. Det.**	**Class.**	**Obj. Det.**	**Sem. Seg.**
'batch_size'	s g	s g	s g	s g
'batch_size_multiplier'	x	s g	s g	s g
'class_ids'	–	g	s g c	s g
'class_weights'	–	s g	g c	–
'device'	s g	s g	s g	s g
'enable_resizing'	s	s	s	s
'fuse_bn_relu'	s	s	s	s
'fuse_conv_relu'	s	s	s	s
'gpu'	s g	s g	s g	s g
'image_dimensions'	s g	s g	g c	s g
'image_height'	s g	s g	g c	s g
'image_width'	s g	s g	g c	s g
'image_num_channels'	s g	s g	g c	s g
'image_range_max'	g	g	g	s g
'image_range_min'	g	g	g	s g
'input_dimensions'	s g	s g	g	s g
'learning_rate'	x	s g	s g	s g
'meta_data'	s g	s g	s g	s g
'momentum'	x	s g	s g	s g
'num_classes' (`NumClasses`)	–	–	g c	g
'num_trainable_params'	g	g	g	g
'optimize_for_inference'	–	s g	s g c	s g
'precision'	g	g	g	g
'precision_is_converted'	g	g	g	g
'runtime'	s g	s g	s g	s g
'runtime_init'	s	s	s	s
'summary'	x	g	g	g
'type'	g	g	g	g
'weight_prior'	x	s g	s g	s g

GenParamName	Ano. Det.	Class.	Obj. Det.	Sem. Seg.
'complexity'	s g	–	–	–
'standard_deviation_factor'	s g	–	–	–
'class_names'	–	s g	–	–
'extract_feature_maps'	–	s g	–	–
'anchor_angles'	–	–	g c	–
'anchor_aspect_ratios'	–	–	g c	–
'anchor_num_subscales'	–	–	g c	–
'backbone' (Backbone)	–	–	g c	–
'backbone_docking_layers'	–	–	s g c	–
'bbox_heads_weight'	–	–	s g c	–
'capacity'	–	–	g c	–
'class_heads_weight'	–	–	s g c	–
'class_ids_no_orientation'	–	–	g c	–
'freeze_backbone_level'	–	–	s g c	–
'ignore_direction'	–	–	g c	–
'instance_type'	–	–	g c	–
'max_level'	–	–	g c	–
'min_level'	–	–	g c	–
'max_num_detections'	–	–	s g c	–
'max_overlap'	–	–	s g c	–
'max_overlap_class_agnostic'	–	–	s g c	–
'min_confidence'	–	–	s g c	–
'ignore_class_ids'	–	–	–	s g

Hierbei stehen die verwendeten Symbole für folgendes:

- 's': Der Parameter kann mit set_dl_model_param gesetzt werden.
- 'g': Der Parameter kann mit get_dl_model_param abgefragt werden.
- 'c': Der Parameter kann mit create_dl_model_detection gesetzt werden.
- 'x': Der Parameter hat für diesen Modelltypen keine Anwendung.
- '-': Der Parameter hat für diesen Modelltypen keine Anwendung und ein Fehler wird geworfen.

Gewisse Parameter werden als nicht-optionale Parameter gesetzt, die entsprechende Repräsentation ist in Klammern angegeben.

Im Folgenden werden die Parameter GenParamName, deren Werte mit get_dl_model_param abgefragt werden können, aufgelistet und erläutert. Die Auflistung ist dem Modelltyp entsprechend sortiert. Für Modelle mit *'type'='segmentation'* hängen die Standardwerte vom spezifischen Netzwerk ab und müssen somit abgefragt werden.

Auf mehrere Modelltypen anwendbar

'batch_size'

Anzahl der Bilder (und ihren entsprechenden Labels) in einem Batch, die gleichzeitig in den Gerätespeicher transferiert werden.
Bei einem Training mittels train_dl_model_batch ist die Menge der Bilder, die in einer Iteration miteinander verarbeitet werden, beinhaltet eine Anzahl *'batch_size'* mal *'batch_size_multiplier'* Bilder. Für weitere Details siehe train_dl_model_batch.
Bei der Inferenz kann die *'batch_size'* im Allgemeinen unabhängig von der Anzahl der gleichzeitig zu inferierenden Bilder gesetzt werden. Siehe apply_dl_model für weitere Informationen, wie man diesen Parameter setzt, um eine größere Effizienz zu erzielen.

Modelle mit *'type'='classification'*:

Der Parameter *'batch_size'* ist im vortrainierten Modell gespeichert. Standardmäßig ist *'batch_size'* so gesetzt, dass ein Training des vortrainierten Klassifikators mit bis zu `100` Klassen problemlos auf einer GPU mit einem Speicher von `8` Gigabyte durchgeführt werden kann.
Für die vortrainierten Klassifikatoren ergeben sich so folgende Standardwerte für die Batchgröße:

Vortrainierter Klassifikator	**Standardwerte für** *'batch_size'*
'pretrained_dl_classifier_alexnet.hdl'	*230*
'pretrained_dl_classifier_compact.hdl'	*160*
'pretrained_dl_classifier_enhanced.hdl'	*96*
'pretrained_dl_classifier_mobilenet_v2.hdl'	*40*
'pretrained_dl_classifier_resnet50.hdl'	*23*

'batch_size_multiplier'

Multiplikationsfaktor für *'batch_size'*, um das Training mit einer größeren Anzahl an Bildern in einem Schritt zu ermöglichen, was aufgrund limitierter GPU-Speichergröße sonst nicht möglich wäre. Dieser Parameter beeinflusst nur das Training mit `train_dl_model_batch` und hat somit weder auf die Evaluierung noch auf die Inferenz einen Einfluss. Für weitere Details, siehe `train_dl_model_batch`.

Modelle mit *'type'='anomaly_detection'*: Der Parameter *'batch_size_multiplier'* hat keinen Einfluss.

Default: 'batch_size_multiplier' = 1

'class_ids':

Eindeutige IDs der Klassen, die vom Modell unterschieden werden sollen. Daher hat das Tupel die Länge von *'num_classes'*.
Hierbei sollen die leicht unterschiedlichen Bedeutungen und Beschränkungen in Abhängigkeit des Modelltyps betont werden:

Modelle mit *'type'='anomaly_detection'*:

Der Parameter *'class_ids'* wird nicht unterstützt.

Modelle mit *'type'='classification'*:

Die IDs sind eindeutige Bezeichner, die automatisch den Klassen zugeordnet werden. Dabei entsprechen die ID einer Klasse dem Index der Klasse im Tupel *'class_names'*.

Modelle mit *'type'='detection'*:

Beinhaltet nur die Klassen der zu erkennenden Objekte, und somit keine Hintergrund-Klasse. Dabei kann als Klassen-ID ein beliebiger Integer-Wert aus dem Wertebereich $[0, 65534]$ gesetzt werden.
Es gilt zu beachten, dass die Werte in *'class_ids_no_orientation'* von jenen in *'class_ids'* abhängen. Falls die Werte in *'class_ids'* nach der Erzeugung des Modells verändert werden, wird *'class_ids_no_orientation'* auf ein leeres Tupel zurückgesetzt.
Default: 'class_ids' = '[0,...,num_classes-1]'

Modelle mit *'type'='segmentation'*:

Jede zum Training genutzte Klasse muss enthalten sein und damit ebenfalls die Klassen-ID der Klasse 'background'. Daher hat für ein solches Modell das Tupel die die minimale Länge 2. Dabei kann als Klassen-ID ein beliebiger Integer-Wert aus dem Wertebereich $[0, 65534]$ gesetzt werden.

'class_weights':

Der Parameter *'class_weights'* ist ein Tupel klassenspezifischer Gewichtungsfaktoren für die Zielfunktion. Erhalten die verschiedenen Klassen unterschiedliche Gewichte, so wird das Netzwerk forciert, den Klassen beim Lernen eine unterschiedliche Bedeutung zu geben. Dies ist nützlich, wenn eine Klasse den Datensatz dominiert. Die einzelnen Gewichtungsfaktoren müssen innerhalb des Intervalls $(0, 1)$ sein. Dabei gilt, je größer das Gewicht einer Klasse, desto stärker ist ihr Einfluss beim Training. Im Tupel *'class_weights'* sind die Faktoren gleich angeordnet wie die Klassen im Tupel *'class_ids'*. Je nach Modelltyp sind folgende Unterschiede zu beachten:

Modelle mit *'type'='anomaly_detection'*:

Der Parameter *'class_weights'* wird nicht unterstützt.

Modelle mit *'type'='classification'*:

Default: 'class_weights' = 1.0 für jede Klasse.

Modelle mit *'type'='detection'*:

Default: 'class_weights' = 0.25 für jede Klasse.

Modelle mit *'type'='segmentation'*:

Der Parameter *'class_weights'* wird nicht unterstützt.

'device':
Handle der Hardware-Einheit, auf der die Deep Learning Operatoren ausgeführt werden. Ein Tupel aller Handles aller potentiell Deep Learning fähigen Geräte kann über `query_available_dl_devices` erzeugt werden.
Default: Handle der standardmäßig gesetzten GPU mit Index *0* in der mit *'gpu'* erhaltenen Liste. Falls keine GPU vorhanden ist, ein leeres Tupel.

'gpu':
Identifikationsnummer der GPU, auf der die Trainings- und Inferenz-Operatoren (`train_dl_model_batch` und `apply_dl_model`) ausgeführt werden. Standardmäßig wird die erste verfügbare GPU verwendet.
Mit dem `get_system` Parameter *'cuda_devices'* kann die Liste der verfügbaren GPUs abgefragt werden. In *'gpu'* muss der Index der gewünschten GPU in dieser Liste übergeben werden.
Es ist zu beachten, dass der Wert für *'gpu'* nur berücksichtig wird, falls *'runtime'='gpu'* gesetzt ist. Deshalb sollte das Setzen der spezifischen GPU vorzugsweise über den Parameter *'device'* erfolgen.
Default: 'gpu' = 0

'image_dimensions':
Tupel, das die Dimensionen der Eingabebilder *'image_width'*, *'image_height'* und Anzahl der Bildkanäle *'image_num_channels'* beinhaltet.
Die jeweiligen Standardwerte und möglichen Wertebereiche sind vom Modelltyp und Modell selbst abhängig. Weitere Angaben dazu finden sich in der Beschreibung des jeweiligen Dimensionsparameters.

'image_height', *'image_width'*:
Höhe und Breite die die Eingabebilder aufweisen und mit denen das Netzwerk arbeitet.
Dieser Parameter kann abhängig vom Modelltyp verschiedene Werte annehmen:

Modelle mit *'type'='anomaly_detection'*:
Die Standardwerte hängen von dem zu verwendenden Netzwerk ab, siehe `read_dl_model`. Die Netzwerkarchitekturen erlauben verschiedene Bildgrößen, welche über `set_dl_model_param` gesetzt werden können. Siehe auch `read_dl_model` um zu erfahren, welche Einschränkungen es für die Eingabebildgröße bei den jeweiligen ausgelieferten Netzwerken gibt. Das Setzen der Bildgröße muss vor dem Training des Modells erfolgen, da ansonsten das Modell in einen unbrauchbaren Zustand versetzt würde. Falls versucht wird, die oben genannten Parameter in einem bereits trainierten Modell zu setzen, wird ein Fehler ausgegeben und das Modell bleibt unverändert.

Modelle mit *'type'='classification'*:
Die Standardwerte hängen vom vortrainierten Klassifikator ab, siehe `read_dl_model`. Die Netzwerkarchitekturen erlauben verschiedene Bildgrößen, welche über `set_dl_model_param` gesetzt werden können. Für Netzwerke mit Fully-connected Layern wird dadurch ein Nachtrainieren notwendig. Netzwerke ohne Fully-connected Layer sind direkt auf verschiedenen Bildgrößen anwendbar. Jedoch wird für Bilder, deren Größe von der Bildgröße der Trainingsbilder abweicht, eine geringere Genauigkeit bei der Klassifikation erwartet.

Modelle mit *'type'='detection'*:
Die Netzwerkarchitekturen erlauben Änderungen der Bilddimensionen. Jedoch werden für jede Ebene die Bildlängen halbiert, weshalb *'image_width'* und *'image_height'* ganzzahlige Vielfache von 2^{level} sein müssen. 2 Dabei ist die Anzahl Ebenen $level$ vom Backbone und dem Parameter *'max_level'* abhängig, siehe `create_dl_model_detection`.
Default: 'image_height' = 640, *'image_width' = 640*

Modelle mit *'type'='segmentation'*:
Die Netzwerkarchitekturen erlauben Änderungen der Bilddimensionen. Der Standardwert und Minimalwerte werden dabei vom Netzwerk vorgegeben, siehe `read_dl_model`.

'image_num_channels':
Anzahl Kanäle der Eingabebilder, die vom Netzwerk verarbeitet werden. Der Standardwert wird dabei vom Netzwerk vorgegeben, siehe `read_dl_model` und `create_dl_model_detection`.
Für Modelle mit *'type'='anomaly_detection'* werden nur die Werte *1* und *3* unterstützt. Außerdem kann für Modelle mit *'type'='anomaly_detection'* dieser Parameter nur vor dem Training des Modells gesetzt werden. Ansonten würde das Modell in einen unbrauchbaren Zustand versetzt. Falls versucht wird, *'image_num_channels'* in einem bereits trainierten Modell zu setzen, wird ein Fehler ausgegeben und das Modell bleibt unverändert. Für andere Modelle ist eine beliebige Anzahl an Kanälen möglich.
Wird die Anzahl auf einen Wert >1 geändert, werden die Werte aller Layer nach dem Eingabe-Layer mit Zufallswerten initialisiert. In diesem Fall sind mehr Daten zum Nachtrainieren notwendig. Wird die Anzahl Kanäle der Eingabebilder auf 1 geändert, so werden die Gewichte der entsprechenden Layer

vereinigt.

Modelle mit *'type'='anomaly_detection'*:
Default: 'image_num_channels' = 3

Modelle mit *'type'='detection'*:
Default: 'image_num_channels' = 3

'image_range_max', *'image_range_min'*:
Maximaler und minimaler möglicher Grauwert der Eingabebilder, welche das Netzwerk verarbeitet. Die Werte werden dabei vom Netzwerk vorgegeben, siehe `read_dl_model` und `create_dl_model_detection`.

'input_dimensions':
Dictionary, welches die Dimensionen aller Netzwerkeingaben beinhaltet. Beispiele für solche Eingaben: Eingabebild, `weight_image` (bei Modellen mit *'type'='segmentation'*).
Diese Dimensionen sind im Dictionary als Tupel [`Breite`, `Höhe`, `Tiefe`] gegeben. Verwendet man diesen Parameter zum Setzen der Werte, kann für jede Dimension der Wert -1 gesetzt werden um den aktuellen Wert beizubehalten.

'learning_rate':
Wert des Faktors, der den Einfluss des Gradienten während des Trainings mit `train_dl_model_batch` bestimmt. Mehr Informationen können in der Dokumentation von `train_dl_model_batch` gefunden werden.
Der Standardwert ist vom Modell abhängig.

Modelle mit *'type'='anomaly_detection'*:
Der Parameter *'learning_rate'* hat keinen Einfluss.

'meta_data':
Dictionary mit benutzerdefinierten Meta-Daten, dessen Einträge frei gesetzt werden können. Die Meta-Daten können beispielsweise verwendet werden um zum Modell gehörige Information wie Versionsnummern oder Autor mit dem Modell abzuspeichern.
Einschränkung: Als Dictionary-Werte sind nur Strings oder Tupel von Strings erlaubt.

'momentum':
Werden die Gewichte des Netzwerkes aktualisiert, gibt der Hyperparameter *'momentum'* an, zu welchem Ausmaß vorherige Aktualisierungsvektoren im aktuellen Aktualisierungsvektor berücksichtigt werden. Für weitere Informationen verweisen wir auf die Dokumentation von `train_dl_model_batch`.
Der Standardwert wird vom Modell gegeben.

Modelle mit *'type'='anomaly_detection'*: Der Parameter *'momentum'* hat keinen Einfluss.

'num_classes':
Anzahl unterschiedlicher Klassen, die das Modell für seine Vorhersagen unterscheiden kann.
Dieser Parameter unterscheidet sich für die Modelltypen:

Modelle mit *'type'='anomaly_detection'*:
Der Parameter *'num_classes'* wird nicht unterstützt.

Modelle mit *'type'='classification'*:
'num_classes' wird implizit durch die Länge von *'class_names'* bestimmt und wird dadurch nicht unterstützt.

Modelle mit *'type'='detection'*:
Dieser Wert wird über den Parameter `NumClasses` mit dem Operator `create_dl_model_detection` gesetzt. Dabei muss *'class_ids'* stets eine Anzahl Elemente gleich *'num_classes'* haben.

Modelle mit *'type'='segmentation'*:
Da ein Modell des Typs *'type'='segmentation'* die Klasse 'background' vorhersagt, ist diese Klasse in *'num_classes'* enthalten. Für diese Modelle wird *'num_classes'* automatisch aus der Länge von *'class_ids'* bestimmt.

'num_trainable_params':
Anzahl trainierbarer Parameter (Gewichte und Bias-Terme) des Modells. Dieser Wert ist ein Indikator für die Größe des Modells, wenn es serialisiert wird.

'optimize_for_inference':
Definiert, ob das Modell optimiert und nur für die Inferenz anwendbar ist.
Das Modell bleibt auch nach der Optimierung auf HALCON Standardeinheiten ausführbar (im Gegensatz zu `optimize_dl_model_for_inference`).

Durch das Setzen des Parameterwertes auf *'true'* wird der Speicher freigegeben, den das Modell für die Inferenz nicht benötigt (z.B. Speicher für die Gradienten). Dies kann den Speicherbedarf des Modells signifikant reduzieren. Infolgedessen sind bei Modellen mit dieser Eigenschaft keine Gradienten verfügbar (z.B. notwendig für das Training oder die Berechnung der Heatmap mit Modus *'grad_cam'*). Operatoren, welche Werte des freigegebenen Speichers verwenden (z.B. `train_dl_model_batch`), setzen diesen Parameter automatisch zurück auf *'false'*.
Wird der Wert auf *'false'* zurückgesetzt (egal ob manuell oder automatisch), wird der zum Trainieren benötigte Speicher erneut allokiert. Dies bedeutet, ein nachfolgendes Training verhält sich als ob *'momentum'* kurzzeitig auf *0* gesetzt wäre (da mögliche Aktualisierungsvektoren erneut akkumuliert werden müssen).

Modelle mit *'type'='anomaly_detection'***:**
Der Parameter *'optimize_for_inference'* wird nicht unterstützt.

Default: 'false'

*'precision'***:**
Definiert den Datentyp, der intern für die Berechnung eines Forward-Pass eines Deep Learning-Modells verwendet wird.
Default: 'float32'

*'precision_is_converted'***:**
Gibt an, ob das Modell nach dem Training einem Konvertierungsprozess über `optimize_dl_model_for_inference` unterzogen wurde.
Default: 'false'

*'runtime'***:**
Standardmäßig von HALCON oder über eine AI²-Schnittstelle unterstützte Hardware-Einheit, auf der die Deep Learning Operatoren ausgeführt werden.
Beachte, der Parameter *'device'* sollte bevorzugt zum Bestimmen der Hardware-Einheit, auf der die Deep Learning Operatoren ausgeführt werden, verwendet werden. *Default: 'runtime' = 'gpu'*

*'cpu'***:**
Die Trainings- und Inferenz-Operatoren werden auf der CPU ausgeführt. Es ist zu beachten, dass das Training nur für spezifische Plattformen unterstützt wird, siehe HALCON „`Installation Guide`", Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods".
Falls zuvor auf der GPU gerechnet wurde, wird Speicher auf der CPU initialisiert und falls notwendig werden auf der GPU gespeicherte Werte auf die CPU verschoben.
Zur Parallelisierung: Die Laufzeit ist stark abhängig von der Anzahl der gesetzten Threads, wobei die Nutzung aller verfügbaren Threads nicht unbedingt zu einer schnelleren Performance führt. Die aktuell gesetzte Anzahl an Threads kann mit dem Operator `get_system` abgefragt werden.
Die implementierte CPU Parallelisierung ist architekturabhängig:

- Intel oder AMD Architekturen: OpenMP. Dabei werden standardmäßig alle verfügbaren Threads der OpenMP Laufzeitumgebung verwendet. Die Anzahl der verwendeten Threads kann mit dem thread-spezifischen Parameter *'tsp_thread_num'* des Operators `set_system` spezifiziert werden.
- Arm Architekturen: Globaler Thread Pool. Die Anzahl der Threads kann mit dem globalen Parameter *'thread_num'* des Operators `set_system` gesetzt werden. Jedoch ist es nicht möglich eine thread-spezifische Anzahl von Threads (über den Parameter *'tsp_thread_num'* des Operators `set_system`) einzustellen.

*'gpu'***:**
Der Speicher wird auf der GPU initialisiert. Die Operatoren `apply_dl_model`, `train_dl_model_batch` und `train_dl_model_anomaly_dataset` werden auf der GPU ausgeführt. Mehr Informationen zu diesen speziellen Voraussetzungen befinden sich im HALCON „`Installation Guide`".

*'tensorrt'***:**
Der Speicher wird über das Nvidia TensorRT-Plugin für die AI²-Schnittstelle initialisiert. Der Operator `apply_dl_model` wird auf der GPU unter Verwendung der TensorRT-Schnittstelle ausgeführt. Um den Operator auf der ausgewählten Einheit erfolgreich auszuführen, kann eine Konvertierung mittels `optimize_dl_model_for_inference` notwendig sein, siehe dessen Dokumentationseintrag.

*'summary'***:**
Mit diesem Parameter werden Informationen über die Layer des Modells zurückgegeben. Genauer gesagt

wird ein Tupel zurückgegeben, welches für jeden Layer einen String der folgenden Form enthält `ID; NAME; TYPE; OUTPUT_SHAPE; CONNECTED_NODES`

- `ID`: Index des Layers im CNN Graphen.
- `NAME`: Von Menschen lesbarer Bezeichner (optional).
- `TYPE`: Bezeichner für den Typ des Layers (z. B. `input` oder `convolution`).
- `OUTPUT_SHAPE`: Größe der Ausgabe in der Form (`Breite`, `Höhe`, `Tiefe`, *'batch_size'*). Dies bedeutet, der Layer hat Merkmalskarten der Größe `Breite` mal `Höhe` und davon eine Anzahl entsprechend der `Tiefe`. Zusammen bilden diese ein Ikonisches Objekt mit einem Kanal pro Merkmalskarte. Die verschiedenen Objekte sind die Layer-Ausgaben verschiedener Samples. Der Parameter *'batch_size'* bestimmt, wie viele Objekte zusammen zurückgegeben werden.
- `CONNECTED_NODES`: Komma separierte Liste mit den Indices der Layer, die die Ausgabe des aktuellen Layers als Eingabe verwenden

Z.B.: `'3; conv1; convolution; (112, 112, 64, 160); 4'`.
Es ist zu beachten, dass für manche mit HALCON ausgelieferten Modelle die Architektur vertraulich ist. In diesem Fall gibt `get_dl_model_param` ein leeres Tupel für *'summary'* zurück.

'type':
Dieser Parameter gibt den HALCON-spezifischen Modelltypen zurück. Folgende Typen werden unterschieden:

- *'anomaly_detection'*
- *'classification'*
- *'detection'*
- *'segmentation'*
- *'generic'* - für spezifische eingelesene oder mitels Framework erstellte Modelle, siehe `set_dl_model_param`.

'weight_prior':
Regularisierungsparameter $\alpha \geq 0.0$ für die Regularisierung der Zielfunktion. Für eine genaue Beschreibung des Regularisierungsterms wird auf `train_dl_model_batch` verwiesen. Einfach gesagt begünstigt die Regularisierung einfachere Modelle, die weniger anfällig sind, Rauschen in den Daten zu lernen. Dadurch generalisieren diese Modelle in der Regel besser. Für den Fall einer Überanpassung des Klassifikators, empfehlen wir verschiedene Werte für den Parameter *'weight_prior'* zu testen. Die Wahl des Wertes ist ein Abwägen zwischen der Fähigkeit des Netzwerkes zu Generalisieren, einer Überanpassung oder einer Unteranpassung. Ist α zu klein, ist eine Überanpassung möglich, ist es zu groß kann es passieren, dass das Modell seine Fähigkeit verliert, sich an die Daten anzupassen, da dann alle Gewichte effektiv Null sind. Um einen idealen Wert für α zu finden, empfehlen wir eine Kreuzvalidierung durchzuführen. Das heißt, das Training wird für eine Reihe von Werten durchgeführt und man wählt den Wert, der den besten Validierungsfehler liefert. Üblich sind zum Testen die Werte für *'weight_prior'* auf einer logarithmischen Skala zwischen $1.0, ..., 0.0001$. Falls das Training sehr lange dauert, besteht die Möglichkeit für die Optimierung der Hyperparameter einen reduzierten Datensatz zu verwenden.

Modelle mit *'type'*=*'anomaly_detection'*: Der Parameter *'weight_prior'* hat keinen Einfluss.

Default: 'weight_prior' = 0.0 (mit der Ausnahme der vortrainierten Klassifikatoren `pretrained_dl_classifier_resnet50`: *'weight_prior' = 0.0001*, `pretrained_dl_classifier_alexnet`: *'weight_prior' = 0.0005*)

Modelle mit *'type'*=*'anomaly_detection'*

'complexity':
Dieser Parameter steuert die Kapazität des Modells, komplexere Anwendungen zu lösen. Ein höherer Wert ermöglicht es dem Modell, Bilder mit größerer Komplexität darstellen zu können. Allerdings steigen dann auch die Laufzeiten von Training und Inferenz. Dieser Parameter kann nur vor dem Training des Modells gesetzt werden, da ansonsten das Modell in einen unbrauchbaren Zustand versetzt würde. Falls versucht wird, 'complexity' in einem bereits trainierten Modell zu setzen, wird ein Fehler ausgegeben und das Modell bleibt unverändert.
Default: 'complexity' = 15

'standard_deviation_factor':
Der Anomalie-Score auf Bildebene wird intern als Mittelwert bestimmter interner Scores s plus ein Vielfaches von deren Standardabweichung berechnet.

$$S_{Image} = \mu(s) + \lambda\sigma(s)$$

Dabei bezeichnet *s* einen Pixelwert des internen `anomaly_image`, $\mu(s)$ der Mittelwert von *s* und $\sigma(s)$ die Standardabweichung von *s*. Der Parameter *'standard_deviation_factor'* setzt den Wert für λ und steuert dadurch, wie stark die Standardabweichung im Vergleich zum Mittelwert gewichtet wird.
Default: 'standard_deviation_factor' = 3.0

Modelle mit *'type'*=*'classification'*

'backbone_docking_layers':
Der Parameter *'backbone_docking_layers'* spezifiziert, welche Layer des Backbones von der Merkmalspyramide als Docking Layer verwendet werden sollen. Dabei werden die Layer über ihre Namen referenziert.
Die Spezifikation der Docking Layer kann bei jedem Klassifikator erfolgen, auch ohne dass diese als Backbone eingelesen werden. Berücksichtigt wird diese Angabe jedoch nur für Backbones der Objektdetektion. Beim Auswählen der Docking Layer muss beachtet werden, dass die Längen der Merkmalskarten von einem Docking Layer zum nächsten jeweils halbiert werden. Daumenregel: Jeweils der tiefste Layer für jede (seitliche) Auflösung im Backbone (entspricht einer Ebene, welche für die spezifische Objektdetektions-Aufgabe notwendigen ist).
Informationen zu den Namen und Größen der Layer eines Modells können mittels *'summary'* abgefragt werden.
Default: Für die vortrainierten Backbones die von HALCON ausgeliefert werden, hängen die Standardwerte vom Klassifikator ab. Andere Klassifikatoren haben keine standardmäßig gesetzten Docking Layer, weshalb dieser Parameter gesetzt werden muss, bevor sie als Backbone verwendet werden können.

'class_names':
Eindeutige Namen der vom Modell zu unterscheidenden Klassen. Die Anordnung der Klassennamen bleibt nach dem Setzen unverändert. Das Tupel hat die Länge *'num_classes'*.

'extract_feature_maps':
Mit diesem Parameterwert kann man die Merkmalskarten des angegebenen Modell-Layers für ein inferiertes Bild extrahieren.
Der ausgewählte Layer muss Bestandteil des vorhandenen Netzwerks sein. Eine Übersicht aller vorhandenen Layer des Modells kann durch den Operator `get_dl_model_param` mit dem zugehörigen Parameter *'summary'* zurückgegeben werden.
Beachte, die Netzwerkarchitektur wird durch diese Option verändert: Das Netzwerk wird nach dem ausgewählten Layer abgeschnitten. Dies kann nicht rückgängig gemacht werden. Soll die originale Netzwerkarchitektur wieder verwendet werden, muss diese mit `read_dl_model` erneut eingelesen werden.

Modelle mit *'type'*=*'detection'*

'anchor_angles':
Der Parameter *'anchor_angles'* bestimmt den Orientierungswinkel der Anchors eines Modells mit *'instance_type'* = *'rectangle2'*. Dabei ist die Orientierung in Bogenmaß angegeben und gibt den Winkel zwischen der horizontalen Achse und *'Length1'* an (mathematisch positiv). Für weitere Informationen zu Anchors siehe das Kapitel Deep Learning / Objektdetektion.
Es kann ein Tupel mit Werten gesetzt werden. Eine höhere Anzahl an Winkel erhöht die Anzahl vorgeschlagener umschließender Rechtecke, was zu einer besseren Lokalisierung führen kann, allerdings auch mehr Laufzeit und Speicherplatz in Anspruch nimmt.
Mögliche Werte: 'anchor_angles' $\in (-\pi, \pi]$ für *'ignore_direction' = 'false'*, *'anchor_angles'* $\in (-\pi/2, \pi/2]$ für *'ignore_direction' = 'true'*
Default: 'anchor_angles' = [0.0]

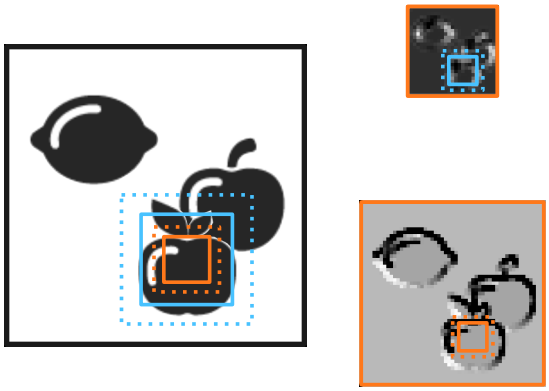
'anchor_aspect_ratios' **(veraltet:** *'aspect_ratios'***)**:
Der Parameter *'anchor_aspect_ratios'* bestimmt das Seitenverhältnis des umschließenden Referenzrechtecks. Dabei bestimmt der Instanztyp *'instance_type'*, wie dieses Verhältnis definiert ist:

- *'rectangle1'*: Verhältnis von Höhe zu Breite
- *'rectangle2'*: Verhältnis von *'Length1'* zu *'Length2'*

So ergibt z.B. für Instanztyp *'rectangle1'* ein Wert von 2 ein schmales initiales Referenzrechteck, während 0.5 ein breites Referenzrechteck ergibt.

Die Größe des generierten umschließenden Rechtecks wird durch den Parameter *'anchor_num_subscales'* beeinflusst und mit seiner Erklärung findet sich auch die genaue Formel für die Größen und die Längen der resultierenden umschließenden Anchors. Für weitere Informationen zu umschließenden Rechtecken siehe das Kapitel Deep Learning / Objektdetektion.
Es kann ein Tupel mit Werten gesetzt werden. Eine höhere Anzahl an Seitenverhältnissen erhöht die Anzahl vorgeschlagener umschließender Rechtecke, was zu einer besseren Lokalisierung führen kann, allerdings auch mehr Laufzeit und Speicherplatz in Anspruch nimmt.
Aus Gründen der Rückwärtskompatibilität kann anstelle des Parameters *'anchor_aspect_ratios'* auch der Parameter *'aspect_ratios'* verwendet werden.
Default: 'anchor_aspect_ratios' = [1.0, 2.0, 0.5]

'anchor_num_subscales' **(veraltet:** *'num_subscales'***):**
Dieser Parameter bestimmt die Anzahl an Anchors, die für jedes angegebenen Größenverhältnis in verschiedenen Skalierungen generiert werden.
In HALCON werden für jedes Pixel jeder Merkmalskarte der Merkmalspyramide umschließende Rechtecke vorgeschlagen. Der Parameter *'anchor_aspect_ratios'* beeinflusst die Form, der Parameter *'anchor_num_subscales'* die Größe. So werden auf jedem oben erwähnten Pixel *'anchor_aspect_ratios'* mal *'anchor_num_subscales'* umschließende Rechtecke generiert. Für weitere Informationen zu umschließenden Rechtecken siehe das Kapitel Deep Learning / Objektdetektion. Ein Beispiel findet sich in unten stehender Darstellung.



Mit *'anchor_num_subscales'*=2 werden für jedes Seitenverhältnis 2 verschieden große umschließende Referenzrechtecke auf jeder Ebene generiert: Ein Referenzrechteck mit Basislänge (durchgezogene Linie) und zusätzlich ein Größeres (gepunktete Linie). Dabei konvergieren diese zusätzlichen umschließenden Referenzrechtecke der unteren Ebene (orange) zu den umschließenden Referenzrechtecken mit Basislänge der nächsthöheren Ebene (blau).

Ein umschließendes Referenzrechteck der Ebene l hat auf dem Eingabebild standardmäßige Kantenlängen von $b \cdot 2^l$, wobei der Parameter den Wert $b = 4$ Pixel hat. Mit dem Parameter *'anchor_num_subscales'* können zusätzliche umschließende Rechtecke generiert werden, die größenmäßig zur Größe des kleinsten umschließenden Referenzrechtecks der Ebene $l + 1$ konvergiert. Genauer haben diese umschließenden Referenzrechtecke der Ebene l auf dem Eingabebild die Kantenlängen

$$scale_{li} = b \cdot 2^{l+i/'anchor_num_subscales'}$$

wobei $i \in [0,'anchor_num_subscales' - 1]$. Dies ergibt auf der Ebene l für subscale i ein umschließendes Referenzrechteck mit Höhe h und Breite w von

$$h_{lij} = scale_{li} \cdot \sqrt{r_j}, \qquad w_{lij} = scale_{li} \cdot \frac{1}{\sqrt{r_j}}$$

wobei r_j das Seitenverhältnis dieses umschließenden Referenzrechtecks ist (siehe *'anchor_aspect_ratios'*.
Eine große Anzahl von Subskalierungen erhöht die Anzahl vorgeschlagener umschließender Rechtecke und nimmt daher auch mehr Laufzeit und Speicherplatz in Anspruch.
Aus Gründen der Rückwärtskompatibilität kann anstelle des Parameters *'anchor_num_subscales'* auch der Parameter *'num_subscales'* verwendet werden.
Default: 'anchor_num_subscales' = 3

*'backbone'***:**

Der Parameter *'backbone'* ist der Name (inklusive Pfad) des Backbone-Netzwerks, das zum Erstellen des Modells verwendet wird. Eine Liste der verschiedenen mitgelieferten Backbone-Netzwerke ist unter `create_dl_model_detection` zu finden.

'bbox_heads_weight', *'class_heads_weight'*

Die Parameter *'bbox_heads_weight'* und *'class_heads_weight'* sind Gewichtungsfaktoren bei der Bestimmung der Gesamt-Zielfunktion. Dies bedeutet, wenn die einzelnen Zielfunktionswerte aufsummiert werden, werden die Beiträge der Heads zur Bestimmung der umschließenden Rechtecke mit dem Faktor *'bbox_heads_weight'* gewichtet und die Beiträge der Heads zur Klassifikation mit dem Faktor *'class_heads_weight'*.

Default: 'bbox_heads_weight' = 1.0, 'class_heads_weight' = 1.0

'capacity':

Bestimmt die Anzahl der Parameter (oder Filtergewichte) in den tieferen Teilen des Netzwerkes zur Objektdetektion (d.h., nach dem Backbone). Mögliche Werte sind *'high'*, *'medium'* und *'low'*.

Er kann verwendet werden, um zwischen Detektionsleistung und Geschwindigkeit abzuwägen. Für einfachere Objektdetektionsaufgaben können die Einstellungen *'low'* oder *'medium'* ausreichen, um ähnliche Detektionsleistung wie mit *'high'* zu erzielen.

Default: 'capacity' = 'high'

'class_ids_no_orientation':

Mit diesem Parameter können Klassen angegeben werden, für welche die Orientierung nicht berücksichtigt wird, z.B. runde oder andere punktsymmetrische Objekte. Für jede Klasse, deren Klassen-ID in *'class_ids_no_orientation'* präsent ist, gibt das Netzwerk ein achsenparalleles umschließendes Rechteck zurück.

Dieser Parameter betrifft nur Netzwerke mit *'instance_type' = 'rectangle2'*.

Es gilt zu beachten, dass die Werte in *'class_ids_no_orientation'* von jenen in *'class_ids'* abhängen. Falls die Werte in *'class_ids'* nach der Erzeugung des Modells verändert werden, wird *'class_ids_no_orientation'* auf ein leeres Tupel zurückgesetzt.

Default: 'class_ids_no_orientation' = []

'freeze_backbone_level':

Dieser Parameter bestimmt die Backbone Levels deren Gewichte während des Trainings unverändert bleiben (also nicht aktualisiert werden und somit eingefroren sind). Dabei bestimmt die gesetzte Zahl den höchsten Level, dessen Layer unverändert bleiben. Setzt man *'freeze_backbone_level'* auf *0*, so sind die Layer keines Levels eingefroren, es werden die Gewichte aller Layer aktualisiert. Dies wird empfohlen, falls die Gewichte zufällig initialisiert wurden (was z.B. nach gewissen Wechsel der Anzahl Bildkanäle passieren kann) oder im Falle eines nicht vortrainierten Backbones.

Default: 'freeze_backbone_level' = 2

'instance_type':

Der Parameter *'instance_type'* bestimmt, welcher Instanztyp für das Objektdetektion-Modell verwendet wird. Die Implementierungen unterscheiden sich bezüglich der erlaubten Orientierungen der umschließenden Rechtecke. Für weitere Erklärungen der verschiedenen Typen und ihren umschließenden Rechtecke wird auf das Kapitel Deep Learning / Objektdetektion verwiesen.

Mögliche Werte: *'rectangle1'*, *'rectangle2'*

Default: 'instance_type' = 'rectangle1'

'max_level', *'min_level'*:

Diese Parameter bestimmen, an welche Ebenen die zusätzlichen Netzwerke an die Merkmalspyramide angehängt werden. Für weitere Informationen zu Merkmalspyramiden und den angeschlossenen Netzwerken wird auf das Kapitel Deep Learning / Objektdetektion verwiesen.

Von diesen (*'max_level'* - *'min_level'* + 1) Netzwerken werden alle Vorhersagen, die eine minimale Konfidenz und keine zu starke Überlappung aufweisen, beibehalten (siehe *'min_confidence'* und *'max_overlap'*).

Die Ebene gibt an, wie oft die Größe der Merkmalskarte bereits herabskaliert wurde. Somit entspricht Ebene 0 den Merkmalskarten mit Größe des Eingabebildes, Ebene 1 den einmalig herabskalierten Merkmalskarten und so weiter. Kleinere Objekte werden daher besser auf den unteren Ebenen, größere Objekte auf den oberen Ebenen erkannt.

Falls *'max_level'* größer ist als die Anzahl Ebenen des Backbones, wird das Backbone durch zusätzliche (zufällig initialisierte) Faltungs-Layer erweitert um tiefere Ebenen zu generieren. Dies bedeutet, dass *'max_level'* einen Einfluss auf die minimale Größe der Eingabebilder haben kann.

Bei der Eingabe kleiner Bilddimensionen ist zu beachten, dass hohe Ebenen weniger aussagekräftig sein können, da die Merkmalskarten bereits zu klein sein könnten um aussagekräftige Informationen zu

enthalten.
Eine höhere Anzahl verwendeter Ebenen kann zu einer Erhöhung von Laufzeit und Speicherplatz führen, wobei insbesondere die tiefen Ebenen ins Gewicht fallen.
Default: 'max_level' = 6, 'min_level' = 2

'max_num_detections':
Dieser Parameter bestimmt die maximale Anzahl vom Netzwerk vorgeschlagener Detektionen (umschließende Rechtecke) pro Bild.
Default: 'max_num_detections' = 100

'max_overlap':
Der maximal zulässige Quotient aus Schnittmenge und Vereinigungsmenge (IoU, von engl. intersection over union) zweier umschließender Rechtecke derselben Klasse. Anders ausgedrückt, wenn zwei umschließende Rechtecke derselben Klasse zugeordnet werden und einen IoU höher als *'max_overlap'* aufweisen, wird das Rechteck mit dem niedrigeren Konfidenzwert unterdrückt. Weitere Informationen zu IoU sind im Kapitel Deep Learning / Objektdetektion zu finden.
Default: 'max_overlap': = 0.5

'max_overlap_class_agnostic':
Der maximal zulässige Quotient aus Schnittmenge und Vereinigungsmenge (IoU, von engl. intersection over union) zweier Rechtecke unabhängig ihrer Klasse. Anders ausgedrückt, wenn zwei umschließende Rechtecke einen IoU höher als *'max_overlap_class_agnostic'* aufweisen, wird das Rechteck mit dem niedrigeren Konfidenzwert unterdrückt. Der Standardwert für *'max_overlap_class_agnostic'* liegt bei 1.0, wodurch die Klassenagnostische Unterdrückung von umschließenden Rechtecken keine Auswirkung hat.
Default: 'max_overlap_class_agnostic' = 1.0

'min_confidence':
Dieser Parameter bestimmt bei der Klassifizierung des Bildausschnitts innerhalb des umschließenden Rechtecks die minimale Konfidenz um das vorgeschlagene umschließende Rechteck beizubehalten. Das heißt, wenn `apply_dl_model` aufgerufen wird, wird die Ausgabe aller umschließender Rechtecke mit einem Konfidenzwert kleiner als *'min_confidence'* unterdrückt.
Default: 'min_confidence' = 0.5

'ignore_direction':
Dieser Parameter bestimmt ob für das orientierte umschließende Rechteck auch die Ausrichtung des Objekts innerhalb des umschließenden Rechtecks berücksichtigt wird oder nicht. Falls die Ausrichtung innerhalb des umschließenden Rechtecks nicht berücksichtigt werden soll, kann *'ignore_direction' = 'true'* gesetzt werden, Um das umschließende Rechteck eindeutig zu bestimmen, gelten dabei (aber nur dann) die folgende Konventionen:

- $-\pi/2 <$ *'phi'* $\leq \pi/2$
- *'bbox_length1' > 'bbox_length2'*

Dies ist konsistent zu `smallest_rectangle2`.
Dieser Parameter betrifft nur Netzwerke mit *'instance_type' = 'rectangle2'*.
Mögliche Werte: *'true'*, *'false'*
Default: 'ignore_direction' = 'false'

Modelle mit *'type'='segmentation'*

'ignore_class_ids':
Mit diesem Parameter können eine oder mehrere Klassen auf 'ignore' gesetzt werden. Für weitere Informationen siehe Deep Learning / Semantische Segmentierung. Diese Klassen werden über ihre ID (Integer-Wert) deklariert.
Zu beachten ist, dass eine Klassen-ID nicht gleichzeitig in *'ignore_class_ids'* und *'class_ids'* gesetzt werden kann.

Parameter

▷ **`DLModelHandle`** (input_control) ..... dl_model ⇝ *handle*
Handle des Deep Learning-basierten Modells.

▷ **`GenParamName`** (input_control) ..... attribute.name ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'batch_size'
Werteliste: `GenParamName` ∈ {'alphabet', 'anchor_aspect_ratios', 'anchor_num_subscales', 'anchor_angles', 'backbone', 'backbone_docking_layers', 'batch_size', 'batch_size_multiplier',

'bbox_heads_weight', 'capacity', 'class_heads_weight', 'class_ids', 'class_ids_no_orientation', 'class_names', 'class_weights', 'complexity', 'device', 'extract_feature_maps', 'freeze_backbone_level', 'gpu', 'image_dimensions', 'input_dimensions', 'ignore_class_ids', 'ignore_direction', 'image_height', 'image_num_channels', 'image_range_min', 'image_range_max', 'image_width', 'instance_type', 'learning_rate', 'max_level', 'max_num_detections', 'max_overlap', 'max_overlap_class_agnostic', 'meta_data', 'min_confidence', 'min_level', 'momentum', 'num_classes', 'num_trainable_params', 'optimize_for_inference', 'precision', 'precision_is_converted', 'runtime', 'standard_deviation_factor', 'summary', 'type', 'weight_prior'}

▷ **GenParamValue** (output_control) . . . . . attribute.name(-array) ⇝ *integer* / string / real
Wert des generischen Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_dl_model_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_dl_model`, `set_dl_model_param`

Nachfolger

`set_dl_model_param`, `apply_dl_model`, `train_dl_model_batch`, `train_dl_model_anomaly_dataset`

Siehe auch

`set_dl_model_param`

Modul

Deep Learning Inference

`get_dl_pruning_param` `( : : DLPruningHandle, GenParamName : GenParamValue )`

Auslesen der Parameter des Pruning Handles.

`get_dl_pruning_param` gibt die Parameterwerte von `GenParamName` aus dem Pruning Handle `DLPruningHandle` in `GenParamValue` aus.

Die Werte der folgenden Parameter `GenParamName` können abgefragt werden:

'mode': Bestimmt die Pruning-Methode. Siehe `create_dl_pruning` für weitere Informationen.
Default: 'oracle'

'percentage': Bestimmt, um wie viel das Modell reduziert werden soll. Z.B. für *'mode'='oracle'* bestimmt der Parameter, wie viel Prozent der Kernels entfernt werden sollen, wobei nur entfernbare Kernels berücksichtigt werden.
Default: 10

Zusätzlich können für *'mode'='oracle'* folgende Werte für `GenParamName` abgefragt werden:

'prunable_conv_num': Anzahl der reduzierbaren Convolutional-Layer im verwendeten Netzwerk. Ob ein Layer reduzierbar ist oder nicht hängt sowohl von den Spezifikationen des Layers selbst sowie von der Netzwerk-Architektur ab.

'prunable_conv_layer_name': Namen der reduzierbaren Convolutional-Layer im verwendeten Netzwerk.

'prunable_kernel_num': Anzahl der reduzierbaren Kernel im verwendeten Netzwerk. Die Zahl berücksichtigt, dass jeder Convolutional-Layer mindestens 1 Kernel besitzen muss.

'pruned_kernel_num': Anzahl reduzierter Kernel im verwendeten Netzwerk nach dem Aufruf von `gen_dl_pruned_model`.

'pruned_percentage': Tatsächlicher Prozentsatz reduzierter Kernel im verwendeten Netzwerk nach dem Aufruf von `gen_dl_pruned_model`: das Verhältnis *'pruned_kernel_num'* zu *'prunable_kernel_num'*.

Die folgende Tabelle gibt eine Übersicht, welche Parameter und Werte mit `set_dl_pruning_param` gesetzt und welche mit `get_dl_pruning_param` abgerufen werden können.

Parameter	set	get
'mode'	x	x
'percentage'	x	x
'prunable_conv_num'	–	x
'prunable_conv_layer_name'	–	x
'prunable_kernel_num'	–	x
'pruned_kernel_num'	–	x
'pruned_percentage'	–	x

Parameter

▷ **`DLPruningHandle`** (input_control) . . . . . dl_pruning ⇝ *handle*
Handle mit den Pruning-Daten.

▷ **`GenParamName`** (input_control) . . . . . attribute.name ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'percentage'
Werteliste: `GenParamName` ∈ {'mode', 'percentage', 'prunable_conv_num', 'prunable_conv_layer_name', 'prunable_kernel_num', 'pruned_kernel_num', 'pruned_percentage'}

▷ **`GenParamValue`** (output_control) . . . . . attribute.name(-array) ⇝ *integer* / string / real
Wert des generischen Parameters.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_dl_pruning`, `set_dl_pruning_param`

Siehe auch

`set_dl_pruning_param`

Modul

Deep Learning Training

`read_dl_model` (: : FileName : DLModelHandle)

Lesen eines Deep Learning-Modells aus einer Datei.

Der Operator `read_dl_model` liest ein Deep Learning-Modell. Diese Modelle müssen im HALCON Format oder alternativ im ONNX Format (siehe die Referenz weiter unten) vorliegen. Im zweiten Fall gelten die unten erwähnten Einschränkungen. Als Ergebnis wird das Handle `DLModelHandle` zurückgegeben.

Das Modell wird aus der Datei `FileName` geladen. Die Datei wird hierbei sowohl im Verzeichnis `$HALCONROOT/dl/`, als auch im aktuell genutzten Verzeichnis gesucht. Die Standard-Dateiendung eines Deep Learning-basierten Klassifikators ist *'.hdl'*.

Zu beachten ist, dass die Werte laufzeitspezifischer Parameter nicht in der Datei abgespeichert werden, siehe `write_dl_model`. Folglich werden diese Parameter beim Einlesen des Modells mit ihrem Standardwert (siehe `get_dl_model_param`) initialisiert.

Für weitere Informationen zu Deep Learning-Modellen in HALCON siehe das Kapitel Deep Learning / Modell.

Einlesen eines von HALCONzur Verfügung gestellten Modells

HALCON stellt Neuronale Netzwerke für die Klassifikation und Semantische Segmentierung bereit, welche auf einem großen Datensatz vortrainiert wurden. Diese Neuronalen Netzwerke sind gute Ausgangspunkte, um damit eigene Neuronale Netze für ein individuelles Problem zu trainieren. Für die Anomalieerkennung stellt HALCON initiale Modelle zur Verfügung.

Modelle für die Anomalieerkennung

Die folgenden Netzwerke stehen für die Anomalieerkennung zur Verfügung:

'initial_dl_anomaly_medium.hdl'

Dieses Netzwerk hat einen vergleichsweise niedrigeren Speicherverbrauch und geringere Laufzeit als *'initial_dl_anomaly_large.hdl'*.

Das Netzwerk erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert es spezifische Bildeigenschaften, welche über `get_dl_model_param` abgefragt werden können. Hier werden die Defaultwerte gelistet:

'image_width': 480

'image_height': 480

'image_num_channels': 3

'image_range_min': -2

'image_range_max': 2

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen, verlangt aber, dass die Bildbreite *'image_width'* und Bildhöhe *'image_height'* Vielfache von 32 Pixel sind, woraus sich ein Minimalwert von 32 Pixeln ergibt.

'initial_dl_anomaly_large.hdl'

Dieses Netzwerk ist für komplexere Aufgaben in der Regel besser geeignet als *'initial_dl_anomaly_medium.hdl'*. Dafür ist es rechen- und speicheraufwändiger.

Das Netzwerk erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert es für spezifische Bildeigenschaften, welche über `get_dl_model_param` abgefragt werden können. Hier werden die Defaultwerte gelistet:

'image_width': 480

'image_height': 480

'image_num_channels': 3

'image_range_min': -2

'image_range_max': 2

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen, verlangt aber, dass die Bildbreite *'image_width'* und Bildhöhe *'image_height'* Vielfache von 32 Pixel sind, woraus sich ein Minimalwert von 32 Pixeln ergibt.

Modelle für die Klassifikation

Die folgenden vortrainierten Neuronalen Netzwerke stehen für die Klassifikation und als Backbone bei der Objektdetektion zur Verfügung:

*'pretrained_dl_classifier_alexnet.hdl'***:**

Dieses Neuronale Netzwerk ist geeignet für einfache Klassifikationsaufgaben. Es zeichnet sich durch die Faltungskerne der ersten Faltungs-Layer aus, welche größer sind als bei Netzwerken vergleichbarer Klassifikationsleistung (z.B. *'pretrained_dl_classifier_compact.hdl'*). Dies kann für das Extrahieren von Merkmalskarten von Vorteil sein.

Der Klassifikator erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert das Neuronale Netzwerk spezifische Bildeigenschaften, welche über `get_dl_model_param` abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen der Klassifikator trainiert wurde:

'image_width': 224

'image_height': 224

'image_num_channels': 3

'image_range_min': -127

'image_range_max': 128

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen. Bildbreite *'image_width'* und Bildhöhe *'image_height'* sollten dabei aber 29 Pixel nicht unterschreiten. Durch die Netzwerkarchitektur

wird die Bildgröße nach oben hin nicht beschränkt, aber zunehmende Bildgrößen erhöhen den Speicher- und Laufzeitbedarf erheblich. Jede Änderung der Bildgröße führt zu einem Reinitialisieren der Gewichte in den Fully-connected Layern und macht dadurch ein Nachtrainieren notwendig.
Man kann die Laufzeit dieses Netzwerkes beschleunigen, indem man die Faltungs- und ReLU-Layer miteinander verbindet, siehe `set_dl_model_param` und den Parameter *'fuse_conv_relu'*.

'pretrained_dl_classifier_compact.hdl':
Dieses Neuronale Netzwerk ist besonders effizient bezüglich des Speicherverbrauchs und der Laufzeit.
Der Klassifikator erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert das Neuronale Netzwerk spezifische Bildeigenschaften, welche über `get_dl_model_param` abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen der Klassifikator trainiert wurde:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Dieses Neuronale Netzwerk enthält keinen Fully-connected Layer. Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen. Bildbreite *'image_width'* und Bildhöhe *'image_height'* sollten dabei aber 15 Pixel nicht unterschreiten.

'pretrained_dl_classifier_enhanced.hdl':
Dieses Neuronale Netzwerk hat mehr verborgene Layer als *'pretrained_dl_classifier_compact.hdl'* und ist daher für komplexere Klassifikationsaufgaben in der Regel besser geeignet. Dafür ist dieses Netzwerk rechen- und speicheraufwändiger.
Der Klassifikator erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert das Neuronale Netzwerk spezifische Bildeigenschaften, welche über `get_dl_model_param` abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen der Klassifikator trainiert wurde:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen. Bildbreite *'image_width'* und Bildhöhe *'image_height'* sollten dabei aber 47 Pixel nicht unterschreiten. Durch die Netzwerkarchitektur wird die Bildgröße nach oben hin nicht beschränkt, aber zunehmende Bildgrößen erhöhen den Speicher- und Laufzeitbedarf erheblich. Jede Änderung der Bildgröße führt zu einem Reinitialisieren der Gewichte in den Fully-connected Layern und macht dadurch ein Nachtrainieren notwendig.

'pretrained_dl_classifier_mobilenet_v2.hdl':
Dieser Klassifikator ist ein kleines, energieeffizientes Modell und daher besser für mobile und embedded Bildverarbeitungsanwendungen geeignet.
Der Klassifikator erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert das Neuronale Netzwerk spezifische Bildeigenschaften, welche über `get_dl_model_param` abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen der Klassifikator trainiert wurde:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen. Bildbreite *'image_width'* und Bildhöhe *'image_height'* sollten dabei aber 32 Pixel nicht unterschreiten. Durch die Netzwerkarchitektur wird die Bildgröße nach oben hin nicht beschränkt, aber zunehmende Bildgrößen erhöhen den Speicher- und Laufzeitbedarf erheblich.
Auf der GPU kann die Netzwerkarchitektur stark von speziellen Optimierungen profitieren, ohne welche das Netz deutlich langsamer sein kann.

'pretrained_dl_classifier_resnet50.hdl':
Wie das Neuronale Netzwerk *'pretrained_dl_classifier_enhanced'* ist dieser Klassifikator für komplexere Aufgaben geeignet. Die spezielle Struktur dieses Klassifikators bringt allerdings den Vorteil mit sich, das Training stabiler zu machen und intern robuster zu sein.

Der Klassifikator erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert das Neuronale Netzwerk spezifische Bildeigenschaften, welche über get_dl_model_param abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen der Klassifikator trainiert wurde:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen. Bildbreite *'image_width'* und Bildhöhe *'image_height'* sollten dabei aber 32 Pixel nicht unterschreiten. Durch die Netzwerkarchitektur wird die Bildgröße nach oben hin nicht beschränkt, aber zunehmende Bildgrößen erhöhen den Speicher- und Laufzeitbedarf erheblich. Trotz Fully-connected Layer führt eine Änderung der Bildgröße nicht zu einem Reinitialisieren der Gewichte.

Modelle für die Semantische Segmentierung

Folgende vortrainierte Netzwerke stehen für die Semantische Segmentierung zur Verfügung:

'pretrained_dl_edge_extractor.hdl':

Dieses Neuronale Netzwerk ist für die Kantenextraktion konzipiert und vortrainiert. Folglich ist das Netz auch primär für Zwei-Klassen-Probleme ausgelegt, einer Klasse Kanten und einer Klasse Hintergrund.
Das Neuronale Netzwerk erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert es spezifische Bildeigenschaften, welche über get_dl_model_param abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen das Modell trainiert wurde:

'image_width': 512
'image_height': 512
'image_num_channels': 1
'image_range_min': -127
'image_range_max': 128
'num_classes': 2

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen, verlangt aber, dass die Bildbreite *'image_width'* und Bildhöhe *'image_height'* Vielfache von 16 Pixel sind, woraus sich ein Minimalwert von 16 Pixeln ergibt.

'pretrained_dl_segmentation_compact.hdl':

Dieses Neuronale Netzwerk ist besonders für feine Strukturen geeignet und ist effizient bezüglich des Speicherverbrauchs und der Laufzeit.
Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen, verlangt aber eine minimale Bildbreite *'image_width'* von 21 Pixeln und eine minimale Bildhöhe von *'image_height'* 21 Pixeln.

'pretrained_dl_segmentation_enhanced.hdl':

Dieses Neuronale Netzwerk hat mehr verborgene Layer als *'pretrained_dl_segmentation_compact.hdl'* und ist daher für Segmentierungsaufgaben in komplexeren Szenarien besser geeignet.
Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen, verlangt aber eine minimale Bildbreite *'image_width'* von 47 Pixeln und eine minimale Bildhöhe von *'image_height'* 47 Pixeln.

Einlesen eines Modells im ONNX Format

ONNX Modelle können eingelesen werden, aber dabei gibt es gewisse Punkte zu beachten.

Einschränkungen

Wird ein ONNX Modell eingelesen, gelten folgende Einschränkungen:

- Version 1.8.1 der ONNX Spezifikation wird unterstützt. Dies bedeutet, dass nur Operatoren bis zur ONNX operator set version (OpSetVersion) 13 unterstützt werden. Für Operatoren, die eine höhere OpSetVersion besitzen kann keine Unterstützung garantiert werden. Für weitere Limitationen wird auf die ONNX Operatorenauflistung verwiesen.
- Nur 32 Bit Gleitkommatensoren werden unterstützt.
- Nur Modelle die mit einem SoftMax Layer enden, werden automatisch als Klassifikator erkannt. Alle anderen Modelle werden als generisches Modell betrachtet, d.h., Modelle mit *'type'* = *'generic'*. Mit set_dl_model_param kann der Typ eines solchen Modells geändert werden.

- Die Eingabeknoten des Graphen (Bilder) müssen von vierdimensionaler Form sein: Anzahl Bilder (='batch_size'), 'num_channels', 'image_height' und 'image_width'.

Automatische Transformationen

Nach dem Einlesen eines ONNX Modells mit `read_dl_model` werden folgende Transformationen automatisch durchgeführt:

- Jeder nicht globale Pooling Layer, dessen Ausgabemerkmalskarte die Dimension 1x1 hat, wird in einen globalen Pooling Layer verwandelt. Dies ermöglicht Eingabebilder variabler Größen. Weitere Informationen zum Pooling Layer und möglichen Modi befinden sich im „`Solution Guide on Classification`".
- Layer Paare bestehend aus einem Faltungs-Layer ohne Aktivierung und einem direkt verknüpften Activation Layer mit Aktivierungsmodus ReLU werden fusioniert. Um dies zu tun, darf die Ausgabe des Faltungs-Layers nur vom Activation Layer als Eingabe verwendet werden. Daraus entsteht ein Faltungs-Layer mit ReLU Aktivierung. Weitere Informationen zu Layern und möglichen Modi befinden sich im „`Solution Guide on Classification`".

Unterstützte Operationen

ONNX Modelle mit den folgenden Operationen können von `read_dl_model` gelesen werden:

'Add': Keine Einschränkungen.

'ArgMax': Es gelten folgende Einschränkungen:

- Attribut *'axis'*: Der Wert muss *1* sein.
- Attribut *'keepdims'*: Der Wert muss *1* sein.
- Attribut *'select_last_index'*: Der Wert muss *0* sein.

'AveragePool': Es gelten folgende Einschränkungen:

- Attribut *'count_include_pad'*: Der Wert muss *0* sein.

'BatchNormalization': Keine Einschränkungen.

'Clip': Es gelten folgende Einschränkungen:

- Attribut *'min'*: Der Wert muss *0* sein.
- Attribut *'max'*: Der Wert muss größer als *0* und kleiner als die maximale float Zahl sein.

'Concat': Keine Einschränkungen.

'Conv': Es gelten folgende Einschränkungen:

- Attribut *'pads'*: Padding Werte größer oder gleich der kernel size sind nicht unterstützt.

'Dropout': Keine Einschränkungen.

'Gemm': Es gelten folgende Einschränkungen:

- Attribut *'alpha'*: Der Wert muss *1* sein.
- Attribut *'beta'*: Der Wert muss *1* sein.
- Attribut *'transA'*: Der Wert muss *0* sein.

'GlobalAveragePool': Keine Einschränkungen.

'GlobalMaxPool': Es gelten folgende Einschränkungen:

- Attribut *'dilations'*: Der Wert muss *1* sein.

'LogSoftmax': Es gelten folgende Einschränkungen:

- Attribut *'axis'*: Der Wert muss *1* sein.

'LRN': Keine Einschränkungen. Hinweis: Attribut *'size'* hat keinen Effekt.

'MaxPool': Keine Einschränkungen.

'ReduceMax': Es gelten folgende Einschränkungen:

- Attribut *'axes'*: Der Wert muss *1* sein.
- Attribut *'keepdims'*: Der Wert muss *1* sein.

'Relu': Keine Einschränkungen.

'Resize': Es gelten folgende Einschränkungen:

- Attribut *'mode'*: Nur die Werte *'linear'* oder *'bilinear'* sind erlaubt.
- Attribut *'coordinate_transformation_mode'*: Nur die Werte *'pytorch_half_pixel'* und *'align_corners'* sind erlaubt.
- Inputtensor *'roi'*: Wenn Werte gesetzt sind haben sie keinen Effekt auf die Inferenz.

- Die Attribute *'cubic_coeff_a'*, *'exclude_outside'*, *'extrapolation_value'*, oder *'nearest_mode'* haben keinen Effekt.

'Reshape': Es gelten folgende Einschränkungen:

- Wird nur dann unterstützt wenn es auf einen 'Initializer' angewandt wird und die Ausgabe direkt als Gewicht in einem Layer verwendet wird.

'Sigmoid': Keine Einschränkungen.

'Softmax': Es gelten folgende Einschränkungen:

- attribute *'axis'*: Der Wert muss *1* sein.

'Sum': Keine Einschränkungen.

Außerdem wird das ONNX Modellattribut `'metadata_props'` unterstützt. Dieses wird im Modellparameter *'meta_data'* gespeichert.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname
Defaultwert: 'pretrained_dl_classifier_compact.hdl'
Werteliste: `FileName` ∈ {'initial_dl_anomaly_large.hdl', 'initial_dl_anomaly_medium.hdl', 'pretrained_dl_classifier_alexnet.hdl', 'pretrained_dl_classifier_compact.hdl', 'pretrained_dl_classifier_enhanced.hdl', 'pretrained_dl_classifier_resnet50.hdl', 'pretrained_dl_classifier_mobilenet_v2.hdl', 'pretrained_dl_segmentation_compact.hdl', 'pretrained_dl_segmentation_enhanced.hdl', 'pretrained_dl_edge_extractor.hdl'}
Dateiendung: `.hdl`, `.onnx`

▷ **DLModelHandle** (output_control) . . . . . dl_model ⇝ *handle*
Handle des Deep Learning-Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_dl_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_dl_model_param`, `get_dl_model_param`, `apply_dl_model`, `train_dl_model_batch`, `train_dl_model_anomaly_dataset`

Alternativen

`create_dl_model_detection`

Literatur

Open Neural Network Exchange (ONNX), https://onnx.ai/

Modul

Deep Learning Inference

serialize_dl_model (: : DLModelHandle : SerializedItemHandle)

Serialisiert ein Deep Learning-Modell.

Der Operator `serialize_dl_model` serialisiert ein durch das Handle `SerializedItemHandle` definiertes Deep Learning-Modell. Das serialisierte Modell wird vom Handle `SerializedItemHandle` zurückgegeben und kann durch `deserialize_dl_model` deserialisiert werden. Siehe `fwrite_serialized_item` für eine Einführung zum Grundprinzip von Serialisierung.

Der Operator verhält sich wie write_dl_model mit dem Unterschied, dass keine Datei, sondern ein serialisiertes Element als Ausgabe generiert wird.

Für eine detailliertere Beschreibung sei auf die Dokumentation des Operators write_dl_model verwiesen.

Für weitere Informationen zu Deep Learning-Modellen in HALCON siehe das Kapitel Deep Learning / Modell.

Parameter

▷ **DLModelHandle** (input_control) . . . . . dl_model ⇝ *handle*
Handle des Deep Learning-Modells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_dl_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_dl_model_batch, train_dl_model_anomaly_dataset

Nachfolger

train_dl_model_batch, train_dl_model_anomaly_dataset, fwrite_serialized_item, send_serialized_item

Siehe auch

deserialize_dl_model, apply_dl_model, train_dl_model_batch, train_dl_model_anomaly_dataset

Modul

Deep Learning Inference

set_dl_model_param (: : DLModelHandle, GenParamName, GenParamValue :)

Setzen der Parameter des Deep Learning-basierten Models.

set_dl_model_param setzt die Parameter und Hyperparameter GenParamName des Deep Learning-Modells DLModelHandle auf die Werte in GenParamValue.

Die Werte, die GenParamName annehmen kann, hängen vom Modelltyp ab: Es gibt Parameter, die für jedes Deep Learning-Modell gesetzt werden können, während andere nur für spezielle Modelle gesetzt werden können. Eine Beschreibung der spezifischen Parameter, deren Werte nur gesetzt aber nicht abgefragt werden können, ist weiter unten gegeben. Für alle anderen Parameter ist die spezifische Beschreibung in get_dl_model_param zu finden. In get_dl_model_param wird zudem ein Überblick darüber gegeben, für welche Modelltypen Parameter gesetzt und welche Operatoren dafür genutzt werden können.

Im Folgenden werden die Parameter aufgelistet, die mit set_dl_model_param gesetzt werden können. Dabei bezeichnen die folgenden Symbole den Modelltypen, für welchen der Parameter einflussreich gesetzt werden kann.

- 'Any': Jede Methode
- 'AD': *'type'='anomaly_detection'*
- 'CL': *'type'='classification'*
- 'OD': *'type'='detection'*
- 'SE': *'type'='segmentation'*
- 'Gen': *'type'='generic'*

Auf mehrere Modelltypen anwendbar

'batch_size': **Any**
'batch_size_multiplier': **CL OD SE**
'class_ids': **OD SE**
'device': **Any**
'enable_resizing': **Any**
Wird diese Operation aufgerufen, so werden gewisse Pooling-Layer des Netzwerks umgewandelt. Genauer gesagt werden alle nicht-globalen Pooling-Layer deren Ausgabe-Merkmalskarten mit Größe 1x1 in globale Pooling-Layer umgewandelt. So wird z.B. ein 'average pooling' (Vereinigung durch Mittelwert) in ein 'global average pooling' umgewandelt. Für weitere Informationen zu Pooling-Layern wird auf den „Solution Guide on Classification" verwiesen.
Es gilt zu beachten, ist diese Operation einmal durchgeführt, kann sie nicht mehr rückgängig gemacht werden. Entsprechend ist nur der Wert *'true'* setzbar. Falls das Netzwerk keinen solchen Layer hat, wird nichts getan und die Operation als erfolgreich betrachtet.
'fuse_conv_relu': **Any**
Wird diese Operation aufgerufen, so werden Layer Paare, bestehend aus einem Faltungs-Layer ohne Aktivierung und einem direkt verknüpften Activation Layer mit Aktivierungsmodus ReLu, fusioniert. Um dies zu tun, darf die Ausgabe des Faltungs-Layers nur vom Activation Layer als Eingabe verwendet werden. Daraus entsteht ein Faltungs-Layer mit ReLu Aktivierung. Weitere Informationen zu Layern und möglichen Modi befinden sich im „Solution Guide on Classification".
Es gilt zu beachten, ist diese Operation einmal durchgeführt, kann sie nicht mehr rückgängig gemacht werden. Entsprechend ist nur der Wert *'true'* setzbar. Falls das Netzwerk kein entsprechendes Layer Paar hat, wird nichts getan und die Operation als erfolgreich betrachtet.
'fuse_bn_relu': **Any**
Wird diese Operation aufgerufen, so werden Layer Paare, bestehend aus einem Batchnormalisierungs-Layer ohne Aktivierung und einem direkt verknüpften Activation Layer mit Aktivierungsmodus ReLu, fusioniert. Um dies zu tun, darf die Ausgabe des Batchnormalisierungs-Layers nur vom Activation Layer als Eingabe verwendet werden. Daraus entsteht ein Batchnormalisierungs-Layer mit ReLu Aktivierung. Weitere Informationen zu Layern und möglichen Modi befinden sich im „Solution Guide on Classification".
Es gilt zu beachten, ist diese Operation einmal durchgeführt, kann sie nicht mehr rückgängig gemacht werden. Entsprechend ist nur der Wert *'true'* setzbar. Falls das Netzwerk kein entsprechendes Layer Paar hat, wird nichts getan und die Operation als erfolgreich betrachtet.
'gpu': **Any**
'image_dimensions': **AD CL SE**
'image_height', *'image_width'*: **AD CL SE**
'image_num_channels': **AD CL SE**
'input_dimensions': **AD CL SE**
'learning_rate': **CL OD SE**
'meta_data': **Any**
'momentum': **CL OD SE**
'optimize_for_inference': **CL OD SE**
'precision': **Any**
'precision_is_converted': **Any**
'runtime': **Any**
'runtime_init': **Any**
Wird *'runtime_init'* auf *'immediately'* gesetzt wird der GPU-Speicher initialisiert und das entsprechende Handle erzeugt. Andernfalls passiert das sobald es notwendig ist, was zu signifikant längeren Laufzeiten beim ersten Aufruf von `apply_dl_model` oder `train_dl_model_batch` führen kann.
Wird die Netzwerkarchitektur nachträglich geändert, wird der GPU-Speicher reinitialisiert. Das kann zum Beispiel auftreten wenn *'batch_size'*, *'image_dimensions'* oder *'input_dimensions'* durch nachträgliches Aufrufen von `set_dl_model_param` geändert werden.
Zu beachten ist, dass dieser Parameter keine Auswirkungen hat, wenn:

- Eine CPU genutzt wird, also falls *'runtime'* auf *'cpu'* gesetzt ist.
- Eine AI²-Schnittstelle genutzt wird.

- Die Hardware-Einheit vorher mit *'device'* gesetzt wurde.

'weight_prior': **CL OD SE**

Models mit *'type'*=*'anomaly_detection'*

'complexity'
'standard_deviation_factor'

Models mit *'type'*=*'classification'*

'backbone_docking_layers'
Die Spezifikation der Docking Layer kann bei jedem Klassifikator erfolgen, auch ohne dass diese als Backbone eingelesen werden. Berücksichtigt wird diese Angabe jedoch nur für Backbones der Objektdetektion.
'class_names'
'extract_feature_maps'

Models mit *'type'*=*'detection'*

'freeze_backbone_level'
'max_num_detections'
'max_overlap'
'max_overlap_class_agnostic'
'min_confidence'

Models mit *'type'*=*'segmentation'*

'ignore_class_ids'
'image_range_max', *'image_range_min'*

Models mit *'type'*=*'generic'*

'type'
Dieser Parameter gibt den HALCON-spezifischen Modelltypen an.
Modelle des Typs *'generic'* können sämtliche Modell-Funktionen erfüllen. Jedoch sind mehrere Deep-Learning-Prozeduren auf einen spezifischen *'type'* angewiesen. Der *'type'* kann von *'generic'* auf folgende Werte gesetzt werden:

- *'classification'*
- *'segmentation'*

Beim Setzen des *'type'* wird überprüft, ob alle für Inferenz und Training erforderlichen Layer vorhanden sind. vorhanden sind. Fehlen solche Layer werden sie hinzugefügt.

Achtung

Systemvoraussetzungen: cuDNN und cuBLAS sind notwendig, um *'gpu'* Parameter zu setzen, d.h. um den Parameter `GenParamName` *'runtime'* auf *'gpu'* zu setzen. Für weitere Details siehe „`Installation Guide`“, Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods“.

Parameter

▷ **`DLModelHandle`** (input_control) . . . . . dl_model ⇝ *handle*
Handle des Deep Learning-basierten Modells.

▷ **`GenParamName`** (input_control) . . . . . attribute.name ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'learning_rate'
Werteliste: `GenParamName` ∈ {'alphabet', 'anchor_aspect_ratios', 'anchor_num_subscales', 'anchor_angles', 'backbone', 'backbone_docking_layers', 'batch_size', 'batch_size_multiplier', 'bbox_heads_weight', 'capacity', 'class_heads_weight', 'class_ids', 'class_names', 'class_weights', 'complexity', 'device', 'enable_resizing', 'extract_feature_maps', 'freeze_backbone_level', 'fuse_bn_relu', 'fuse_conv_relu', 'gpu', 'image_dimensions', 'ignore_class_ids', 'ignore_direction', 'image_height',

'image_num_channels', 'image_range_min', 'image_range_max', 'image_width', 'input_dimensions', 'instance_type', 'learning_rate', 'max_level', 'max_num_detections', 'max_overlap', 'max_overlap_class_agnostic', 'meta_data', 'min_confidence', 'min_level', 'momentum', 'num_classes', 'optimize_for_inference', 'runtime', 'runtime_init', 'standard_deviation_factor', 'summary', 'type', 'weight_prior'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *real* / string / integer
Wert des generischen Parameters.
Defaultwert: 0.001
Wertevorschläge: GenParamValue ∈ {1, 2, 3, 50, [80,60], 80, 60, 0.001, -127, 128, 'cpu', 'gpu', 'immediately'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_dl_model_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_dl_model, get_dl_model_param

Nachfolger

get_dl_model_param, apply_dl_model, train_dl_model_batch, train_dl_model_anomaly_dataset

Siehe auch

get_dl_model_param

Modul

Deep Learning Inference

set_dl_pruning_param (: : DLPruningHandle, GenParamName, GenParamValue :)

Setzen der Parameter in einem Pruning Handle.

set_dl_pruning_param setzt den Parameter GenParamName im Pruning Handle DLPruningHandle auf den Wert GenParamValue.

Eine Übersicht sowie eine Beschreibung der setzbaren Parameter ist in get_dl_pruning_param gegeben.

Parameter

▷ **DLPruningHandle** (input_control) . . . . . dl_pruning ⇝ *handle*
Handle mit den Pruning-Daten.

▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'percentage'
Werteliste: GenParamName ∈ {'mode', 'percentage'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *real* / string / integer
Wert des generischen Parameters.
Defaultwert: 10
Wertevorschläge: GenParamValue ∈ {'oracle', 10, 20, 30, 40, 50}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_dl_pruning_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_dl_pruning`, `get_dl_pruning_param`

Siehe auch

`get_dl_pruning_param`

Modul

Deep Learning Training

train_dl_model_batch (: : DLModelHandle,
DLSampleBatch : DLTrainResult)

Trainieren eines Deep Learning-Modells.

Der Operator `train_dl_model_batch` führt einen Trainingsschritt des in `DLModelHandle` enthaltenen Deep Learning-Modells aus. Die aktuellen Zielfunktionswerte werden im Dictionary `DLTrainResult` zurückgegeben.

Das Modell `DLModelHandle` muss von einem der folgenden Typen (*'type'*) sein:

- *'classification'*
- *'detection'*
- *'segmentation'*

Ein Trainingsschritt bedeutet hier das einmalige Aktualisieren der Gewichte, basierend auf dem Batch mit Bildern, der in `DLSampleBatch` gegeben ist. Der Optimierungsalgorithmus, der dazu genutzt wird, ist im Abschnitt „Weitere Informationen zu den Algorithmen“ weiter unten genauer erklärt. Für weitere Informationen zum Trainieren eines Netzwerks, siehe Unterkapitel „Das Netzwerk und der Trainingsprozess“ im Kapitel Deep Learning.

Um ein Modell erfolgreich zu trainieren, müssen die verwendbaren Hyperparameter gesetzt und die Trainingsdaten den Modellanforderungen entsprechend übergeben werden. Für Informationen zu den Hyperparameter wird auf das Kapitel des jeweiligen Modells sowie das allgemeine Kapitel Deep Learning verwiesen. Die Trainingsdaten bestehen aus Bildern und den zugehörigen Informationen. Dieser Operator erwartet einen Batch an Trainingsdaten, übergeben durch ein Tupel von Dictionaries `DLSampleBatch`. Ein solches `DLSample` Dictionary wird aus `DLDataset` für jedes einzelne Bild erzeugt, z.B. durch die Prozedur `gen_dl_samples`. Siehe das Kapitel Deep Learning / Modell für weitere Informationen zu den verwendeten Dictionaries und deren Schlüssel.

Die Anzahl an Bildern in einem `DLSampleBatch` Tupel muss einem Vielfachen der *'batch_size'* entsprechen. Insbesondere auf der GPU wird der Parameter *'batch_size'* durch die Menge vorhandenen Speichers limitiert. Um mehr Bilder in einem Trainingsschritt verarbeiten zu können, kann der Modellparameter *'batch_size_multiplier'* auf einen Wert größer 1 gesetzt werden. Die Anzahl der `DLSample` Dictionaries, die dem Trainings-Operator übergeben werden, beträgt *'batch_size'* mal *'batch_size_multiplier'*. Dabei sei angemerkt, dass ein Trainingsschritt auf einem Batch mit einem *'batch_size_multiplier'* größer 1 eine Approximation eines Trainingsschrittes ist, der auf dem gleichen Batch, aber mit einem *'batch_size_multiplier'* von 1 und einer entsprechend größeren *'batch_size'* berechnet wurde. Beispielsweise kann man Unterschiede erwarten zwischen Werten, die mit einer *'batch_size'* von 4 und *'batch_size_multiplier'* von 2 berechnet wurden, und Werten, die mit einer *'batch_size'* von 8 und *'batch_size_multiplier'* von 1 berechnet wurden, obwohl in beiden Fällen mit der gleichen Anzahl an `DLSample` Dictionaries gearbeitet wurde. Allerdings liefert diese Approximation im Allgemeinen vergleichbar gute Ergebnisse, so dass sie verwendet werden kann, wenn man mit mehr `DLSample` Dictionaries trainieren möchte als der Speicher der GPU erlaubt. In seltenen Fällen kann es vorkommen, dass die Approximation mit einer *'batch_size'* von 1 und einem entsprechend großen *'batch_size_multiplier'* nicht die erwartete Performance zeigt. Das Problem kann in den meisten Fällen gelöst werden, indem die *'batch_size'* auf einen Wert größer 1 gesetzt wird.

Im Ausgabe-Dictionary `DLTrainResult` befinden sich der aktuelle Wert der Gesamt-Zielfunktion unter dem Schlüssel `total_loss` sowie die Werte aller weiteren im Modell enthaltenen Zielfunktionen.

Für Modelle mit *'type'* = *'detection'* sind dies z.B. die Zielfunktionen der Heads aller ausgewählten Ebenen, namentlich der 'Huber Loss' für die Heads zur Bestimmung der Umschließenden Rechtecke und der 'Focal Loss'

für die Heads zur Klassifizierung (siehe auch Deep Learning / Objektdetektion sowie *'max_level'* und *'min_level'* in `get_dl_model_param`).

Weitere Informationen zu den Algorithmen

Ein nichtlinearer Optimierungsalgorithmus wird während des Trainings angewendet, mit dem Ziel den Wert der Gesamt-Zielfunktion zu minimieren. Diese wird basierend auf den Vorhersagen des Neuronalen Netzwerks zum aktuellen Batch von Bildern bestimmt. Der Algorithmus, der zur Optimierung genutzt wird, ist ein stochastisches Gradientenabstiegsverfahren (SGD). Er aktualisiert die Gewichte der Layer vorhergehender Iterationen t, w^t, auf die neuen Werte w^{t+1} für Iteration $t+1$ wie folgt:

$$\begin{aligned} v^{t+1} &= \mu v^t - \lambda \nabla_w L \\ w^{t+1} &= w^t + v^{t+1}. \end{aligned}$$

Hierbei ist λ die Lernrate, μ das Momentum, L die Gesamt-Zielfunktion und $\nabla_w L$ der Gradient der Gesamt-Zielfunktion bezüglich der Gewichte. Die Variable v^t wird genutzt um den Einfluss des Momentums μ miteinzubeziehen.

Die unterschiedlichen Modelle haben verschiedene Zielfunktionen implementiert, welche aufsummiert werden. Dazu kommt der Regularisierungsterm $E_\alpha(w)$, welcher insbesondere größere Gewichte bestraft, und zusammen ergeben sie die Gesamt-Zielfunktion.

Die verschiedenen Arten von Zielfunktionen (englisch 'loss') sind:

`Huber Loss` (Modell mit *'type'='detection'*): Der gesamte 'Huber Loss' bildet sich aus der Summe der Beiträge aller Variablen der umschließenden Rechtecke für jede gefundene Instanz des Batches. Der Beitrag einer einzelnen Variable ist wie folgt definiert:

$$L_{Huber}(x) = \begin{cases} 0.5x^2/\beta & \text{falls } |x| < \beta \\ |x| - 0.5\beta & \text{sonst} \end{cases}.$$

Dabei ist x eine Variable des umschließenden Rechtecks und β ein Parameter, der auf den Wert 0.11 festgelegt ist.

`Focal Loss` (Modell mit *'type'='detection'*): Der 'Focal Loss' bildet sich aus der Summe der Beiträge für jede gefundene Instanz des Batches. Der Beitrag ist für eine einzelne Instanz wie folgt definiert:

$$L_{focal}(p) = -\sum_{c=0}^{C-1} \alpha_t^c \left(1 - p_t^c\right)^\gamma \log\left(p_t^c\right),$$

wobei γ ein Parameter ist, der auf den Wert 2 festgelegt ist. α^c steht für das klassenspezifische Gewicht ('class_weights') der c-ten Klasse steht und p_t, α_t definiert sind als

$$p_t := \begin{cases} p & \text{falls } y = 1 \\ 1 - p & \text{sonst} \end{cases}, \quad \text{und} \quad \alpha_t := \begin{cases} \alpha & \text{falls } y = 1 \\ 1 - \alpha & \text{sonst} \end{cases}.$$

Hierbei ist $p = (p^0, \ldots, p^{C-1})$ ein Tupel der vom Modell geschätzten Wahrscheinlichkeiten für jede der C-vielen Klassen und y_c ein One-Hot-kodierter Zielvektor, der die Klasse der Annotation kodiert.

`Multinomial Logistic Loss` (Modell mit *'type'='classification'*, *'segmentation'*): Der 'Multinomial Logistic Loss' ist auch als 'Cross Entropy Loss' bekannt. Definiert ist der 'Multinomial Logistic Loss' wie folgt:

$$L_{mn}\left(f(x,w)\right) = -\frac{1}{N}\sum_{n=0}^{N-1} \alpha(y_n) \langle y_n, \log\left(f(x_n,w)\right)\rangle$$

Hierbei steht $f(x,w)$ für das vorhergesagte Ergebnis, welches von den Gewichten des Netzwerks w und dem Eingabe-Batch x abhängt. y_n ist ein One-Hot-kodierter Zielvektor, der das Label des n-ten Bildes x_n des Batches x mit N-vielen Bildern kodiert und $\log(f(x_n,w))$ soll als Vektor verstanden werden, sodass $\log$ auf jede Komponente von $f(x_n,w)$ angewendet wird. Der Wert $\alpha(y_n)$ ist eine Gewichtung für die durch y_n gegebene Klasse. Die Gewichtung entspricht dem durch 'class_weights' gesetzten Wert und wird zusätzlich durch die Summe der Gewichte aller Klassen normiert.

Der Regularisierungsterm $E_\alpha(w)$ ist eine gewichtete l^2-Norm, die alle K Gewichte, mit Ausnahme der Bias-Werte, miteinbezieht. Sein Einfluss kann durch α kontrolliert werden. Letzteres steht für den Hyperparameter *'weight_prior'*, der mit `set_dl_model_param` gesetzt werden kann.

$$E_\alpha(w) = \frac{\alpha}{2} \sum_{k=0}^{K-1} |w_k|^2$$

Der Index k läuft hierbei über alle Gewichte des Netzwerks, mit der Ausnahme von Bias-Termen. Diese werden nicht regularisiert. Der Regularisierungsterm $E_\alpha(w)$ schiebt die Gewichte gegen Null und bestraft damit insbesondere größere Gewichte, was effektiv die Komplexität des Modells verringert.

Achtung

Der Operator `train_dl_model_batch` basiert auf Algorithmen, die in manchen Fällen nicht deterministisch sind. Daher können Ergebnisse von mehreren Aufrufen von `train_dl_model_batch` trotz gleicher Eingabewerte leicht voneinander abweichen. Durch Setzen des Parameters *'cudnn_deterministic'* von `set_system` kann dies beeinflusst werden.

Systemvoraussetzungen: Die CPU-Implementierung dieses Operators unterstützt nur spezifische Plattformen. Um diesen Operator auf der GPU auszuführen, indem *'runtime'* auf *'gpu'* gesetzt ist (siehe `get_dl_model_param`), werden cuDNN und cuBLAS benötigt. Für die konkreten Systemvoraussetzungen wird auf den „`Installation Guide`“, Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods“, verwiesen.

Parameter

- ▷ **`DLModelHandle`** (input_control) ..... dl_model ⇝ *handle*
  Handle des Deep Learning-Modells.
- ▷ **`DLSampleBatch`** (input_control) ..... dict-array ⇝ *handle*
  Tupel von Dictionaries mit Eingabebildern und zugehörigen Informationen.
- ▷ **`DLTrainResult`** (output_control) ..... dict ⇝ *handle*
  Dictionary mit den Trainingsergebnissen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `train_dl_model_batch` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_dl_model`, `set_dl_model_param`, `get_dl_model_param`

Nachfolger

`apply_dl_model`

Siehe auch

`apply_dl_model`

Modul

Deep Learning Training

`write_dl_model` (: : DLModelHandle, FileName :)

Abspeichern eines Deep Learning-Modells in eine Datei.

`write_dl_model` speichert das Deep Learning-Modell `DLModelHandle` in der Datei `FileName` ab. Zu beachten ist, dass die Werte der laufzeitspezifischen Parameter *'gpu'*, *'runtime'* und *'runtime_init'* nicht abgespeichert werden.

Die HALCON Standard-Dateiendung eines Deep Learning-Modells ist `'.hdl'`.

Das Modell kann mit `read_dl_model` eingelesen werden.

Für weitere Informationen zu Deep Learning-Modellen in HALCON siehe das Kapitel Deep Learning / Modell.

Parameter

▷ **`DLModelHandle`** (input_control) . . . . . dl_model ⇝ *handle*
Handle des Deep Learning-Modells.

▷ **`FileName`** (input_control) . . . . . filename.write ⇝ *string*
Dateiname
Dateiendung: `.hdl`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_dl_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_dl_model`, `train_dl_model_batch`, `train_dl_model_anomaly_dataset`, `set_dl_model_param`

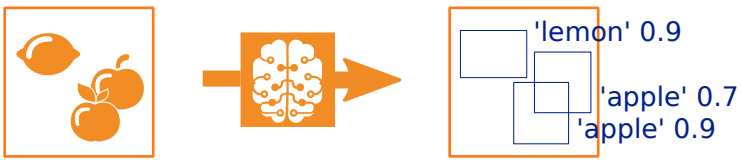
Modul

Deep Learning Inference

8.5 Objektdetektion

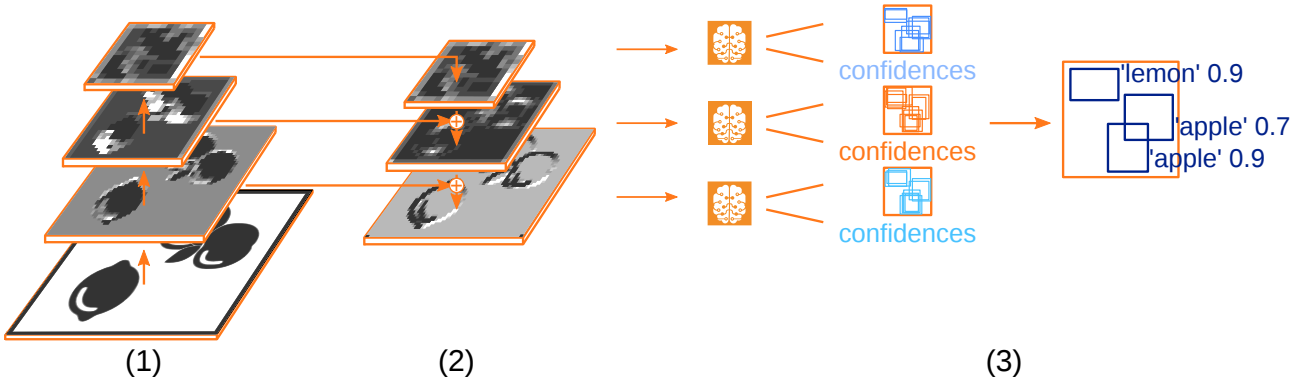
Dieses Kapitel beschreibt, wie Deep Learning-basierte Objektdetektion genutzt wird.

Mit der Objektdetektion will man verschiedene Instanzen auf dem Bild finden und einer Klasse zuordnen. Dabei werden auch partiell überlappende Instanzen als eigenständige Instanzen unterschieden. Dies wird im folgenden Schema illustriert.



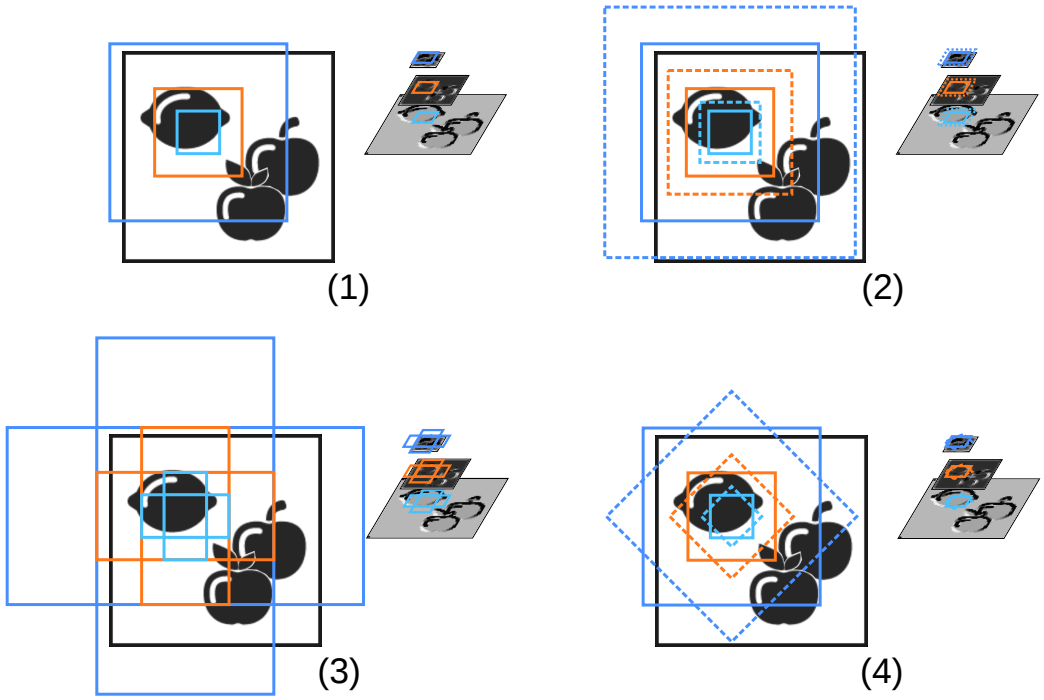
Ein mögliches Beispiel der Objektdetektion: Auf dem Eingabebild werden drei verschiedene Instanzen gefunden und ihrer entsprechenden Klasse zugeordnet.

Objektdetektion führt zu zwei verschiedenen Aufgabenstellungen: Die Instanzen finden und sie einer Klasse zuordnen. Zu diesem Zweck wird ein kombiniertes Netzwerk verwendet, welches aus drei Hauptteilen besteht. Der erste Teil, das Backbone, besteht aus einem vortrainierten Netzwerk, welches auch zur Klassifikation verwendet werden kann. Seine Aufgabe ist das Generieren verschiedener Merkmalskarten, weshalb der Klassifikations-Layer entfernt wurde. Diese Merkmalskarten verschlüsseln verschiedenste Informationen auf verschiedenen Größenskalen, abhängig davon wie tief im Netzwerk sie sind, siehe dazu auch das Kapitel Deep Learning. Dabei ordnet man Merkmalskarten gleicher Größe (Breite und Höhe) der gleichen Ebene zu. Im zweiten Teil werden ausgewählte Merkmalskarten miteinander verbunden. Genauer, Backbone-Layers verschiedener Ebenen sind als Docking Layer festgelegt. Ihre Merkmalskarten werden miteinander verbunden. Daraus entstehen neue Merkmalskarten, die sowohl Informationen von tiefer als auch höher liegenden Ebenen beinhalten. Diese Merkmalskarten werden im dritten Teil verwendet. Den zweite Teil nennt man auch Merkmalspyramide und zusammen mit dem ersten Teil bilden sie das Merkmalspyramiden-Netzwerk. Der dritte Teil besteht aus zusätzlichen Netzwerken, den Heads, für jede ausgewählte Ebene. Diese kriegen je eine der Merkmalskarten als Eingabe und lernen, potentielle Objekte zu finden, respektive zu klassifizieren. Zusätzlich beinhaltet dieser dritte Teil auch die Reduktion sich überlappender umschließender Rechtecke. Eine Übersicht zu den drei Teilen ist im folgenden Schema gegeben.



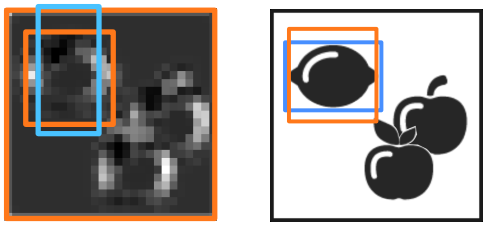
Schematische Übersicht der genannten drei Teile: (1) Das Backbone. (2) Merkmalskarten des Backbones werden miteinander kombiniert, woraus neue Merkmalskarten entstehen. (3) Zusätzliche Netzwerke, die Heads, lernen, potentielle Objekte zu erkennen und zu klassifizieren. Überlappende umschließende Rechtecke werden unterdrückt.

Der dritte Teil soll nun genauer betrachtet werden. In der Objektdetektion wird die Stelle einer Instanz im Bild durch ein umschließendes Rechteck angegeben. Somit besteht die erste Aufgabe aus dem Finden eines passenden umschließenden Rechtecks für jede Instanz. Dafür generiert das Netzwerk umschließende Referenzrechtecke und lernt, wie diese anzupassen sind, um bestmöglich zu den Instanzen zu passen. Diese Referenzrechtecke werden Anchors genannt. Je näher die Anchors bereits an die Gestalt der verschiedenen Ground Truth umschließenden Rechtecke kommen, desto einfacher ist es für das Netzwerk die korrekten Formen zu lernen. Zu diesem Zweck generiert das Netzwerk einen Satz Anchors auf jedem Anchor-Punkt, d.h., auf jedem Pixel aller verwendeten Merkmalskarten der Merkmalspyramide. Ein solcher Satz besteht aus jeder möglichen Kombination aus Form, Größe und für den Instanztyp *'rectangle2'* auch Orientierung. Dabei wird die Form über den Parameter *'anchor_aspect_ratios'*, die Größe über den Parameter *'anchor_num_subscales'* und die Orientierung über den Parameter *'anchor_angles'* beeinflusst, siehe die unten folgende Illustration und `get_dl_model_param`. Falls die gewählten Parameter mehrere identische Anchors erzeugen, werden Duplikate vom Netzwerk intern ignoriert.



Schema der Anchors auf der Merkmalskarte (rechts) und dem Eingabebild (links). (1) Anchors werden auf den Merkmalskarten der verschiedenen Ebenen generiert, z.B. die gezeigten Referenzrechtecke (in Hellblau, in Orange und in Dunkelblau). (2) Anchors verschiedener Größen werden generiert durch Setzen von *'anchor_num_subscales'*. (3) Anchors verschiedener Seitenverhältnisse werden generiert durch Setzen von *'apsect_ratios'*. (4) Anchors verschiedener Orientierungen werden generiert durch das Setzen von *'anchor_angles'* (nur für den Instanztyp *'rectangle2'*).

Das Netzwerk sagt die Anpassungen voraus, um aus den Anchors angepasste umschließende Referenzrechtecke zu erhalten, welche besser zu möglichen Instanzen passen. Es lernt dies über seine Heads für die umschließenden Rechtecke, welche die vorgeschlagenen umschließenden Rechtecke mit den Ground Truth umschließenden Rechtecken vergleichen, d.h. mit der Information, wo im Bild sich einzelne Objekte befinden. Eine Illustration ist unten gegeben.



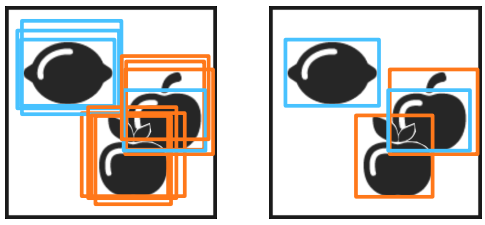
(1) (2)

Darstellung der Vergleiche von umschließenden Rechtecke, illustriert für den Instanztyp *'rectangle1'*. (1) Das Netzwerk passt das umschließenden Referenzrechteck (in Hellblau) an, um ein besser geeignetes umschließendes Rechteck (in Orange) vorauszusagen. (2) Während des Trainings vergleicht das Netzwerk das vorhergesagte umschließende Rechteck (in Orange) mit dem Ground Truth umschließenden Rechteck, mit dem es sich am meisten überlagert, und lernt daraus die Anpassungen.

Wie zuvor erwähnt werden Merkmalskarten verschiedener Ebenen verwendet. Ob sehr frühe Merkmalskarten (die noch nicht sehr komprimiert sind und somit auch kleine Merkmale aufzeigen) und sehr tiefe Merkmalskarten (die sehr komprimiert sind und somit nur große Merkmale aufzeigen) benutzt werden sollen oder nicht, hängt vom Größenverhältnis der gesuchten Instanzen zum Gesamtbild ab. Welche Ebenen berücksichtigt werden, kann über die Parameter *'min_level'* und *'max_level'* bestimmt werden.

Mit diesen umschließenden Rechtecken sind potentielle Instanzen auf dem Bild geortet, aber noch keiner Klasse zugeordnet. Folglich besteht die zweite Aufgabe aus dem Klassifizieren des Bildteils innerhalb des umschließenden Rechtecks. Dies geschieht über die Heads für die Klassen. Für weitere Informationen zur Klassifikation im Allgemeinen wird auf das Kapitel Deep Learning / Klassifikation und den „`Solution Guide on Classification`" verwiesen.

Höchstwahrscheinlich findet das Netzwerk verschiedene vielversprechende Instanzen für ein einzelnes Objekt. Solche sich überlappende, vorhergesagte umschließenden Rechtecke werden über die 'non-maximum suppression' unterdrückt. Letztere kann über die Parameter *'max_overlap'* und *'max_overlap_class_agnostic'* eingestellt werden, entweder direkt beim Erstellen des Modells oder über `set_dl_model_param`. Eine Illustration ist unten gegeben.



(1) (2)

Unterdrückung signifikant überlappender umschließender Rechtecke, illustriert für den Instanztyp *'rectangle1'*. (1) Das Netzwerk findet mehrere vielversprechende Instanzen. (2) Die Unterdrückung sich überlappender Instanzen wird über den Parameter *'max_overlap'* gesetzt. Sich überlappende Instanzen verschiedener Klassen werden nicht unterdrückt. (3) Über den Parameter *'max_overlap_class_agnostic'* werden auch sich stark überlappende Instanzen verschiedener Klassen unterdrückt.

Als Ausgabe werden die umschließenden Rechtecke und die Konfidenzwerte zurück gegeben. Erstere schlagen mögliche Orte für Objekte vor während letztere die Zugehörigkeit des darin enthaltenen Bildinhaltes zu einer der Klassen angibt.

In HALCON ist die Objektdetektion mittels DL als Teil des allgemeinen DL Modells implementiert. Für weitere Information zu letzterem wird auf das Kapitel Deep Learning / Modell verwiesen. Es sind zwei verschiedene Instanztypen implementiert, welche sich in den möglichen Orientierungen der umschließenden Rechtecke unterscheiden:

Instanztyp *'rectangle1'* Die umschließenden Rechtecke sind achsenparallel.

Instanztyp *'rectangle2'* Die umschließenden Rechtecke können beliebig orientiert sein.

Die folgenden Abschnitte behandeln den generellen Ablauf der Objektdetektion, Informationen zu den involvierten Daten und Parameter sowie Erklärungen zu den Evaluierungsmaßen.

Genereller Ablauf

Dieser Abschnitt beschreibt den generellen Ablauf einer Objektdetektion mittels Deep Learning. Er ist unterteilt in die vier Bereiche Erstellen eines Modells und Vorverarbeitung des Datensatzes, Trainieren des Modells, Evaluierung des trainierten Modells und Inferenz auf neuen Bildern. Dabei wird angenommen, dass der Datensatz bereits gelabelt ist, siehe den Absatz „Daten“ unten. Die HDevelop Beispiels-Reihe `detect_pills_deep_learning` zeigt eine mögliche Anwendung unterteilt in die einzelnen Bereiche.

Erstellen eines Modells und Vorbereiten des Datensatzes Dieser Abschnitt behandelt das Erstellen eines DL Modells für Objektdetektion und die notwendigen Anpassungen des Datensatzes an die Anforderungen des Modells. Die einzelnen Schritte sind im HDevelop Beispiel `detect_pills_deep_learning_1_prepare.hdev` gezeigt.

1. Erstellen eines Modells mit dem Operator
   - `create_dl_model_detection`.

   Dabei müssen zumindest das Backbone und die Anzahl zu unterscheidender Klassen angegeben werden. Weitere Parameter können über das Dictionary `DLModelDetectionParam` angegeben werden. Diese Parameterwerte sollten gut für die spezifische Aufgabe gewählt werden, nicht zuletzt um möglicherweise Speicherbedarf und Laufzeit zu reduzieren. Für weitere Informationen wird auf die Dokumentation des Operators verwiesen. Dem Datensatz entsprechende Anchorparameter können mit Hilfe der Prozedur
   - `determine_dl_model_detection_param`

   geschätzt werden. Falls der Datensatz jedoch nicht repräsentativ ist, was die möglichen Orientierungen während des Trainings betrifft (z.B. aufgrund möglicher Augmentation), müssen die vorgeschlagenen Werte noch entsprechend angepasst werden.

   Es gilt zu beachten, dass nach dem Erstellen eines Modells seine zugrunde liegende Architektur auf die entsprechenden Eingabewerte festgelegt ist. Als Resultat gibt der Operator das Handle *'DLModelHandle'* zurück.

   Alternativ kann auch mit dem Operator
   - `read_dl_model`.

   ein Modell eingelesen werden, welches zuvor mit `write_dl_model` gespeichert wurde.
2. Die Information, was auf welchem Bild des Trainings-Datensatzes gefunden werden soll, muss eingelesen werden. Dies kann geschehen durch Einlesen
   - eines `DLDataset` durch den Operator `read_dict`
   - einer Datei im COCO Format durch die Prozedur `read_dl_dataset_from_coco`, wobei ein Dictionary `DLDataset` erstellt wird.

   Das Dictionary `DLDataset` dient als Datenbank, in welchem alle notwendigen Informationen über die Daten abgespeichert werden. Für mehr Information zu den Daten und die Art und Weise, wie sie übergeben werden, wird auf den unteren Absatz „Daten“ und das Kapitel Deep Learning / Modell verwiesen.
3. Die Daten von `DLDataset` sollen in drei getrennten Datensätze geteilt werden. Dazu kann die Prozedur
   - `split_dl_dataset`.

   verwendet werden. Die resultierende Aufteilung wird für jeden Sample Eintrag von `DLDataset` jeweils über den Schlüssel `split` gespeichert.
4. Das Netzwerk stellt mehrere Anforderungen an die Bilder, wie die Bilddimensionen und den Grauwertbereich. Jeder einzelne Wert kann über den Operator

- get_dl_model_param.

abgefragt werden. Alternativ können alle Werte auf einmal über die Prozedur

- create_dl_preprocess_param_from_model.

abgefragt werden. Die umschließenden Rechtecke der unter *'class_ids_no_orientation'* angegeben Klassen müssen bei der Vorverarbeitung anders behandelt werden. Deshalb sollten diese Klassen spätestens zu diesem Zeitpunkt gesetzt sein.

Der Datensatz kann nun vorverarbeitet werden. Dafür kann die Prozedur

- preprocess_dl_dataset

verwendet werden. Für die Implementierung einer selbst erstellten Vorverarbeitung kann man sich an dieser Prozedur orientieren. Es wird empfohlen, alle für das Training benötigten Bilder vor dem Training vorzuverarbeiten und abzuspeichern, was das Trainieren signifikant beschleunigt.

Die vorverarbeiteten Daten können mit der Prozedur

- dev_display_dl_data

visualisiert werden.

Trainieren des Modells Dieser Abschnitt behandelt das Trainieren des DL Modells für Objektdetektion. Die einzelnen Schritte sind im HDevelop Beispiel detect_pills_deep_learning_2_train.hdev gezeigt.

1. Setzen der Trainingsparameter und Abspeichern derselben im Dictionary TrainingParam. Diese Parameter beinhalten:
   - die Hyperparameter, für eine Übersicht wird auf den unteren Abschnitt „Modellparameter und Hyperparameter“ sowie das Kapitel Deep Learning verwiesen.
   - Parameter zur eventuellen Datenanreicherung.
2. Trainieren des Modells. Dafür kann die Prozedur
   - train_dl_model.

   verwendet werden. Diese Prozedur benötigt:
   - das Handle des Modells, DLModelHandle
   - das Dictionary mit den Informationen über den Datensatz, DLDataset
   - das Dictionary mit den Trainingsparametern, TrainingParam
   - die Angabe, über wie viele Epochen trainiert werden soll.

   Während des Trainings sollte sich der Wert der Gesamt-Zielfunktion verringern.

Evaluierung des trainierten Modells Dieser Abschnitt behandelt das Evaluieren eines DL Modells für Objektdetektion. Die einzelnen Schritte sind im HDevelop Beispiel detect_pills_deep_learning_3_evaluate.hdev gezeigt.

1. Setzen der Modellparameter, welche die Evaluierung beeinflussen.
2. Die Evaluierung kann bequem mit der Prozedur
   - evaluate_dl_model.

   durchgeführt werden. Diese Prozedur erwartet ein Dictionary GenParamEval mit den Evaluierungsparametern. Um die Resultate danach zu visualisieren, muss der Parameter detailed_evaluation auf *'true'* gesetzt sein.
3. Die Ergebnisse der Evaluierung können mit der Prozedur
   - dev_display_detection_detailed_evaluation

   visualisiert werden.

Inferenz auf neuen Bildern Dieser Abschnitt behandelt die Inferenz auf neuen Bildern durch ein DL Modell für Objektdetektion. Die einzelnen Schritte sind im HDevelop Beispiel detect_pills_deep_learning_4_infer.hdev gezeigt.

1. Anfragen der Bildanforderungen des Netzwerks über den Operator

- `get_dl_model_param`

oder die Prozedur

- `create_dl_preprocess_param_from_model`.

2. Setzen der Modellparameter die im unteren Abschnitt „Modellparameter und Hyperparameter“ erwähnt sind, über den Operator

   - `set_dl_model_param`.

   Dabei kann die *'batch_size'* im Allgemeinen unabhängig von der Anzahl der gleichzeitig zu inferierenden Bilder gesetzt werden. Siehe `apply_dl_model` für weitere Informationen, wie man diesen Parameter setzt, um eine größere Effizienz zu erzielen.

3. Generieren des Dictionaries `DLSample` für jedes Bild. Dies kann über die Prozedur

   - `gen_dl_samples_from_images`

   getan werden.

4. Jedes Bild muss auf dieselbe Art und Weise wie für das Training vorverarbeitet werden. Dies kann mit der Prozedur

   - `preprocess_dl_samples`

   getan werden.

5. Anwenden des Modells über den Operator

   - `apply_dl_model`.

6. Die Resultate können aus dem Dictionary *'DLResultBatch'* abgerufen werden.

Daten

Es wird zwischen den Daten für Training und Evaluierung versus Daten für Inferenz unterschieden. Letztere bestehen ausschließlich aus Bildern. Für Erstere wird jedoch für jedes einzelne Objekt die Information benötigt, zu welcher Klasse es gehört und wo im Bild es sich befindet (gegeben über das umschließende Rechteck).

Ein Grundgedanke der Datenhandhabung: Das Modell interagiert mit den Daten über Dictionaries. Dies bedeutet, das Modell erhält die Eingabedaten über das Dictionary `DLSample` und gibt die Ausgabe über das Dictionary `DLResult`, bzw. `DLTrainResult` zurück. Für weitere Informationen zur Datenhandhabung wird auf das Kapitel Deep Learning / Modell verwiesen.

Daten für das Training und die Evaluierung Der Datensatz besteht aus Bildern und den dazugehörigen Informationen. Damit das Modell diese verarbeiten kann, müssen sie bestimmte Anforderungen erfüllen. Informationen zu den Bildanforderungen finden sich im unteren Abschnitt „Bilder“.

Die Trainingsdaten werden dazu genutzt, ein Netzwerk für eine spezifische Aufgabe zu trainieren. Mit Hilfe dieser Daten kann das Netzwerk lernen, welche Klassen es unterscheiden soll, wie deren Vertreter aussehen und wie es diese findet. Die dafür notwendigen Informationen werden dadurch gegeben, dass für jedes Objekt eines jeden Bildes gesagt wird, zu welcher Klasse es gehört und wo auf dem Bild es sich befindet. Getan wird dies über das Klassenlabel und das umschließende Rechteck. Es gibt verschiedene Arten, wie diese Informationen gespeichert und abgerufen werden können. Wie die Daten in HALCON für ein DL Modell vorliegen müssen, wird im Kapitel Deep Learning / Modell erklärt. Kurz gefasst fungiert ein Dictionary `DLDataset` als Datenbank der Informationen, die von den Trainings- und Evaluierungs-Prozeduren benötigt werden. Die Daten können direkt mit Hilfe des MVTec Deep Learning Tools gelabelt werden, erhältlich auf der MVTec Webseite. Dabei wird das Dictionary `DLDataset` direkt erzeugt. Sind die Daten bereits im Standard COCO-Format gelabelt, kann die Prozedur `create_dl_dataset_from_coco` verwendet werden (nur für *'instance_type'* = *'rectangle1'*). Diese formatiert die Daten und erstellt das Dictionary `DLDataset`. Für weitere Informationen zum benötigten Part des COCO-Formats wird auf die Dokumentation der Prozedur verwiesen.

Es sollten genügend Trainingsdaten vorliegen um diese in die drei Untermengen aufzuteilen, welche für das Training, die Evaluierung und das Testen des Netzwerks verwendet werden. Diese Untermengen sind idealerweise unabhängig und gleich verteilt, siehe den Abschnitt „Daten“ im Kapitel Deep Learning.

Es gilt zu beachten, dass das Netzwerk lernen muss, wie es mögliche Positionen und Größen von Instanzen finden soll. Deshalb ist es wichtig, dass auch alle später wichtigen Größen und Positionen im Datensatz repräsentativ vorhanden sind.

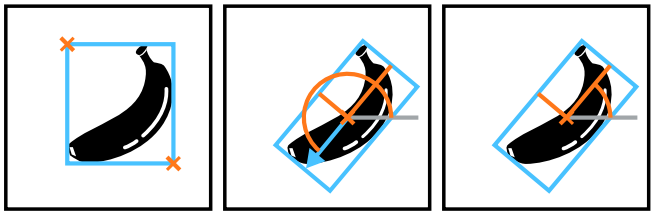
Bilder Unabhängig von der Anwendung stellt das Netzwerk Anforderungen an die Bilder wie z.B. die Bilddimensionen. Die spezifischen Werte hängen vom Netzwerk selbst ab und können mit `get_dl_model_param` abgefragt werden. Um diese Anforderungen zu erfüllen, müssen die Bilder eventuell vorverarbeitet werden. Die Standard-Vorverarbeitung für den ganzen Datensatz und damit auch der Bilder kann mit Hilfe der Prozedur `preprocess_dl_dataset` durchgeführt werden, bzw. `preprocess_dl_samples` für ein einzelnes Sample. Für eine selbst erstellte Vorverarbeitung liefern diese Prozeduren eine Anleitung für die Implementierung.

Umschließende Rechtecke Abhängig vom Instanztyp, werden die umschließenden Rechtecke unterschiedlich parametrisiert:

Instanztyp *'rectangle1'* Die umschließenden Rechtecke werden über das linke obere Eck (*'bbox_row1'*, *'bbox_col1'*) und das rechte untere Eck (*'bbox_row2'*, *'bbox_col2'*) beschrieben. Dies ist konsistent zu `gen_rectangle1`.

Instanztyp *'rectangle2'* Die umschließenden Rechtecke werden über den Schwerpunkt (*'bbox_row'*, *'bbox_col'*), der Orientierung *'bbox_phi'* und den halben Kantenlängen *'bbox_length1'* und *'bbox_length2'* beschrieben. Die Orientierung wird in Bogenmaß angegeben und gibt den Winkel zwischen der horizontalen Achse und *'bbox_length1'* an (mathematisch positiv). Dies ist konsistent zu `gen_rectangle2`.

Ist im Falle von *'rectangle2'* das orientierte umschließende Rechteck von Interesse ohne die Ausrichtung des Objekts innerhalb dieses umschließenden Rechtecks zu berücksichtigen, so kann der Parameter *'ignore_direction'* auf *'true'* gesetzt werden. Die verschiedenen Typen sind in der folgenden Abbildung illustriert.



(1) (2) (3)

Darstellung der umschließenden Rechtecke der verschiedenen Instanztypen: (1) Instanztyp *'rectangle1'*. (2) Instanztyp *'rectangle2'*, wobei das umschließende Rechteck in Richtung Bananenspitze orientiert ist. (3) Instanztyp *'rectangle2'*, wobei das orientierte umschließende Rechteck von Interesse ist, jedoch ohne Berücksichtigung der Ausrichtung der Bananenspitze innerhalb des umschließenden Rechtecks.

Netzwerk Ausgabe Als Trainingsausgabe gibt der Operator `train_dl_model_batch` für das Modell ein Dictionary `DLTrainResult` mit dem aktuellen Wert der Gesamt-Zielfunktion sowie die Werte aller weiteren im Modell enthaltenen Zielfunktionen.

Als Inferenz- und Evaluierungs-Ausgabe gibt der Operator `apply_dl_model` für das Modell für jedes Bild ein Dictionary `DLResult` zurück. Im Falle der Objektdetektion beinhaltet dieses Dictionary für jede gefundene Instanz das umschließende Rechteck sowie den Konfidenzwert der zugeordneten Klasse. Dabei können für ein Objekt des Bildes mehrere Instanzen gefunden werden, siehe die obere Erklärung zur 'non-maximum suppression'. Die umschließenden Rechtecke werden entsprechend dem Instanztyp (über *'instance_type'* bestimmt) parametrisiert und in pixel-zentrierten, subpixel-genauen Koordinaten angegeben. Für weitere Informationen zu den dabei verwendeten Bildkoordinaten wird auf das Kapitel Transformationen / 2D-Transformationen verwiesen. Weitere Informationen zum Ausgabe-Dictionary können im Kapitel Deep Learning / Modell gefunden werden.

Modellparameter und Hyperparameter

Nebst den allgemeinen DL Hyperparametern aus dem Kapitel Deep Learning gibt es für die Objektdetektion noch weitere Hyperparameter:

- *'bbox_heads_weight'*
- *'class_heads_weight'*
- *'class_weights'*

Diese Hyperparameter können über den Operator `create_dl_model_detection` gesetzt werden. Für weitere Informationen wird auf die Dokumentation des Operators `get_dl_model_param` verwiesen.

Für die Objektdetektion gibt es zwei Arten von Modellparametern:

- Parameter, welche die Architektur beeinflussen. Diese können für ein erstelltes Modell nicht mehr geändert werden. Diese Parameter werden beim Erstellen des Modells über den Operator `create_dl_model_detection` gesetzt.
- Parameter, welche die Vorhersage und somit auch die Evaluierung beeinflussen. Dabei sind folgende Parameter nur für die Objektdetektion relevant:
  - *'max_num_detections'*
  - *'max_overlap'*
  - *'max_overlap_class_agnostic'*
  - *'min_confidence'*

  Weitere Erklärungen dazu finden sich in `get_dl_model_param`. Diese Parameter können beim Erstellen des Modells mit `create_dl_model_detection` gesetzt werden oder danach mit `set_dl_model_param`.

Evaluierungsmaße für die Daten der Objektdetektion

Für die Objektdetektion werden in HALCON folgende Evaluierungsmaße unterstützt. Es gilt zu beachten, dass zur Berechnung eines solchen Maßes die zugehörige Ground Truth Annotation notwendig ist.

- Mean average precision, `mAP` und average precision (AP) einer Klasse für einen gegebenen IoU, `ap_iou_classname`

  Die Größe AP ist ein Mittelwert maximaler Precision-Werte für verschiedene Recall-Werte. In einfachen Worten besagt dieses Maß, ob die vorhergesagten Objekte dieser Klasse weitgehend richtige Vorhersagen sind oder nicht. Dabei erhalten Vorhersagen mit hoher Konfidenz ein höheres Gewicht. Je höher der Wert desto besser.

  Damit eine Vorhersage als richtig zählt, muss beides stimmen, die top-1 Klassenzuordnung sowie auch die Lokalisierung im Bild. Die Lokalisierung wird über die Intersection over Union (IoU) gemessen: Eine Instanz zählt als korrekt geortet, wenn der IoU höher als der geforderte Schwellenwert ist. Der IoU wird unten ausführlicher erklärt. Somit ist die Größe AP sowohl von der Klasse als auch dem IoU Schwellenwert abhängig.

  Als Resultate können folgenden Werte erhalten werden: Die spezifischen AP-Werte, die Mittelung über alle Klassen, die Mittelung über die IoU Schwellenwerte sowie die Mittelung über sowohl alle Klassen als auch die Schwellenwerte. Letzteres ist die Größe mean average precision, `mAP`, ein Maß zur Bestimmung wie gut die Instanzen gefunden und klassifiziert wurden.

- 'True Positives', 'False Positives' und 'False Negatives'

  Das Konzept von 'True Positives', 'False Positives' und 'False Negatives' wird im Kapitel Deep Learning erklärt. Es trifft auch bei der Objektdetektion zu, mit der Ausnahme dass es verschiedene Arten von 'False Positives' gibt wie z.B.:
  - Eine Instanz wurde falsch klassifiziert.
  - Eine Instanz wurde auf dem Hintergrund gefunden
  - Die Position einer Instanz wurde ungenügend präzise gefunden, d.h. der IoU zwischen der Instanz und dem Ground Truth Objekt ist kleiner als der bei der Evaluierung verwendete IoU Schwellenwert.
  - Duplikate, d.h. mindestens zwei Instanzen überlappen hauptsächlich mit dem selben Ground Truth Objekt aber überlappen miteinander weniger als durch *'max_overlap'* erlaubt, weshalb keine der Instanzen unterdrückt wird.

  Es gilt zu beachten, dass diese Werte nur aus der detaillierten Evaluierung erhältlich sind. Dies bedeutet, in `evaluate_dl_model` muss der Parameter `detailed_evaluation` auf *'true'* gesetzt sein.

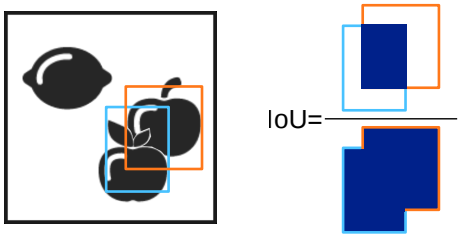
- Score of Angle Precision (SoAP)

  Die Größe SoAP ist ein Maß für die Präzision der inferierten Orientierungswinkel. Dieses Maß bestimmt sich aus der Winkeldifferenzen zwischen den inferierten Instanzen (I) und den zugehörigen Ground Truth Annotationen (GT):

  $\text{SoAP} = 1.0 - \frac{1}{\pi} \cdot \sum_k \left| \varphi_k^{\text{I}} - \varphi_k^{\text{GT}} \right|,$

  wobei der Index k über alle inferierten Instanzen läuft. Diese Größe ist nur für Modelle mit Instanztyp *'instance_type' 'rectangle2'* definiert.

Vorher erklärte Maße verwenden die Intersection over Union (IoU). Die IoU ist ein Maß zur Genauigkeit der Lokalisierung. Für ein vorgeschlagenes umschließendes Rechteck wird das Verhältnis folgenden beiden Flächen betrachtet: Die Fläche der Schnittmenge und die Fläche der Vereinigungsmenge des vorgeschlagenen umschließenden Rechtecks und des Ground Truth umschließenden Rechtecks. Die folgende Abbildung zeigt ein visuelles Beispiel.



(1) (2)

Visuelles Beispiel des IoU, illustriert für den Instanztyp *'rectangle1'*. (1) Im Eingabebild sieht man das Ground Truth umschließende Rechteck (in Orange) und das vorhergesagte umschließende Rechteck (in Hellblau). (2) Die IoU ist das Verhältnis zwischen den beiden Flächen Schnittmenge und Vereinigungsmenge.

```
create_dl_model_detection ( : : Backbone, NumClasses,
    DLModelDetectionParam : DLModelHandle )
```

Erzeugen eines Deep Learning-Netzwerks zur Objektdetektion.

Mit dem Operator `create_dl_model_detection` wird ein Deep Learning-Netzwerk zur Objektdetektion erzeugt. Für weitere Informationen zu Deep Learning-basierter Objektdetektion siehe das Kapitel Deep Learning / Objektdetektion. Das Handle dieses Netzwerks wird in `DLModelHandle` zurückgegeben.

Mit den unten aufgelisteten Parametern können das Modell und dessen Architektur spezifiziert werden. Um erfolgreich ein Modell zu erzeugen müssen dessen Backbone und die Anzahl an Klassen, die das Modell unterscheiden können soll, festgelegt werden. Ersteres wird durch den Parameter `Backbone` übergeben, welcher im Abschnitt „Mögliche Backbones“ genauer erläutert wird. Letzteres ist durch den Parameter `NumClasses` gegeben. Dieser Parameter legt die Anzahl Klassen fest, welche das Netzwerk unterscheiden können soll und damit auch die Anzahl der Einträge in *'class_ids'*.

Die Werte aller verwendbaren Parameter können über das Dictionary `DLModelDetectionParam` spezifiziert werden. Ein solcher Parameter ist z.B. *'instance_type'*, der die Art der im Modell verwendeten umschließenden Rechtecke bestimmt. Die Liste mit den setzbaren Parametern findet sich im Abschnitt „Setzbare Parameter“. Für den Fall, dass ein Parameter nicht spezifiziert ist, wird dessen Standardwert zur Erzeugung des Modells verwendet. Zu beachten ist, dass Parameter, welche die Netzwerkarchitektur beeinflussen, nicht mehr änderbar sind sobald das Netzwerk erzeugt wurde. Alle weiteren Parameter können mit dem Operator `set_dl_model_param` weiterhin gesetzt oder geändert werden. Ein Überblick wie die Parameter gesetzt werden können und ihre Bedeutung findet sich in `get_dl_model_param`. Nach Erzeugung des Objektdetektions-Modells, wird *'type'* automatisch auf *'detection'* gesetzt.

Mögliche Backbones

Der Parameter `Backbone` bestimmt das Backbone, das vom Netzwerk benutzt wird. Detailliertere Informationen zum Backbone sind im Kapitel Deep Learning / Objektdetektion zu finden. Kurz zusammengefasst besteht das Backbone aus einem vortrainierten Klassifikator, von dem nur die Layer zur Generierung der Merkmalskarten behalten werden. Daher gibt es in dem Netzwerk keine Fully Connected Layer. Daraus ergibt sich, dass ein Klassifikator als Merkmalsextraktor für das angeschlossene Erkennungsnetzwerk eingelesen wird. Hierfür kann

ein Modell im HALCON Format oder im ONNX Format eingelesen werden, siehe `read_dl_model` für weitere Informationen.

`create_dl_model_detection` hängt eine Merkmalspyramide an verschiedene Ebenen des Backbones an. Genauer, im Backbone ist für verschiedene Ebenen ein Layer als Docking Layer spezifiziert. Wird ein Detektions-Modell erstellt, so wird die Merkmalspyramide an den entsprechenden Docking Layern angedockt. Bei vortrainierten Klassifikatoren, die von HALCON zur Verfügung gestellt werden, sind die Docking Layers bereits spezifiziert. Wird ein selbsterstellter Klassifikator als Backbone verwendet, müssen die Docking Layer jedoch spezifiziert werden. `backbone_docking_layers` kann als Eigenschaft des Klassifikators über `set_dl_model_param` oder als Eigenschaft des Backbones über diesen Operator gesetzt werden.

Die Docking Layer gehören zu verschiedenen Ebenen und entsprechend haben ihre Merkmalskarten in der Merkmalspyramide verschiedene Größen. Genauer, in der Merkmalspyramide werden die Längen der Merkmalskarten von Ebene zu Ebene halbiert. Dies bedeutet, dass auch die Längen der Eingabebilder für jede weitere Ebene halbiert werden. Die Netzwerkarchitekturen erlauben Änderungen bezüglich der Bilddimensionen, aber die Dimensionen *'image_width'* und *'image_height'* müssen ein ganzzahliges Mehrfaches von 2^{level} sein. Dabei steht $level$ für die höchste Ebene, an welche die Merkmalspyramide anbaut. Dies hängt sowohl von den zusätzlichen Netzwerken als auch den Docking Layern ab. Für die zur Verfügung gestellten Klassifikatoren wird in der unten stehenden Liste erwähnt, bis zu welcher Ebene die Merkmalspyramide standardmäßig geht.

Folgende vortrainierte Klassifikatoren werden von HALCON zur Verfügung gestellt und können als Backbone eingelesen werden:

'pretrained_dl_classifier_alexnet.hdl':
Dieses Neuronale Netzwerk ist geeignet für einfache Klassifikationsaufgaben. Es zeichnet sich durch die Faltungskerne der ersten Faltungs-Layer aus, welche größer sind als bei Netzwerken vergleichbarer Klassifikationsleistung (z.B. *'pretrained_dl_classifier_compact.hdl'*). Dies kann für das Extrahieren von Merkmalskarten von Vorteil sein.

Dieses Backbone erwartet Bilder des Typs *'real'*. Zusätzlich ist das Backbone auf gewisse Bildeigenschaften ausgelegt. Die entsprechenden Werte können mit `get_dl_model_param` abgerufen werden. Hier sind die Standardwerte, mit denen der Klassifikator trainiert wurde, aufgelistet:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Standardmerkmalspyramide geht bei diesem Backbone bis zur 4. Ebene.

'pretrained_dl_classifier_compact.hdl':
Dieses Neuronale Netzwerk ist dafür entworfen speicher- und laufzeiteffizient zu sein. Dieses Backbone erwartet Bilder des Typs *'real'*. Zusätzlich ist das Backbone auf gewisse Bildeigenschaften ausgelegt. Die entsprechenden Werte können mit `get_dl_model_param` abgerufen werden. Hier sind die Standardwerte, mit denen der Klassifikator trainiert wurde, aufgelistet:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Standardmerkmalspyramide geht bei diesem Backbone bis zur 4. Ebene.

'pretrained_dl_classifier_enhanced.hdl':
Dieses Neuronale Netzwerk hat mehr verborgene Layer als *'pretrained_dl_classifier_compact.hdl'* und es kann daher angenommen werden, dass es sich besser für komplexe Aufgaben eignet. Allerdings werden dadurch mehr Laufzeit und Speicherplatz benötigt.

Dieses Backbone erwartet Bilder des Typs *'real'*. Zusätzlich ist das Backbone auf gewisse Bildeigenschaften ausgelegt. Die entsprechenden Werte können mit `get_dl_model_param` abgerufen werden. Hier sind die Standardwerte, mit denen der Klassifikator trainiert wurde, aufgelistet:

'image_width': 224
'image_height': 224

'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Standardmerkmalspyramide geht bei diesem Backbone bis zur 5. Ebene.

'pretrained_dl_classifier_mobilenet_v2.hdl':
Dieser Klassifikator ist ein kleines, energieeffizientes Modell und daher besser für mobile und embedded Bildverarbeitungsanwendungen geeignet.

Dieses Backbone erwartet Bilder des Typs *'real'*. Zusätzlich ist das Backbone auf gewisse Bildeigenschaften ausgelegt. Die entsprechenden Werte können mit `get_dl_model_param` abgerufen werden. Hier sind die Standardwerte, mit denen der Klassifikator trainiert wurde, aufgelistet:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Standardmerkmalspyramide geht bei diesem Backbone bis zur 4. Ebene.

'pretrained_dl_classifier_resnet50.hdl':
Wie das Neuronale Netzwerk *'pretrained_dl_classifier_enhanced.hdl'* ist dieses Neuronale Netzwerk für komplexere Aufgaben ausgelegt. Da sich die Struktur aber unterscheidet, entsteht hier der Vorteil, dass das Training stabiler und das Netzwerk intern robuster ist.

Dieses Backbone erwartet Bilder des Typs *'real'*. Zusätzlich ist das Backbone auf gewisse Bildeigenschaften ausgelegt. Die entsprechenden Werte können mit `get_dl_model_param` abgerufen werden. Hier sind die Standardwerte, mit denen der Klassifikator trainiert wurde, aufgelistet:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Standardmerkmalspyramide geht bei diesem Backbone bis zur 5. Ebene.

Setzbare Parameter

Parameter, die bei der Erzeugung eines Modells mit `create_dl_model_detection` gesetzt werden können (siehe `get_dl_model_param` für Erklärungen):

- *'anchor_angles'*
- *'anchor_aspect_ratios'* (veraltet: *'aspect_ratios'*)
- *'anchor_num_subscales'* (veraltet: *'num_subscales'*)
- *'backbone_docking_layers'*
- *'bbox_heads_weight'*, *'class_heads_weight'*
- *'capacity'*
- *'class_ids'*
- *'class_ids_no_orientation'*
- *'class_weights'*
- *'freeze_backbone_level'*
- *'ignore_direction'*
- *'image_dimensions'*
- *'image_height'*, *'image_width'*
- *'image_num_channels'*
- *'instance_type'*
- *'max_level'*, *'min_level'*

- *'max_num_detections'*
- *'max_overlap'*
- *'max_overlap_class_agnostic'*
- *'min_confidence'*
- *'optimize_for_inference'*

Achtung

Systemvoraussetzungen: Zum erfolgreichen Setzen von *'gpu'* Parametern werden cuDNN und cuBLAS benötigt, d.h. zum Setzen des Parameters `GenParamName` *'runtime'* auf *'gpu'*. Für weitere Details wird auf den „`Installation Guide`", Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods", verwiesen.

Parameter

▷ **`Backbone`** (input_control) . . . . . filename.read ⇝ *string*
Deep Learning Klassifikator, genutzt als Backbone-Netzwerk.
Defaultwert: 'pretrained_dl_classifier_compact.hdl'
Werteliste: `Backbone` ∈ {'pretrained_dl_classifier_alexnet.hdl', 'pretrained_dl_classifier_compact.hdl', 'pretrained_dl_classifier_enhanced.hdl', 'pretrained_dl_classifier_mobilenet_v2.hdl', 'pretrained_dl_classifier_resnet50.hdl'}
Dateiendung: `.hdl`

▷ **`NumClasses`** (input_control) . . . . . integer ⇝ *integer*
Anzahl von Klassen.
Defaultwert: 3

▷ **`DLModelDetectionParam`** (input_control) . . . . . dict ⇝ *handle*
Parameter für das Objektdetektion-Modell.
Defaultwert: []

▷ **`DLModelHandle`** (output_control) . . . . . dl_model ⇝ *handle*
Deep Learning-Modell für die Objektdetektion.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_dl_model_detection` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_dl_model_param`, `get_dl_model_param`, `apply_dl_model`, `train_dl_model_batch`

Alternativen

`read_dl_model`

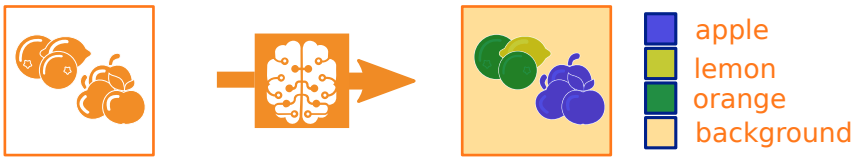
Modul

Deep Learning Training

8.6 Semantische Segmentierung

Dieses Kapitel beschreibt, wie Deep Learning-basierte Semantische Segmentierung in der Trainings- und Inferenzphase genutzt wird.

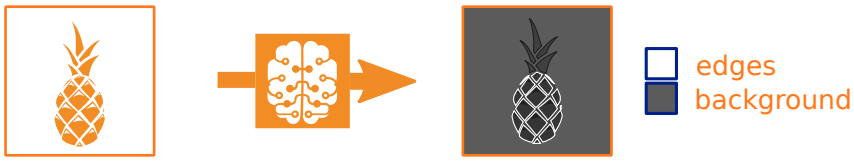
In der Semantischen Segmentierung ordnet ein Deep Learning (DL) Netzwerk jedem Pixel des Eingabebildes eine Klasse zu.



Ein mögliches Beispiel der Semantischen Segmentierung: Jedem Pixel des Eingabebildes wird eine Klasse zugeordnet, dabei sind weder die drei Instanzen der Klasse 'apple' noch die zwei Instanzen der Klasse 'orange' unterschiedene Objekte.

Das Resultat der Semantischen Segmentierung ist ein Bild, in welchem die Pixelwerte für die zugeordnete Klasse des entsprechenden Pixels auf dem Eingabebild stehen. Somit hat das Ausgabebild in HALCON die gleiche Größe wie das Eingabebild. Im Allgemeinen sind im Netzwerk die Merkmalskarten, die komplexere Merkmale verkörpern, kleiner als das Eingabebild (siehe den Abschnitt „Das Netzwerk und der Trainingsprozess“ im Kapitel Deep Learning). Damit das Ausgabebild die gleiche Größe wie das Eingabebild hat, werden in HALCON Netzwerke mit den folgenden beiden Komponenten verwendet: Einem Encoder und einem Decoder. Der Encoder bestimmt die Merkmale des Eingabebildes, wie dies z.B. in der DL-basierten Klassifikation getan wird. Da diese Information in einem komprimierten Format 'kodiert' ist, braucht es einen Decoder, welcher das gewünschte Ergebnis rekonstruiert. Bei der Semantischen Segmentierung ist dies die Zuordnung jedes Pixels zu einer Klasse. Da das Netzwerk die einzelnen Pixel klassifiziert, werden sich überlappende Instanzen der gleichen Klasse nicht als eigenständig unterschieden.

Die Kantenextraktion ist ein Spezialfall der Semantischen Segmentierung. Das Modell ist trainiert zwei Klassen zu unterscheiden: Kanten ('edge') und Hintergrund ('background'). Für weitere Informationen siehe „`Solution Guide I - Basics`“.



Ein mögliches Beispiel der Kantenextraktion: Ein Spezialfall der Semantischen Segmentierung, bei dem jedem Pixel des Eingabebildes eine der Klassen 'edge' und 'background' zugeordnet wird.

In HALCON ist die Semantische Segmentierung mittels DL als Teil des allgemeinen DL Modells implementiert. Für weitere Informationen zum DL Modell wird auf das Kapitel Deep Learning / Modell verwiesen.

Die folgenden Abschnitte behandeln den generellen Ablauf der Semantischen Segmentierung, Informationen zu den involvierten Daten und Parametern, sowie Erklärungen zu den Evaluierungsmaßen.

Genereller Ablauf

Dieser Abschnitt beschreibt den generellen Ablauf einer Semantischen Segmentierung mittels Deep Learning. Er ist unterteilt in die vier Bereiche Vorverarbeitung des Datensatzes, Trainieren des Modells, Evaluierung des trainierten Modells und Inferenz auf neuen Bildern. Dabei wird angenommen, dass der Datensatz bereits gelabelt ist, siehe den Absatz „Daten“ unten. Die HDevelop Beispiels-Reihe `segment_pill_defects_deep_learning` zeigt eine mögliche Anwendung unterteilt in die einzelnen Bereiche.

Vorverarbeitung des Datensatzes Dieser Abschnitt behandelt die notwendigen Anpassungen des Datensatzes an die Anforderungen eines Modells der Semantischen Segmentierung. Die einzelnen Schritte werden im HDevelop Beispiel `segment_pill_defects_deep_learning_1_preprocess.hdev` dargestellt.

1. Die Information, was auf welchem Bild des Trainings-Datensatzes gefunden werden soll, muss eingelesen und konvertiert werden. Dazu kann die Prozedur

   - `read_dl_dataset_segmentation`

   verwendet werden. Dabei wird ein Dictionary `DLDataset` erstellt, welches als Datenbank dient und alle notwendigen Informationen über die Daten abspeichert. Für mehr Information zu den Daten und wie sie übergeben werden, wird auf den unteren Absatz „Daten“ und das Kapitel Deep Learning / Modell verwiesen.

2. Die Daten von `DLDataset` sollen in drei getrennten Datensätze geteilt werden. Dazu kann die Prozedur

   - `split_dl_dataset`

   verwendet werden. Die resultierende Aufteilung wird für jeden Sample Eintrag von `DLDataset` jeweils über den Schlüssel `split` gespeichert.

3. Der Datensatz kann nun vorverarbeitet werden. Dazu kann die Prozedur

   - `preprocess_dl_dataset`

   verwendet werden. Für die Implementierung einer selbst erstellten Vorverarbeitung kann man sich an dieser Prozedur orientieren.

   Um diese Prozedur zu verwenden, müssen die Vorverarbeitungsparameter wie z.B. die Bildgröße festgelegt werden. Für letztere sollte die kleinstmögliche Bildgröße gewählt werden, mit der die einzelnen Regionen noch gut erkennbar sind. Die Parameter und ihre Werte sind im Dictionary `DLPreprocessParam` zu speichern, wofür die Prozedur

   - `create_dl_preprocess_param`

   verwendet werden kann. Es wird empfohlen, dieses Dictionary `DLPreprocessParam` abzuspeichern, um während der späteren Inferenz-Phase Zugang zu den Werten zu haben.

   Während der Vorverarbeitung werden von der Prozedur `preprocess_dl_dataset` auch die Bilder `weight_image` für den Trainings-Datensatz erstellt. Diese ordnen jeder Klasse das Gewicht (’class weights’) zu, welche ihre Pixel während des Trainings erhalten (siehe den unteren Abschnitt „Modellparameter und Hyperparameter“).

Trainieren des Modells Dieser Abschnitt behandelt das Trainieren des DL Modells für Semantische Segmentierung. Die einzelnen Schritte werden im HDevelop Beispiel `segment_pill_defects_deep_learning_2_train.hdev` dargestellt.

1. Einlesen eines Modells mit dem Operator

   - `read_dl_model`.

2. Die Modellparameter müssen mit dem Operator

   - `set_dl_model_param`

   gesetzt werden. Solche Parameter sind z.B. *’image_dimensions’* und *’class_ids’*, siehe die Dokumentation von `get_dl_model_param`.

   Die aktuellen Werte können jederzeit über den Operator

   - `get_dl_model_param`

   abgefragt werden.

3. Setzen der Trainingsparameter und Abspeichern derselben im Dictionary *’TrainParam’*. Diese Parameter beinhalten:

   - die Hyperparameter. Für eine Übersicht wird auf den unteren Abschnitt „Modellparameter und Hyperparameter“ sowie das Kapitel Deep Learning verwiesen.
   - Parameter zur eventuellen Datenanreicherung (optional).
   - Parameter zur Evaluierung während des Trainings.
   - Parameter zur Visualisierung von Trainingsergebnissen.
   - Parameter zur Serialisierung.

   Für dies kann die Prozedur

   - `create_dl_train_param`

   verwendet werden.

4. Trainieren des Modells. Dafür kann die Prozedur

   - `train_dl_model`

   verwendet werden. Diese Prozedur benötigt:

   - das Handle des Modells, `DLModelHandle`
   - das Dictionary mit den Informationen über den Datensatz, `DLDataset`

- das Dictionary mit den Trainingsparametern, *'TrainingParam'*
- die Angabe, über wie viele Epochen trainiert werden soll.

Wird die Prozedur `train_dl_model` verwendet, werden die Gesamt-Zielfunktion sowie optionale Evaluierungsmaße visualisiert.

Evaluierung des trainierten Modells Dieser Abschnitt behandelt das Evaluieren eines DL Modells für Semantische Segmentierung. Die einzelnen Schritte werden im HDevelop Beispiel `segment_pill_defects_deep_learning_3_evaluate.hdev` dargestellt.

1. Setzen der Modellparameter, welche die Evaluierung beeinflussen, wie z.B. *'batch_size'*, über den Operator
   - `set_dl_model_param`.
2. Die Evaluierung kann bequem mit der Prozedur
   - `evaluate_dl_model`

   durchgeführt werden.
3. Das Dictionary `EvaluationResults` enthält die angefragten Evaluierungsmaße. Die Ergebnisse der Evaluierung können mit der Prozedur
   - `dev_display_segmentation_evaluation`

   visualisiert werden.

Inferenz auf neuen Bildern Dieser Abschnitt behandelt die Inferenz auf neuen Bildern durch ein DL Modell für Semantische Segmentierung. Die einzelnen Schritte werden im HDevelop Beispiel `segment_pill_defects_deep_learning_4_infer.hdev` dargestellt.

1. Setzen der Parameter, wie z.B. *'batch_size'* über den Operator
   - `set_dl_model_param`.
2. Generieren des Dictionaries `DLSample` für jedes Bild. Dafür kann die Prozedur
   - `gen_dl_samples_from_images`

   verwendet werden.
3. Jedes Bild muss auf dieselbe Art und Weise wie für die Trainingsbilder vorverarbeitet werden. Es wird empfohlen, dafür die Prozedur
   - `preprocess_dl_samples`

   zu verwenden. Wurde während der Vorverarbeitung das Dictionary `DLPreprocessParam` gespeichert, kann es direkt als Eingabe verwendet werden, um die Parameterwerte festzulegen.
4. Anwenden des Modells über den Operator
   - `apply_dl_model`.
5. Die Resultate können aus dem Dictionary *'DLResultBatch'* abgerufen werden. Die Regionen der einzelnen Klassen kann man z.B. über den Operator `threshold` selektieren.

Daten

Es wird zwischen den Daten für Training und Evaluierung versus Daten für Inferenz unterschieden. Letztere bestehen ausschließlich aus Bildern. Für Erstere wird jedoch die Information benötigt, zu welcher Klasse jeder einzelne Pixel gehört (über `segmentation_image`, siehe die Erklärungen weiter unten).

Ein Grundgedanke der Datenhandhabung: Das Modell interagiert mit den Daten über Dictionaries. Dies bedeutet, das Modell erhält die Eingabedaten über das Dictionary `DLSample` und gibt die Ausgabe über das Dictionary `DLResult`, bzw. `DLTrainResult` zurück. Für weitere Informationen zur Datenhandhabung wird auf das Kapitel Deep Learning / Modell verwiesen.

Daten für das Training und die Evaluierung Die Trainingsdaten werden benötigt, um das Netzwerk für Ihre spezifische Aufgabe zu trainieren. Der Datensatz besteht aus Bildern und den dazugehörigen Informationen. Damit das Modell diese verarbeiten kann, müssen sie bestimmte Anforderungen erfüllen. Informationen zu den Bildanforderungen finden sich im unteren Abschnitt „Bilder“. Die Information über die Bilder und ihre Ground Truth Annotationen wird über das Dictionary `DLDataset` und für jedes Sample das entsprechende Bild `segmentation_image` gegeben, welches für jedes Pixel die Klasse definiert.

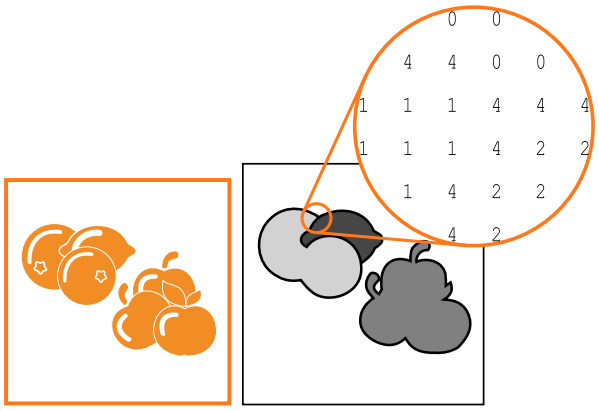
Klassen Die verschiedenen Klassen bilden die vom Netzwerk unterschiedene Sets oder die unterschiedenen Kategorien. Diese Klassen werden im Dictionary `DLDataset` gesetzt und dem Modell über den Operator `set_dl_model_param` gegeben.

Bei der Semantischen Segmentierung gibt es zwei Besonderheiten: Die Hintergrund-Klasse 'background' und Klassen, die ignoriert werden sollen und deshalb auf 'ignore' gesetzt werden:

- 'background' Klasse: Das Netzwerk behandelt die Hintergrund-Klasse 'background' wie jede andere Klasse auch. Ebenso ist es nicht notwendig, eine Hintergrund-Klasse zu haben. Aber falls im Datensatz verschiedene Klassen vorkommen, deren Ergebnis nicht von Interesse ist, die aber trotzdem vom Netzwerk gelernt werden sollen, können diese alle als Hintergrund deklariert werden. Als Resultat davon wird die Hintergrund-Klasse diverser. Für weitere Informationen wird auf die Prozedur `preprocess_dl_samples` verwiesen.
- 'ignore' Klassen: Eine oder mehrere Klassen können als 'ignore' deklariert werden. Pixel, die einer solchen Klasse zugehören, werden bei der Berechnung der Zielfunktion sowie allen Evaluierungsmaßen ignoriert. Für weitere Informationen zur Zielfunktion wird auf den Abschnitt „Das Netzwerk und der Trainingsprozess“ im Kapitel Deep Learning verwiesen. Das Netzwerk ordnet kein Pixel einer als 'ignore' deklarierte Klasse zu. Die Pixel die mit einer solchen 'ignore'-Klasse gelabelt wurden, werden vom Netzwerk wie alle anderen Pixel auch einer nicht als 'ignore' deklarierten Klasse zugeordnet. Im unten folgenden Beispiel bedeutet dies, dass das Netzwerk auch die Pixel der Klasse 'Rand' klassifiziert, aber es wird kein Pixel des Eingabebildes der Klasse 'Rand' zuordnen. Eine Klasse kann über den Parameter *'ignore_class_ids'* im Operator `set_dl_model_param` als 'ignore' deklariert werden.

`DLDataset` Dieses Dictionary dient als Datenbank. Dies bedeutet, dass darin alle für das Netzwerk notwendigen Informationen über den Datensatz gespeichert werden, z.B. den Namen und Pfad zu den Bildern, die Klassen, etc. Für Informationen zum Konzept und den einzelnen Einträgen wird auf das Kapitel Deep Learning / Modell verwiesen. Nur auf die Semantische Segmentierung anwendbare Schlüssel betreffen die Bilder `segmentation_image` (siehe den unteren Eintrag). Über die Schlüssel `segmentation_dir` und `segmentation_file_name` werden die Informationen zur Verfügung gestellt, wie die Bilder heißen und wo sie gespeichert sind.

`segmentation_image` Damit das Netzwerk lernen kann, wie die Angehörigen der verschiedenen Klassen aussehen, muss für jedes Pixel jedes Bildes des Trainings-Datensatzes mitgeteilt werden, zu welcher Klasse er gehört. Um dies zu tun wird für jedes Pixel eines Eingabebildes die Klasse im Bild `segmentation_image` als Pixelwert gespeichert. Diese Annotationen sind die Ground Truth Annotationen.



(1) (2)

Schema des Bildes `segmentation_image`. Zur besseren Darstellung werden Grauwerte für die verschiedenen Nummern verwendet. (1) Eingabebild. (2) Das entsprechende Bild `segmentation_image`, welches die Klassenannotationen zur Verfügung stellt, 0: Hintergrund (weiß), 1: Orange, 2: Zitrone, 3: Apfel und 4: Rand (schwarz) als separate Klasse, damit diese als 'ignore' deklariert werden kann.

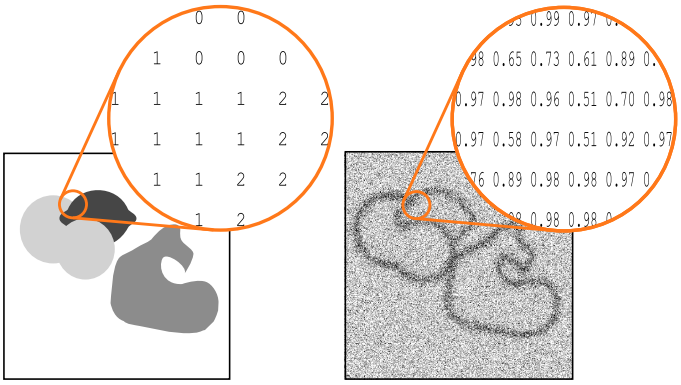
Es sollten genügend Trainingsdaten vorliegen, um diese in die drei Untermengen aufzuteilen, welche für das Training, die Evaluierung und das Testen des Netzwerks verwendet werden. Diese Untermengen sind idealerweise unabhängig und gleich verteilt, siehe den Abschnitt „Daten“ im Kapitel Deep Learning. Zur Aufteilung kann die Prozedur `split_dl_data_set` verwendet werden.

Bilder Unabhängig von der Anwendung stellt das Netzwerk Anforderungen an die Bilder wie z.B. die Bilddimensionen. Die spezifischen Werte hängen vom Netzwerk selbst ab, für die spezifischen Werte der verschiedenen Netzwerke siehe die Dokumentation von `read_dl_model`. Für ein eingelesenes Netzwerk können die Werte mit `get_dl_model_param` abgefragt werden. Um diese Anforderungen zu erfüllen, müssen die Bilder eventuell vorverarbeitet werden. Die Standard-Vorverarbeitung für das ganze Sample und damit auch für das Bild kann mit Hilfe der Prozedur `preprocess_dl_samples` durchgeführt werden. Für eine selbst erstellte Vorverarbeitung liefert diese Prozedur eine Anleitung für die Implementierung.

Netzwerk Ausgabe Als Trainingsausgabe gibt das Modell ein Dictionary `DLTrainResult` mit dem aktuellen Wert der Gesamt-Zielfunktion sowie den Werten aller weiteren im Modell enthaltenen Zielfunktionen.

Als Inferenz- und Evaluierungs-Ausgabe gibt das Netzwerk für jedes Bild ein Dictionary `DLResult` zurück. Im Falle der Semantischen Segmentierung beinhaltet dieses Dictionary für jedes Eingabebild das Handle der folgenden beiden Bildern:

- `segmentation_image`: In diesem Bild hat jedes Pixel den Wert der Klasse, in welche das entsprechende Pixel des Eingabebildes klassifiziert wurde (siehe die untere Darstellung).
- `segmentation_confidence`: In diesem Bild hat jedes Pixel den Konfidenzwert der Klassifikation des entsprechenden Pixels im Eingabebild (siehe die untere Darstellung).



(1) (2)

Schema zu verschiedenen Daten-Bilder. Zur besseren Darstellung werden Grauwerte für die verschiedenen Nummern und Zahlen verwendet. (1) `segmentation_image`: Auch die Pixel von Klassen die als 'ignore' deklariert wurden, werden klassifiziert (siehe auch die obere Darstellung). (2) `segmentation_confidence`.

Modellparameter und Hyperparameter

Nebst den allgemeinen DL Hyperparametern aus dem Kapitel Deep Learning gibt es für die Semantische Segmentierung noch einen weiteren Hyperparameter:

- 'class weights', siehe die unten folgende Erklärung.

Für DL Modelle der Semantischen Segmentierung werden die Modellparameter und Hyperparameter (mit Ausnahme der 'class weights') über den Operator `set_dl_model_param` gesetzt. Für Informationen zu den Modellparametern wird auf die Dokumentation von `get_dl_model_param` verwiesen.

Es gilt zu beachten, dass aufgrund hohen Speicherbedarfs normalerweise für das Training nur kleine Batchgrößen verwendet werden können. Deshalb ist das Training eher langsam und es wird empfohlen, ein größeres Momentum als z.B. bei der Klassifikation zu verwenden. Das HDevelop Beispiel `segment_pill_defects_deep_learning_2_train.hdev` liefert gute initiale Parameterwerte für das Training eines Modells der Semantischen Segmentierung in HALCON.

'class weights' Mit dem Hyperparameter 'class_weights' kann jeder Klasse ein Wert zugeordnet werden, mit welchem die jeweiligen Pixel während des Trainings gewichtet werden. Erhalten die verschiedenen Klassen unterschiedliche Gewichte, so wird das Netzwerk forciert, den Klassen beim Lernen eine unterschiedliche

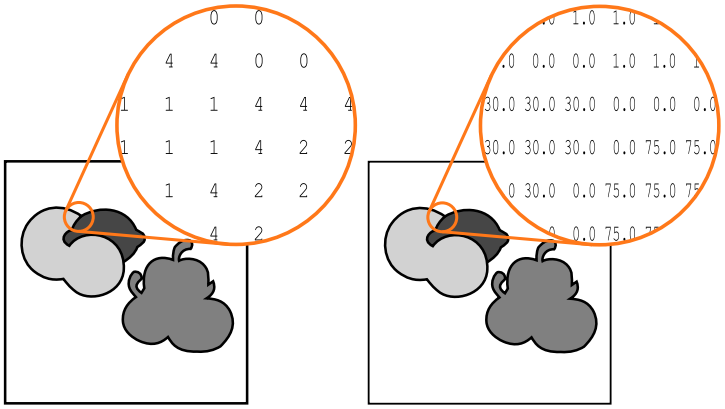
Bedeutung zu geben. Dies ist nützlich, wenn eine Klasse die Bildanteile dominiert, wie z.B. bei der Defekterkennung, wo die Defekte nur einen kleinen Teil des Bildes einnehmen. In solchen Fällen würde ein Netzwerk einen guten Wert für die Zielfunktion erzielen, indem es das gesamte Bild als Hintergrund (und somit als 'nicht defekt') klassifiziert. Den Klassen verschiedene Gewichte zuzuordnen hilft, die Verteilung besser auszubalancieren. Kurz, die Zielfunktion wird darauf fokussiert, insbesondere die Pixel zu lernen, die von Ihnen als wichtig bestimmt wurden.

Das Netzwerk erhält diese Gewichte über das Bild `weight_image`, welches für jedes Trainingsbild erstellt wird. In `weight_image` entspricht jeder Pixelwert dem Gewicht, welches das entsprechende Pixel des Eingabebildes während des Trainings erhält. Ein solches Bild `weight_image` kann durch Verwendung der folgenden beiden Prozeduren erstellt werden:

- `calculate_class_weights_segmentation` bestimmt die verschiedenen Klassen-Gewichte. Die Prozedur verwendet das Konzept der Gewichte nach inverser Klassenhäufigkeit.
- `gen_dl_segmentation_weight_images` verwendet die Klassen-Gewichte und generiert die Bilder `weight_image`.

Dies muss vor dem Training getan werden. Normalerweise werden die Bilder während der Vorverarbeitung generiert und dies ist Teil der Prozedur `preprocess_dl_samples`. Es gilt zu beachten, dass dieser Hyperparameter in den verschiedenen Prozeduren `class_weights` oder `ClassWeights` genannt wird. Eine Darstellung, wie solche Bilder mit unterschiedlichen Gewichten aussehen können, ist in der Abbildung weiter unten gegeben.

Es gilt zu beachten, wird einem bestimmten Part eines Bildes das Gewicht von 0.0 gegeben, tragen diese Pixel nicht zur Zielfunktion bei (siehe „Das Netzwerk und der Trainingsprozess“ für mehr Informationen zur Zielfunktion).



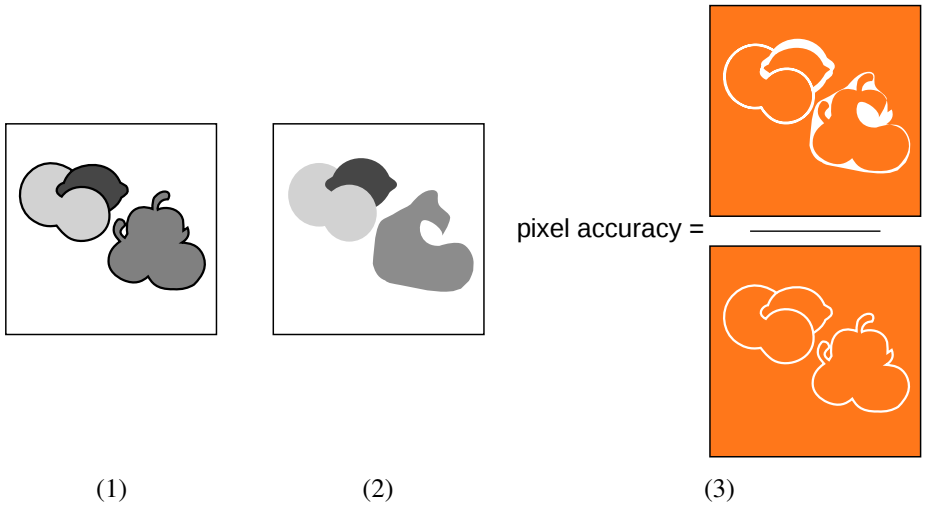
(1) (2)

Schema zu den Bildern `weight_image`. Zur besseren Darstellung werden Grauwerte für die verschiedenen Nummern und Zahlen verwendet. (1) Das Bild `segmentation_image` definiert für jedes Pixel des Bildes seine Klasse, 0: Hintergrund (weiß), 1: Orange, 2: Zitrone, 3: Apfel und 4: Rand (schwarz), welche als 'ignore' deklariert wurde. (2) Das zugehörige Bild `weight_image`, welches die Klassen-Gewichte `class_weights` 0 liefert, Hintergrund: 1.0, Orange: 30.0, Zitrone: 75.0. Pixel von Klassen, die als 'ignore' deklariert wurden, erhalten das Gewicht 0.0. Dies betrifft hier die Klasse Rand.

Evaluierungsmaße für die Daten der Semantischen Segmentierung

Für die Semantische Segmentierung werden in HALCON folgende Evaluierungsmaße unterstützt. Es gilt zu beachten, dass zur Berechnung eines solchen Maßes die zugehörige Ground Truth Annotation notwendig ist. Die verschiedenen Evaluierungsmaße die unten für ein einzelnes Bild erklärt werden (wie z.B. `mean_iou`), können auch für eine beliebige Anzahl Bilder berechnet werden. Dafür kann man sich ein einzelnes, großes Bild vorstellen, gebildet aus dem Ensemble an Einzelbildern. Es gilt zu beachten, dass die Pixel aller Klassen, die als 'ignore' deklariert wurden, für die Berechnung der Maße ignoriert werden.

`pixel_accuracy` Die Pixel Genauigkeit ist das Verhältnis aller korrekt klassifizierten Pixel zur Gesamtzahl Pixel.



(1) (2) (3)

Darstellungsbeispiel zur Pixel Genauigkeit `pixel_accuracy`: (1) Das Bild `segmentation_image` definiert die Ground Truth Klassenzuordnungen aller Pixel (siehe den obere Abschnitt „Daten“). (2) Das Ausgabebild, in welchem auch alle Pixel einer als 'ignore' deklarierten Klasse klassifiziert werden. (3) Die Pixel Genauigkeit ist das Verhältnis der beiden orangen Flächen. Es gilt zu beachten, dass die Pixel jeder Klasse, die als 'ignore' deklariert wurde, ignoriert werden.

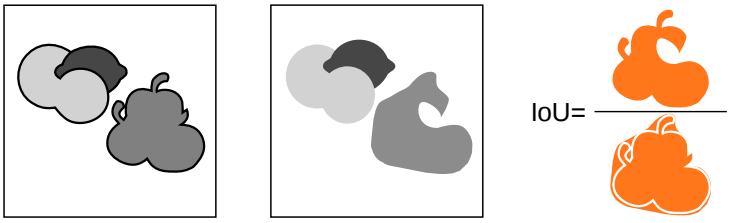
`class_pixel_accuracy` Die klassenweise Pixel Genauigkeit berücksichtigt nur die Pixel einer Klasse. Dieses Maß ist definiert als das Verhältnis aller korrekt dieser Klasse zugeordneten Pixel zur Gesamtzahl Pixel dieser Klasse.

Falls eine Klasse nicht vertreten ist, wird ihr für `class_pixel_accuracy` ein Wert *'-1'* zugewiesen und sie trägt nicht zum Mittelwert `mean_accuracy` bei.

`mean_accuracy` Die durchschnittliche Genauigkeit ist der Mittelwert der klassenweisen Pixel Genauigkeiten `class_pixel_accuracy` über alle vertretenen Klassen.

`class_iou` Die klassenweise Intersection over Union (IoU) gibt für eine Klasse das Verhältnis von korrekt zugeordneten Pixeln zur Anzahl an Pixeln, die entweder tatsächlich zu dieser Klasse gehören oder dieser zugeordnet wurden. Bildlich entspricht dies dem Verhältnis zwischen der Schnittmenge und der Vereinigungsmenge der Flächen, siehe auch die untere Darstellung.

Falls eine Klasse nicht vertreten ist, wird ihr für `class_iou` ein Wert *'-1'* zugewiesen und sie trägt nicht zum Mittelwert `mean_iou` bei.



(1) (2) (3)

Bildliches Beispiel der klassenweisen IoU, `class_iou`, anhand der Klasse Apfel. (1) Das Bild `segmentation_image` definiert die Ground Truth Klassenzuordnungen aller Pixel (siehe den obere Abschnitt „Daten“). (2) Das Ausgabebild, in welchem auch alle Pixel einer als 'ignore' deklarierten Klasse klassifiziert werden. (3) Die IoU ist das Verhältnis zwischen der Schnittmenge und der Vereinigungsmenge der beiden Flächen aus Pixel der Klasse Apfel. Es gilt zu beachten, dass die Pixel jeder Klasse, die als 'ignore' deklariert wurde, ignoriert werden.

`mean_iou` Die durchschnittliche IoU ist der Mittelwert der klassenweisen IoU `class_iou` über alle vertretenen Klassen. Es gilt zu beachten, dass alle vertretenen Klassen den gleichen Einfluss auf dieses Maß haben, unabhängig von ihrer Anzahl an Pixel.

`frequency_weighted_iou` Wie für die durchschnittliche IoU werden erst die klassenweisen IoU berechnet. Der Beitrag einer jeden Klasse zu diesem Maß ist abhängig von der Anzahl Pixel dieser Klasse. Dadurch können Klassen mit einer großen Anzahl an Pixeln dieses Maß dominieren.

pixel_confusion_matrix Das Konzept der Konfusionsmatrix wird im Abschnitt „Das Training beaufsichtigen“ des Kapitels Deep Learning erklärt. Dieses ist auch für die Semantische Segmentierung anwendbar, wo die Instanzen einzelne Pixel sind.

Kapitel 9

Develop

dev_clear_obj (Objects : : :)

Löschen von Bildobjekten aus der HALCON-Datenbank.

dev_clear_obj löscht ikonische Objekte, die nicht mehr benötigt werden, aus der Datenbank.

Achtung

In HDevelop dürfen ikonische Objekte nicht mit clear_obj gelöscht werden. Soll ein Objekt freigegeben werden, muss dev_clear_obj verwendet werden.

Parameter

▷ **Objects** (input_object) ..... object(-array) ⇝ *object*
Zu löschende Objekte.

Ergebnis

dev_clear_obj liefert immer TRUE zurück.

Siehe auch

clear_obj, dev_set_check, reset_obj_db

Modul

Foundation

dev_clear_window (: : :)

Löscht den Inhalt des aktiven Grafikfensters.

dev_clear_window löscht den Inhalt des aktiven Grafikfensters einschließlich der Historie der ausgegebenen Objekte. Die im Kontextmenü, über den Visualisierungsparameterdialog oder die entsprechenden Operatoren (z.B. dev_set_color, dev_set_draw, usw.) vorgenommenen Einstellungen der Ausgabeparameter bleiben dabei erhalten.

Der Operator ist äquivalent zur Löschen-Aktion in der Aktionsleiste des aktiven Grafikfensters oder im Visualisierungsmenü.

Ein Grafikfenster kann mit Hilfe von dev_set_window oder über den Aktivierungsknopf in der Aktionsleiste des Grafikfensters aktiviert werden.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Beispiel

```
read_image (Image, 'fabrik')
regiongrowing (Image, Regions, 3, 3, 6, 100)
Number := |Regions|
dev_update_window ('off')
for i := 1 to Number by 1
  select_obj (Regions, RegionSelected, i)
  dev_clear_window ()
  dev_display (RegionSelected)
  * stop ()
endfor
```

Ergebnis

`dev_clear_window` liefert immer TRUE zurück.

Vorgänger

`dev_set_window`, `dev_open_window`, `dev_display`

Nachfolger

`dev_display`

Siehe auch

`clear_window`

Modul

Foundation

dev_close_inspect_ctrl (: : Variable :)

Schließen von Inspektionsfenstern für eine oder mehrere Steuervariablen.

`dev_close_inspect_ctrl` ist das Gegenstück zum Operator `dev_inspect_ctrl`. Er schließt entsprechend der im Parameter `Variable` angegebenen Variablen ein oder mehrere Inspektionsfenster. Der Parameter `Variable` kann einen Ausdruck (z.B. ein Tupel) mit mehreren Variablennamen enthalten. Zuerst versucht `dev_close_inspect_ctrl` ein Inspektionsfenster zu finden, das genau die angegebenen Variablen enthält. Wird ein solches Fenster gefunden, wird es geschlossen. Ist kein Inspektionsfenster offen, das genau die spezifizierten Variablen enthält, wird für jede Variable das erste Inspektionsfenster, das diese Variable enthält, gesucht, und die entsprechende Spalte aus der Tabelle entfernt. Wird auf diese Art die letzte Spalte einer Inspektionstabelle entfernt, dann wird das Inspektionsfenster geschlossen.

Alternativ zum Operator `dev_close_inspect_ctrl` lassen sich die Inspektionsfenster auch über den `Schließen`-Knopf in der Titelzeile schließen.

Achtung

Dieser Operator funktioniert nur für einzelne schwebende Inspektionsfenster, d.h. Inspektionsfenster, die weder angedockt noch mit Registerkarten versehen sind.

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **Variable** (input_control) . . . . . real(-array) ⇝ *real* / integer / string
Name der Variablen, deren Inspektionsfenster geschlossen werden soll.

Beispiel

```
Var := 1
dev_inspect_ctrl (Var)
Var := [1,2,3,9,5,6,7,8]
Var[3] := 4
stop ()
dev_close_inspect_ctrl (Var)
```

Ergebnis

`dev_close_inspect_ctrl` liefert immer TRUE zurück.

Vorgänger

dev_inspect_ctrl

Modul

Foundation

dev_close_tool (: : ToolId :)

Schließt das angegebene schwebende Tool-Fenster.

dev_close_tool schließt das Tool mit der Id ToolId.

Anstatt der ToolId kann auch der Name verwendet werden, der beim Öffnen mit dev_open_tool angeben werden muss. Allerdings wird dann ein beliebiges Fenster, zu dem der Parameter ToolId passt, verwendet.

Achtung

Dieser Operator funktioniert nur für einzelne schwebende Tools, d.h. Tools, die weder angedockt noch mit Registerkarten versehen sind.

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **ToolId** (input_control) . . . . . dev_tool ⇝ *string*
Tool-Identifikator.

Beispiel

```
dev_open_tool ('matching_assistant', 0, 0, 'default','default', ToolId)
dev_close_window (ToolId)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_window den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_set_tool_geometry, dev_open_tool

Nachfolger

dev_open_tool

Siehe auch

dev_show_tool

Modul

Foundation

dev_close_window (: : :)

Schließen des aktiven schwebenden Grafikfensters.

dev_close_window schließt das aktive schwebende Grafikfenster.

Der Operator ist äquivalent zum Drücken des Schließen-Knopfes in der Titelzeile des aktiven Grafikfensters oder dem Betätigen des entsprechenden Menüeintrages im Visualisierungsmenü.

Ein Grafikfenster kann mit Hilfe von dev_set_window oder über den Aktivierungsknopf in der Aktionsleiste des Grafikfensters aktiviert werden.

Achtung

Dieser Operator funktioniert nur für einzelne schwebende Grafikfenster, d.h. Grafikfenster, die weder angedockt noch mit Registerkarten versehen sind.

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide“ im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Beispiel

```
* close all windows
for i := 1 to 10 by 1
  dev_close_window ()
endfor
read_image (For5, 'for5')
get_image_size (For5, Width, Height)
dev_open_window (0, 0, Width, Height, 'black', WindowHandle)
dev_display (For5)
```

Ergebnis

`dev_close_window` liefert immer TRUE zurück.

Vorgänger

`dev_set_window`, `dev_open_window`

Nachfolger

`dev_open_window`, `dev_get_window`

Siehe auch

`close_window`

Modul

Foundation

dev_disp_text (: : String, CoordSystem, Row, Column, Color, GenParamName, GenParamValue :)

Ausgabe von Text in einem Grafikfenster.

`dev_disp_text` gibt Text im aktiven Grafikfenster an der Position (`Row`,`Column`) aus.

Wird nur eine Position übergeben, wird für jedes Element in `String` eine Textzeile ausgegeben. Außerdem wird auch *'\n'* als Zeilenumbruch interpretiert.

Werden mehrere Positionen übergeben, ist nur ein einzelner String oder ein String für jede Position zulässig. In diesem Fall kann ein Zeilenumbruch nur über *'\n'* erreicht werden.

Zeilenumbrüche (*'\n'*) am Ende von `String` werden ignoriert.

Die Position des Texts kann in Fensterkoordinaten (`CoordSystem` = *'window'*) oder in Bildkoordinaten (`CoordSystem` = *'image'*), was für gezoomte Bilder nützlich ist, angegeben werden.

Zusätzlich zu (`Row`,`Column`) Koordinaten können auch vordefinierte Werte an `Row` und `Column` übergeben werden um Text an festen Positionen im Fenster anzuzeigen (nur für `CoordSystem` = *'window'*).:

'top', *'left'*	*'top'*, *'center'*	*'top'*, *'right'*
'center', *'left'*	*'center'*, *'center'*	*'center'*, *'right'*
'bottom', *'left'*	*'bottom'*, *'center'*	*'bottom'*, *'right'*

Der Parameter `Color` akzeptiert auch Farben-Tupel. In diesem Fall werden die Farben zyklisch für jede neue Textposition (oder für jede Textzeile wenn nur eine Position angegeben wurde) verwendet.

Generische Parameter

`disp_text` kann den `String` innerhalb einer Box anzeigen. Dieses Verhalten und das Aussehen der Box wird mit den generischen Parametern in `GenParamName` und `GenParamValue` beeinflusst.

'box' Ist *'box'* auf *'true'* gesetzt, wird der Text innerhalb einer Box angezeigt. Die folgenden Parameter beeinflussen das Aussehen dieser Box und des optionalen Schattens der Box.

Mögliche Werte: *'true'* und *'false'*

Vorgegebener Wert: *'true'*

'box_color' Setzt die Farbe der Box.

Mögliche Werte: Ein String mit dem Namen der Farbe (z.B. *'white'*, *'red'* oder *'#aa00bba0'*)

Vorgegebener Wert: *'#fce9d4'* (ein helles Orange)

'shadow' Wird *'shadow'* auf *'true'* gesetzt, wird zusätzlich ein Schatten zu der Box angezeigt.

Mögliche Werte: *'true'* und *'false'*

Vorgegebener Wert: *'true'* wenn *'box_color'* auf eine Farbe ohne Alphawert gesetzt wurde, ansonsten *'false'*

'shadow_color' Setzt die Farbe des Schattens.

Mögliche Werte: Ein String mit dem Namen der Farbe (z.B. *'black'*, *'red'* oder *'#aa00bba0'*)

Vorgegebener Wert: *'#f28d26'* (ein dunkleres Orange) wenn *'box_color'* nicht gesetzt wurde, ansonsten *'white'*

'border_radius' Setzt die Abrundung der Ecken der Box. Für scharfe Ecken muss dieser Parameter auf *0*, für abgerundete Ecken auf höhere Werte gesetzt werden.

Mögliche Werte: positive reelle Zahl oder 0

Vorgegebener Wert: 2

'box_padding' Bestimmt, wie stark die Box über den Text hinaus geht.

Mögliche Werte: positive reelle Zahl

Vorgegebener Wert: 0

'shadow_sigma' Bestimmt, wie stark der Schatten hinter der Box weichgezeichnet wird. Für scharfe Schatten muss dieser Parameter auf 0 gesetzt werden.

Mögliche Werte: positive reelle Zahl oder 0

Vorgegebener Wert: 1.5

'shadow_dx' and 'shadow_dy' Der Offset des Schattens in Spalten- (*'shadow_dx'*) und Zeilenrichtung (*'shadow_dy'*) in Pixeln.

Mögliche Werte: reelle Zahlen

Vorgegebener Wert: 2

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „`HDevelop User's Guide`" im Kapitel `Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows`.

Parameter

▷ **`String`** (input_control) . . . . . string(-array) ⇝ *string*
Ein Tupel von Strings mit den anzuzeigenden Texten. Jeder String wird in einer Zeile ausgegeben.
Defaultwert: 'hello'

▷ **`CoordSystem`** (input_control) . . . . . string ⇝ *string*
Koordinatensystem für die Textposition. 'window' steht für das Fensterkoordinatensystem, 'image' für das Bildkoordinatensystem.
Defaultwert: 'window'
Werteliste: `CoordSystem` ∈ {'window', 'image'}

▷ **`Row`** (input_control) . . . . . point.y(-array) ⇝ *integer* / real / string
Vertikale Textausrichtung oder Zeilenkoordinate der Textposition.
Defaultwert: 12
Werteliste: `Row` ∈ {12, 'bottom', 'center', 'top'}

▷ **`Column`** (input_control) . . . . . point.x(-array) ⇝ *integer* / real / string
Horizontale Textausrichtung oder Spaltenkoordinate der Textposition.
Defaultwert: 12
Werteliste: `Column` ∈ {12, 'center', 'left', 'right'}

▷ **`Color`** (input_control) . . . . . string(-array) ⇝ *string*
Ein Tupel von Strings für die Farben der Texte.
Defaultwert: 'black'
Werteliste: `Color` ∈ {'black', 'blue', 'yellow', 'red', 'green', 'cyan', 'magenta', 'forest green', 'lime green', 'coral', 'slate blue'}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'box', 'shadow', 'box_color', 'shadow_color', 'border_radius', 'box_padding', 'shadow_sigma', 'shadow_dx', 'shadow_dy'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Werte der generischen Parameter.
Defaultwert: []
Werteliste: GenParamValue ∈ {'true', 'false', 'white', 'red', 'forest green', 'black', 'blue', 5.0}

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
dev_disp_text ('Display some text in a box', 'window', 12, 12, \
               'black', [], [])
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_disp_text den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_open_window, set_font, get_string_extents

Alternativen

disp_text, write_string

Siehe auch

get_string_extents, set_font

Modul

Foundation

dev_display (Object : : :)

Ausgabe von Bildobjekten in einem Grafikfenster.

dev_display gibt ein Bildobjekt (Bild, Region oder XLD) im aktiven Grafikfenster aus. Dies ist äquivalent zu einem Doppelklick auf das entsprechende Icon im Variablenfenster.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **Object** (input_object) . . . . . object(-array) ⇝ *object*
Auszugebende Bildobjekte.

Beispiel

```
read_image (Image, 'fabrik')
regiongrowing (Image, Regions, 3, 3, 6, 100)
dev_clear_window ()
dev_display (Image)
dev_set_colored (12)
dev_set_draw ('margin')
dev_display (Regions)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_display den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

disp_obj, disp_image, disp_region, disp_xld

Siehe auch

dev_set_color, dev_set_colored, dev_set_draw, dev_set_line_width

Modul

Foundation

dev_error_var (: : ErrorVar, Mode :)

Fehlervariable festlegen.

dev_error_var legt eine Fehlervariable fest, die den Rückgabewert des letzten Operatoraufrufs enthält. ErrorVar enthält 2 (H_MSG_TRUE), wenn kein Fehler aufgetreten ist. Der Parameter Mode legt fest, ob die Fehlervariable benutzt werden soll (*1*) oder nicht (*0*). Ist eine Fehlervariable aktiv, wird sie nach jeder Operatorausführung aktualisiert. Aus diesem Grunde ist ein Wert nur bis zum nächsten Operatoraufruf gültig. Der Wert kann gespeichert werden, indem er einer anderen Variablen zugewiesen wird (siehe Beispiel) oder durch Aufruf von dev_error_var(ErrorVar,0).

Parameter

▷ **ErrorVar** (input_control) . . . . . integer ⇝ *integer*
Name der Variablen, die den Status enthalten soll.
Defaultwert: 'ErrorVar'

▷ **Mode** (input_control) . . . . . integer ⇝ *integer*
Ein- oder Ausschalten der Fehlervariablen.
Defaultwert: 1
Werteliste: Mode ∈ {0, 1}

Beispiel

```
dev_close_window ()
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
dev_error_var (Error, 1)
dev_set_check ('~give_error')
FileName := 'wrong_name'
read_image (Image, FileName)
ReadError := Error
if (ReadError != H_MSG_TRUE)
  write_string (WindowHandle, 'wrong file name: '+FileName)
endif
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_error_var den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_set_check

Nachfolger

dev_set_check, if, elseif, else, assign

Siehe auch

dev_set_check, set_check

Modul

Foundation

dev_get_exception_data (: : Exception, Name : Value)

Ermöglicht den Zugriff auf definierte Elemente eines Exception-Tupels.

Der Operator dev_get_exception_data erlaubt den Zugriff auf definierte Elemente eines mit catch gefangenen Exception-Tupels. Bis auf den Fehlercode, der immer an der ersten Stelle des Exception-Tupels steht, müssen alle anderen Daten des Exception-Tupels mittels dev_get_exception_data abgefragt werden, da sich die Reihenfolge und der Umfang der bereitgestellten Informationen unter Umständen in zukünftigen Versionen ändern und sich beim exportierten Code von Sprache zu Sprache unterscheiden kann.

In HDevelop und HDevEngine enthält das Exception-Tupel, das an den Parameter Exception übergeben wird, bei einem Operatorfehler die nachfolgend aufgelisteten Daten. Dabei ist der Slot-Name an den Parameter Name zu übergeben, die gewünschte Information wird in Value zurückgeliefert. Es können auch mehrere Slots gleichzeitig abgefragt werden. In diesem Fall ist ein Tupel mit den Slotnamen zu übergeben, in Value wird ein entsprechendes Tupel mit den angeforderten Daten zurückgeliefert. Ist eine gewünschte Information nicht verfügbar, wird als Ergebnis für diesen Wert ein leerer String (”) zurückgeliefert.

Der Operator dev_get_exception_data ist nur für die Verwendung in HDevelop und HDevEngine gedacht. Manche Informationen sind nicht verfügbar, wenn der Operator exportiert wird.

'error_code': HALCON- oder benutzerdefinierter Fehlercode. Die HALCON-Fehlercodes (Codes < 10000) werden im Anhang des „Extension Package Programmer's Manual“ aufgelistet.

'add_error_code': Zusätzlicher HDevelop-spezifischer Fehlercode, der darüber informiert, ob der Fehler innerhalb des HALCON-Operators generiert wurde (Fehlercode = 21000), oder ob er bereits bei der Übergabe der Parameter, d.h. bei der Auswertung der Parameterausdrücke aufgetreten ist. Im zweiten Fall wird mit diesem Fehlercode die Art des Fehlers genauer spezifiziert. Die relevanten HDevelop-Fehlercodes sind unten aufgelistet.

'error_message': HALCON-Fehlertext.

'add_error_message': Fehlertext, der HDevelop-spezifischen Fehler beschreibt.

'program_line': Zeilennummer, in der der Fehler aufgetreten ist.

'operator': Name des Operators (tritt der Fehler in einer geschützten Prozedur auf, wird '--protected--' zurückgeliefert).

'call_stack_depth': Aufruftiefe ('main' hat die Tiefe 1).

'procedure': Name der Prozedur.

'user_data': Benutzerdefiniertes Exception-Tupel - das sind die Daten, die beim Weitergeben einer Operatorfehler-Exception an das Exception-Tupel hinten angehängt wurden, bzw. die Daten, die bei einer benutzerdefinierten Exception hinter dem Fehlercode ($\geq$ 30000) folgen. Enthält das Exception-Tupel keine benutzerdefinierten Daten, wird ein leeres Tupel zurückgeliefert. Achtung: Diese Daten können nicht zusammen mit anderen Informationen in einem dev_get_exception_data-Aufruf abgefragt werden.

Parameter

▷ **Exception** (input_control) . . . . . exception-array ⇝ *integer* / string
Tupel mit den Fehlerdaten oder eigenen Fehlercodes.

▷ **Name** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der gewünschten Fehlerdaten.
Defaultwert: 'error_code'
Werteliste: Name ∈ {'error_code', 'add_error_code', 'error_message', 'add_error_message', 'operator', 'procedure', 'program_line', 'call_stack_depth', 'user_data'}

▷ **Value** (output_control) . . . . . attribute.value-array ⇝ *integer* / string
Angeforderte Fehlerdaten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_get_exception_data (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

catch

Siehe auch

try, catch, endtry, throw, dev_set_check

Modul

Foundation

dev_get_preferences (: : PreferenceNames : PreferenceValues)

Abfragen von HDevelop-Einstellungen innerhalb eines Programms.

dev_get_preferences ermöglicht die Abfrage ausgewählter HDevelop-Einstellungen aus einem Programm heraus. Folgende Einstellungen werden unterstützt:

'graphics_window_context_menu': Fragt ab, ob ein Rechtsklick im Grafikfenster das Kontextmenü öffnet oder nicht. Standardmäßig ist das Kontextmenü aktiviert.

'graphics_window_mouse_wheel': Fragt ab, ob das Mausrad im Grafikfenster zum Zoomen verwendet wird oder nicht. Standardmäßig ist das Mausrad aktiviert.

'graphics_window_tool_tip': Fragt ab, ob im Grafikfenster beim Drücken der Ctrl-Taste ein Tooltipp mit der Mausposition und dem Grauwert unter der Maus erscheinen soll oder nicht. Standardmäßig ist der Tooltipp aktiviert.

'suppress_handled_exceptions_dlg': Fragt ab, ob bei einer Exception, die innerhalb eines try-catch Blocks geworfen wird, der Fehler-Dialog unterdrückt werden soll oder nicht. Diese Option ist persistent in der HDevelop.ini Datei gespeichert und kann auch über den Einstellungen Dialog / Allgemeine Optionen / Erfahrene Anwender konfiguriert werden.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

- ▷ **PreferenceNames** (input_control) . . . . . attribute.name-array ⇝ *string*
  Auswahl der abzufragenden Präferenzen.
  Defaultwert: 'graphics_window_context_menu'
  Werteliste: PreferenceNames ∈ {'graphics_window_context_menu', 'graphics_window_mouse_wheel', 'graphics_window_tool_tip', 'suppress_handled_exception_dlg'}
- ▷ **PreferenceValues** (output_control) . . . . . attribute.value-array ⇝ *string*
  Werte der ausgewählten Präferenzen.

Siehe auch

dev_set_preferences

Modul

Foundation

dev_get_system (: : SystemQueries : SystemInformations)

Informationen über die aktuellen HDevelop-Systemparameter.

dev_get_system liefert Informationen über die aktuellen HDevelop Systemparameter zurück.

Bislang können folgende Systemparameter abgefragt werden:

Nur lesbare Parameter

'engine_environment': Liefert 'HDevelop' wenn er aus HDevelop abgefragt wird, ansonsten 'HDevEngine'.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

- ▷ **SystemQueries** (input_control) . . . . . attribute.name-array ⇝ *string*
  Auswahl der abzufragenden Systemparameter.
  Defaultwert: 'engine_environment'
  Werteliste: SystemQueries ∈ {'engine_environment'}
- ▷ **SystemInformations** (output_control) . . . . . attribute.value-array ⇝ *string*
  Werte der ausgewählten Systemparameter.

Modul

Foundation

dev_get_window (: : : WindowHandle)

Abfragen des Fensterhandles des aktiven Grafikfensters.

dev_get_window liefert den Fensterhandle das aktiven Grafikfenster zurück. Ist kein Grafikfenster offen, ist der Rückgabewert *-1*.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **WindowHandle** (output_control) ..... window ⇝ *handle*
Fenster-Handle.

Beispiel

```
read_image (Image,'mreut')
threshold (Image, Region, 100, 200)
dev_open_window (1, 1, 200, 200, 'black', WindowID1)
dev_open_window (1, 220, 200, 200, 'black', WindowID2)
dev_get_window (CurrentWindowID)
dev_set_window (WindowID1)
dev_set_color ('blue')
dev_display (Image)
dev_display (Region)
dev_set_window(CurrentWindowID)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_get_window den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_close_window

Nachfolger

dev_set_window

Modul

Foundation

dev_inspect_ctrl (: : Variable :)

Öffnet ein Fenster zur Inspektion einer oder mehrerer Steuervariablen.

dev_inspect_ctrl öffnet ein Inspektionsfenster, in dem die Werte einer oder mehrerer Steuervariablen angezeigt werden. Dies entspricht dem Punkt Ansehen im Kontextmenü einer Steuervariablen im Variablenfenster.

Das normale Inspektionsfenster zeigt die Werte und optional noch zusätzliche Statistiken in einer Tabelle an. Für einige semantische Typen, wie z.B. Matrizen, wird automatisch eine spezielle Repräsentation verwendet. Für mehr Informationen hierzu siehe das Kapitel "Inspecting Variables"im HDevelop User's Guide.

Durch die Übergabe eines Tupels von Variablen in Variable kann ein Inspektionsfenster für mehrere Variablen geöffnet werden. Es ist weiter möglich, den Inspektionsmodus explizit zu bestimmen, indem man als erstes Element eine entsprechende Zeichenkette als Literal einfügt:

'tuple': Zeigt die Werte im normalen Inspektionsfenster an, selbst wenn für den semantischen Type der Variable eine spezielle Repräsentation existiert.

'plot': Stellt die Werte numerischer Variablen grafisch gegen den Tupelindex dar.

'plot_xy': Stellt die Werte numerischer Variablen grafisch als X-Y Paare dar. Hierzu übergibt man eine gerade Anzahl an Variablen, die jeweils paarweise dieselbe Länge haben müssen.

Normalerweise wird der Inhalt des Dialogs aktualisiert sobald sich die Werte der Variablen ändern. Dieses Verhalten kann jedoch durch den Operator `dev_update_var` oder die Einstellung `Variablen aktualisieren` beeinflusst werden.

Der Dialog kann durch Drücken des `Schließen`-Knopfes oder den Aufruf von `dev_close_inspect_ctrl` geschlossen werden.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **`Variable`** (input_control) ..... integer(-array) ⇝ *integer* / real / string
Name der Variablen, deren Inhalt untersucht werden soll.

Beispiel

```
read_image (Image, 'fabrik')
regiongrowing (Image, Regions, 3, 3, 6, 100)
area_center (Regions, Area, Row, Column)
dev_inspect_ctrl (Area)
dev_inspect_ctrl (['plot_xy', Row, Column])
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dev_inspect_ctrl` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Siehe auch

`dev_update_var`

Modul

Foundation

`dev_open_dialog` (: : DialogName :)

Öffnen eines modalen HDevelop-Dialogs.

`dev_open_dialog` öffnet einen modalen HDevelop-Dialog vom Typ `DialogName`. `DialogName` gibt an welcher Dialog geöffnet werden soll.

Folgende Dialoge werden unterstützt:

`DialogName`	**Dialog**
'about_dialog'	Über Dialog
'duplicate_procedure_dialog'	Prozedur duplizieren Dialog
'export_dialog'	Exportieren Dialog
'open_graphics_windows_dialog'	Grafikfenster öffnen Dialog
'insert_visualization_code_dialog'	Visualisierungscode einfügen Dialog
'read_image_dialog'	Read Image dialog
'print_dialog'	Drücken Dialog
'properties_dialog'	Eigenschaften Dialog
'save_window_dialog'	Grafikfenster als Bild speichern Dialog

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **DialogName** (input_control) . . . . . string ⇝ *string*
Name des zu öffnenden Dialogs.

Beispiel

```
dev_open_dialog ('print_dialog')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_window den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Modul

Foundation

dev_open_file_dialog (: : Filter, Mode, Path : Selection)

Öffnet den Dateiauswahldialog.

dev_open_file_dialog öffnet den Dateiauswahldialog zum Lesen von einer Datei (Mode = *'read'*), zum Lesen von einer oder mehrerer Dateien (Mode = *'read_multi'*), zum Schreiben (Mode = *'write'*) einer Datei oder zur Auswahl eines Verzeichnisses (Mode = *'dir'*). Wird Mode auf *'default'* gesetzt und für den Filter ein HALCON Operator eingesetzt, dann wird die Auswahldialog passend zum Operator verwendet. .

Der Filter kann verwendet werden um die Dateitypen zu selektieren, welche zum Lesen oder Schreiben ausgewählt werden sollten. Filter kann auf einen HALCON Operator gesetzt werden, zum Beispiel read_image, welcher die ausgwählte Datei dann zum Lesen oder Schreiben verwenden sollte. Wird ein HALCON Operator verwendet, dann wird der Filter passend für diesen Operator gesetzt. Um Bilder vom Typ JPG und TIFF auszuwählen, könnnte der Filter so aussehen *'Bilder (*.jpg *.tiff *.tif)'* oder so *'JPEG Files (*.jpg);;TIFF Files (*.tiff *.tif)'*. Im ersten Fall werden alle unterstützten Bilder gleichzeitig aufgelistet, im zweiten Fall ist es möglich zwischen JPEG Files und TIFF Files zu wählen. Beim Filter wird nicht zwischen Groß- und Kleinschreibung unterschieden, somit liefert *'JPEG Files (*.jpg)'* und *'JPEG Files (*.JPG)'* das gleiche Ergebnis. Der Filter wird mit dem Eintrag *'All Files (*)'* ergänzt. Mehrer Filter müssen mit *';;'* kombiniert werden.

Beispiele:

Der Filter *'Bilder (*.jpg *.tiff *.tif)'* listet alle Dateien mit der Endung jpg,tiff und tif.

Der Filter *'JPEG Files (*.jpg)'* listet alle Dateien mit der Endung jpg

Kombination von 3 Filtern: *'Bilder (*.jpg *.tiff *.tif);;JPEG Files (*.jpg);; TIFF Files (*.tiff *.tif)'*

Der Parameter Path kann verwendet werden um das Arbeitsverzeichnis zu wählen. Falls Path auf *'default'* gesetzt wird, verwendet der Dialog das für diesen Mode zuletzt verwendete Arbeitsverzeichnis.

Nach der erfolgreichen Ausführung enthält Selection die selektierte Datei, bzw. das Verzeichnis, oder die selektierten Dateien.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **Filter** (input_control) . . . . . string ⇝ *string*
Auswahl der Dateitypen.

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Modus der Dateiauswahldialogs.
Defaultwert: 'default'
Werteliste: Mode ∈ {'default', 'read', 'read_multi', 'write', 'dir'}

▷ **Path** (input_control) . . . . . string ⇝ *string*
Arbeitsverzeichnis.
Defaultwert: 'default'

▷ **Selection** (output_control) . . . . . string(-array) ⇝ *string*
Selektierte Dateinamen oder Verzeichnis.
Defaultwert: 'default'

Beispiel

```
dev_open_file_dialog ('read_image', 'default','c:/',Selection)
read_image(ToolId, Selection)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dev_update_pc` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`dev_open_dialog`

Siehe auch

`dev_open_dialog`

Modul

Foundation

dev_open_tool (: : ToolName, Row, Column, Width, Height, GenParamName, GenParamValue : ToolId)

Öffnen eines HDevelop Tools, eines nicht-modalen Dialogs oder eines Assistenten.

`dev_open_tool` öffnet ein HDevelop Tool, einen nicht-modalen Dialog oder einen Assistenten. Der Parameter `ToolName` enthält den Namen des zu öffnenden Tools. Als Rückgabewert liefert der Operator im Parameter `ToolId` die Tool Id. Diese kann dann in Operatoren, wie `dev_set_tool_geometry`, `dev_show_tool` und `dev_close_tool` verwendet werden um ein bestimmtes HDevelop Tool anzusprechen. Das geöffnete Fenster wird automatisch aktiv.

Wenn der Parameter `ToolName` auf *'halcon_news'* gesetzt wird, so wird `ToolId` nur ein leeres Tupel zurückgegeben, da diese Aktion nur zum Öffnen der News-Seite dient.

Die Parameter `GenParamName` und `GenParamValue` können verwendet werden um eine bestimmte Seite des Tools anzuzeigen. Die Verwendung dieser Parameter ist optional. Werden sie auf *[]* gesetzt, so wird das Tool mit den Defaulteinstellungen geöffnet.

Folgende Tools werden unterstützt:

ToolName	Hauptfenster
'graphics_window'	Grafikfenster
'operator_window'	Operatorfenster
'program_window'	Programmfenster
'variable_window'	Variablenfenster

ToolName	Assistenten
'calibration_assistant'	Calibration Assistent
'image_acquisition_assistant'	Image Aquisition Assistent
'matching_assistant'	Matching Assistent
'measure1d_assistant'	Measure Assistent
'ocr_assistant'	OCR Assistent

ToolName	Tools
'call_stack'	Aufrufliste
'canvas'	Canvas-Fenster
'feature_histogram'	Merkmalshistogramm
'feature_inspection'	Merkmalsinspektion
'gray_histogram'	Grauwerthistogramm
'line_profile'	Linienprofile
'zoom_window'	Zoomfenster
'ocr_training_file_browser'	OCR Trainingsdateien Browser

ToolName	Dialoge
'browse_examples_dialog'	Beispiele durchsuchen
'create_procedure_dialog'	Prozedur erstellen
'edit_procedure_interface_dialog'	Prozedurschnittstellen bearbeiten
'find_replace_dialog'	Suchen Ersetzen
'output_console'	Ausgabekonsole
'quick_navigation'	Schnellzugriff
'breakpoints_dialog'	Schnellzugriff / Haltepunkte
'bookmarks_dialog'	Schnellzugriff / Lesezeichen
'invalid_lines_dialog'	Schnellzugriff / Ungültige Zeilen
'visualization_parameters_dialog'	Visualisierungs-Parameter
'help'	Hilfefenster
'preferences'	Einstellungen
'halcon_news'	HALCON News Webseite

Weitere Einstellungen können mit GenParamName und GenParamValue durchgeführt werden und sind für die folgenden Tools möglich:

'calibration_assistant' **- Calibration Assistent:** *'page'***:** Legt die anzuzeigende Seite fest.

Value	Page
'setup'	Aufbau
'calibration'	Kalibrierung
'results'	Ergebnisse
'code_generation'	Codegenerierung

Zum Beispiel: dev_open_tool ('calibration_assistant',..., 'page', 'results',...) öffnet den Calibration Assistenten mit der Tab-Karte Ergebnisse.

'image_acquisition_assistant' **- Image Aquisition Assistent:** *'page'***:** Legt die anzuzeigende Seite fest.

Value	Page
'source'	Quelle
'connection'	Verbindung
'parameters'	Parameter
'code_generation'	Codegenerierung

Zum Beispiel: dev_open_tool ('image_acquisition',..., 'page', 'source',...) öffnet den Image Acquisition Assistenten mit der Tab-Karte Quelle.

'matching_assistant' **- Matching Assistent:** *'page'***:** Legt die anzuzeigende Seite fest.

Value	Page
'model_creation'	Modellerzeugung
'model_parameter'	Modellparametrierung
'model_use'	Modellanwendung
'inspect'	Inspektion
'code_generation'	Codegenerierung

Zum Beispiel: dev_open_tool ('matching_assistant',..., 'page', 'inspect',...) öffnet den Matching Assistenten mit der Tab-Karte Inspektion.

'measure1d_assistant' **- Measure Assistent:** *'page'***:** Legt die anzuzeigende Seite fest.

Value	Page
'input'	Eingabe
'edges'	Kanten
'fuzzy'	Fuzzy
'results'	Ergebnisse
'code_generation'	Codegenerierung

Zum Beispiel: dev_open_tool ('measure1d_assistant',..., 'page', 'edges',...) öffnet den Measure Assistenten mit der Tab-Karte Kanten.

'quick_navigation' **- Schnellzugriff** *'page'***:** Legt die anzuzeigende Seite fest.

Value	Page
'invalid_lines'	Ungültige Zeilen
'find_results'	Suchergebnisse
'breakpoints'	Haltepunkte
'bookmarks'	Lesezeichen

Zum Beispiel: dev_open_tool ('quick_navigation',..., 'page', 'breakpoints',...) öffnet die Programmzeilenansicht mit der Tab-Karte Haltepunkte.

'help' **- Hilfefenster:** Mögliche Eingaben für `GenParamName` sind *'page'*, *'operator'*, *'manual'*.

*'page'***:** Legt die anzuzeigende Seite fest.

Value	Page
'contents'	Inhalt
'operator'	Operator
'search'	Suche
'index'	Index
'bookmarks'	Lesezeichen

Zum Beispiel: dev_open_tool ('help',...,'page', 'search',...) öffnet die Hilfe und zeigt die Tab-Karte Suche an.

*'operator'***:** Legt fest welche Operatorbeschreibung angezeigt werden soll. Zum Beispiel: dev_open_tool ('help',..., '['page','operator'],['search','read_image'] ...) öffnet die Hilfe, wählt die Tab-Karte Suche und zeigt die Beschreibung des Operators read_image.

*'manual'***:** Legt das anzuzeigende Manual fest.

'preferences' **- Einstellungen Dialog:** *'page'***:** Die Einstellungen für 'page' sind eine Kombination aus zwei Werten. Zunächst wird die Gruppe gesetzt und danach die Tab-Karte in dieser Gruppe. Die beiden Werte müssen durch '/' von einander getrennt werden. Siehe nachfolgendes Beispiel.
Zum Beispiel: dev_open_tool ('preferences',..., 'page','user_interface/language', ...) öffnet die Tab-Karte *'language'* der Gruppe *'user_interface'*.
Mögliche *Grouppen* sind:

Value	Group
'user_interface'	Benutzerschnittstelle
'procedures'	Prozeduren
'general_options'	Allgemeine Optionen
'visualization_settings'	Visualisierung
'runtime_settings'	Laufzeiteinstellungen
'telemetry_settings'	Telemetrieeinstellungen

*'user_interface'***:** Legt die anzuzeigende Gruppe fest.

Value	Page
'user_interface/program_window'	Programmfenster
'user_interface/fonts'	Schriftarten
'user_interface/language'	Sprache
'user_interface/themes'	Themes

'procedure': Legt die anzuzeigende Gruppe fest.

Value	Page
'procedures/directories'	Verzeichnisse
'procedures/external_procedures'	Externe Prozeduren
'procedures/manage_procedure_libraries'	Prozedurbibliothek verwalten
'procedures/manage_passwords'	Passwörter verwalten
'procedures/procedure_use'	Prozedurverwendung
'procedures/unresolved_procedure_calls'	Nicht aufgelöste Prozeduraufrufe

'general_options': Legt die anzuzeigende Gruppe fest.

Page	Value
'general_options/general_options'	Allgemeine Optionen
'general_options/experienced_user'	Erfahrener Anwender

'visualization_settings': Legt die anzuzeigende Gruppe fest.

Value	Page
'visualization_settings/pen'	Stift
'visualization_settings/lut'	LUT
'visualization_settings/paint'	Darstellung

'runtime_settings': Legt die anzuzeigende Gruppe fest.

Value	Page
'runtime_settings/runtime_settings'	Laufzeiteinstellung
'runtime_settings/override_operator_behavior'	Operatorverhalten aufheben

Mit den Parametern `Row` und `Column` kann das HDevelop Tool an einer bestimmten Position geöffnet werden. Beachten Sie, dass die unter `Bearbeiten` ▷ `Einstellungen` ▷ `Allgemeine Optionen` ▷ `Allgemeine Optionen` ▷ `Window open offset` angegebenen Versatzwerte zum Zeileindex bzw. zum Spaltenindex addiert werden. Weitere Informationen finden Sie im Kapitel „Menu `Edit`“ im „`HDevelop User's Guide`“. Zum Verwenden des Standardverhalten, also das Tool an der zuletzt geöffneten Position zu positionieren, sollte *'default'* übergeben werden.

Die Parameter `Width` und `Height` können verwendet werden um das Tool mit einer bestimmten Größe zu öffnen. Um die zuletzt verwendete Größe zu verwenden, sollte der Wert *'default'* übergeben werden.

Mithilfe des `Schließen`-Knopfes im Fensterrahmen, oder durch Aufruf von `dev_close_tool` kann das Tool geschlossen werden.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

- ▷ **`ToolName`** (input_control) . . . . . string ⇝ *string*
  Toolname.
- ▷ **`Row`** (input_control) . . . . . rectangle.origin.y ⇝ *integer* / string
  Zeile der oberen linken Ecke.
  Defaultwert: 'default'
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **`Column`** (input_control) . . . . . rectangle.origin.x ⇝ *integer* / string
  Spalte der oberen linken Ecke.
  Defaultwert: 'default'
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

▷ **Width** (input_control) . . . . . rectangle.extent.x ⇝ *integer* / string
Breite des Tools.
Defaultwert: 'default'
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Width > 0

▷ **Height** (input_control) . . . . . rectangle.extent.y ⇝ *integer* / string
Höhe des Tools.
Defaultwert: 'default'
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Height > 0

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string*
Werte der generischen Parameter.
Defaultwert: []

▷ **ToolId** (output_control) . . . . . dev_tool ⇝ *string*
Tool-Identifikator.

Beispiel

```
dev_open_tool ('zoom_window',0, 0, Width, Height, [], [], ToolId)
dev_close_tool (ToolId)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_window den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

dev_set_tool_geometry, dev_show_tool, dev_close_tool

Modul

Foundation

dev_open_window (: : Row, Column, Width, Height,
Background : WindowHandle)

Öffnen eines neuen Grafikfensters.

dev_open_window öffnet ein neues Grafikfenster, in dem ikonische Objekte wie Bilder, Regionen und Linien sowie Text und geometrische Objekte ausgegeben werden können. Nach dem Öffnen ist automatisch das neue Grafikfenster aktiv, d.h. dass alle HDevelop-Ausgabeoperatoren sich auf dieses Grafikfenster beziehen. Das aktive Grafikfenster erkennt man an der leuchtenden Lampe im Aktivierungsknopf.

Jedes Grafikfenster wird durch sein WindowHandle identifiziert. Diese logische Nummer wird in der Titelleiste des Grafikfensters angezeigt und wird als Eingabeparameter für einige Operatoren wie disp_image, disp_region, draw_circle, get_mbutton, write_string und set_rgb benötigt.

Mithilfe des Schließen-Knopfes im Fensterrahmen, über das Visualisierungsmenü oder durch Aufruf von dev_close_window kann das Grafikfenster geschlossen werden.

Grafikfenster-Position

Das Grafikfenster wird entweder als schwebendes Fenster oder innerhalb des Canvas-Fensters geöffnet. Sobald das Canvas-Fenster geöffnet ist, werden alle neuen Grafikfenster automatisch im Canvas angezeigt. Siehe auch das Kapitel Organizing Graphics Windows on the Canvas im „HDevelop User's Guide".

Bei schwebenden Grafikfenstern kann mit den Parametern Row und Column die Position des Fensters eingestellt werden. Sie haben aber noch weitere Möglichkeiten, diese Position an Ihre Bedürfnisse anzupassen. Eine Möglichkeit ist, den Bezugspunkt der Fensterposition zu ändern, der standardmäßig die linke obere Desktopecke ist. Sie können diesen Bezugspunkt mit der Einstellung Origin of coordinates ändern.

(Bearbeiten ▷ Einstellungen ▷ Allgemeine Optionen ▷ Allgemeine Optionen ▷ Origin of coordinates) Außerdem können Sie die über Window open offset angegebenen Offset-Werte anpassen. Diese Werte werden zum Zeilen- bzw. zum Spaltenindex addiert. Siehe auch Menü Edit im „HDevelop User's Guide".

Bei Grafikfenstern auf dem Canvas können die Parameter Row und Column verwendet werden, um die Grafikfenster relativ zueinander anzuordnen. Eine absolute Positionierung ist nicht möglich, da der Canvas seine Grafikfenster zentriert.

Grafikfenster-Visualisierung

Die Hintergrundfarbe des Grafikfensters wird über den Parameter Background gesetzt. Dieser Parameter ist beim HALCON-Operator open_window nicht verfügbar. Um das gleiche Verhalten zu erreichen wie mit dev_open_window muss vor dem Öffnen des Fensters set_window_attr(::'background_color',Background:) aufgerufen werden.

Der sichtbare Bildausschnitt im Grafikfenster (viewport) wird per Default so eingestellt, dass das aktuelle Bild vollständig ausgegeben wird und das Fenster dabei vollständig ausgefüllt wird. Dabei wird der Bildausschnitt entsprechend dieser Vorgabe automatisch immer dann angepasst, wenn zum ersten Mal nach einem Programmreset bzw. dem Laden eines neuen Programms ein Bild ausgegeben wird oder wenn das aktuelle Bild eine andere Bildgröße als das zuletzt Ausgegebene besitzt. Die Fenstergröße wird dabei nicht automatisch angepasst, so dass bei unterschiedlichen Seitenverhältnissen von Bild und Fenster das Bild verzerrt ausgegeben wird. Dies kann über das Menü Fenstergröße behoben werden.

Der sichtbare Bildausschnitt kann interaktiv verändert werden. Entweder durch Drehen des Mausrades, durch Verschieben und Zoomen in den entsprechenden Modi, über das Menü Bildgröße und die Zoom Tabkarte des Visualisierungs Parameter Dialogs, oder mit Hilfe des Operators dev_set_part.

Zum Setzen der Ausgabeparameter des Grafikfensters stehen verschiedene Möglichkeiten zur Verfügung: das Kontextmenü des Grafikfensters, das Visualisierungsmenü im Hauptmenü, der Visualisierungsparameter-Dialog und die entsprechenden HDevelop-Operatoren wie dev_set_color, dev_set_line_width, dev_set_draw, usw. In Abhängigkeit von der Auswahlbox Sofort anwenden werden die Änderungen unmittelbar auf das zuletzt ausgegebene Bildobjekt angewendet oder sie gelten erst für alle zukünftigen Objekte. Im Gegensatz zu den Standard-HALCON-Fensteroperatoren gelten diese Einstellungen auch für alle Grafikfenster, die danach geöffnet werden.

Grafikfenster-Historie

Jedes Grafikfenster verwaltet eine Historie mit den

- Bildobjekten und
- Ausgabeparametern

die ausgegeben bzw. eingestellt wurden nach dem letzten Löschen des Fensterinhalts oder der Ausgabe eines Bildes mit voller Domäne. Diese Historie wird beim Neuzeichnen des Grafikfensters genutzt, um beispielsweise bei einer Änderung der Größe des Grafikfensters den kompletten Fensterinhalt rekonstruieren zu können. Andere Bildobjekte, die mit Hilfe von HALCON-Operatoren, wie disp_image oder disp_region, ausgegeben wurden, Texte (write_string) und geometrische Objekte (disp_line, disp_circle, usw.) sind **nicht** Bestandteil der Historie und können daher beim Neuaufbau des Grafikfensters nicht wieder hergestellt werden. Nur die Bildobjekte (Bilder, Regionen und XLDs), die mit Hilfe des HDevelop-Operators dev_display oder durch HDevelop-Aktionen wie Doppelklick auf das entsprechende Icon im Variablenfenster ausgegeben wurden, sind Teil der Historie.

Durch Drücken des Löschen-Knopfes in der Werkzeugleiste kann der gesamte Inhalt des Grafikfensters einschließlich der Historie der ausgegebenen Objekte gelöscht werden. Die Ausgabeparameter werden dadurch nicht verändert.

Weitere Informationen

Weitere Informationen über das dem HDevelop-Grafikfenster zugrundeliegende HALCON-Fenster finden sich beim HALCON-Operator open_window. Für eine detaillierte Beschreibung des HDevelop-Grafikfensters sei auf den „HDevelop User's Guide" verwiesen.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's

Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **Row** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilenindex der linken oberen Ecke.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ Row
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Row >= 0

▷ **Column** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltenindex der linken oberen Ecke.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ Column
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Column >= 0

▷ **Width** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
Breite des Fensters.
Defaultwert: 512
Typischer Wertebereich: 0 ≤ Width
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Width > 0 || Width == -1

▷ **Height** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
Höhe des Fensters.
Defaultwert: 512
Typischer Wertebereich: 0 ≤ Height
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Height > 0 || Height == -1

▷ **Background** (input_control) . . . . . integer ⇝ *integer* / string
Hintergrundfarbe des neuen Fensters.
Defaultwert: 'black'

▷ **WindowHandle** (output_control) . . . . . window ⇝ *handle*
Fenster-Handle.

Beispiel

```
dev_close_window ()
read_image (For5, 'for5')
get_image_size (For5, Width, Height)
dev_open_window (0, 0, Width, Height, 'black', WindowHandle)
dev_display (For5)
dev_set_lut ('rainbow')
dev_display (For5)
stop ()
dev_set_lut ('default')
dev_display (For5)
stop ()
dev_set_part (100, 100, 300, 300)
dev_display (For5)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_open_window den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

dev_display, dev_set_lut, dev_set_color, dev_set_draw, dev_set_part

Alternativen

open_window

Siehe auch

query_color

Modul

Foundation

dev_set_check (: : Mode :)

Festlegen des Fehlerverhaltens in HDevelop.

dev_set_check legt fest, wie HDevelop im Falle eines Fehlers reagieren soll, also dann, wenn der Rückgabewert eines Operators nicht H_MSG_TRUE (2) ist.

Hat Mode den Wert *'give_error'* – was der Standardeinstellung entspricht – so wird im Fehlerfall eine Exception geworfen, die innerhalb des HDevelop-Programms von einer catch-Anweisung gefangen werden kann. Befindet sich jedoch kein umgebender try-catch-Block im HDevelop-Programm, wird die Programmausführung beim fehlerverursachenden Operator angehalten und ein Fehlerdialog mit der entsprechenden Fehlermeldung angezeigt. Darüber hinaus wird der entsprechende Operatoraufruf im *Operatorfenster* angezeigt, damit der Fehler möglichst gleich behoben werden kann. Wenn die Prozedur aus der HDevEngine aufgerufen und die Exception nicht innerhalb des HDevelop-Programms abgefangen wurde, wird eine HDevEngineException geworfen, sowie die Prozedur verlassen.

Hat Mode den Wert *'~give_error'*, wird der Fehler ignoriert und die Programmausführung mit dem nächsten Operator fortgesetzt. dev_set_check('~give_error') sollte in Verbindung mit dev_error_var verwendet werden, was die Auswertung der Rückgabewerte der Operatoraufrufe ermöglicht.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **Mode** (input_control) ..... string ⇝ *string*
Art des Fehlerverhaltens.
Defaultwert: 'give_error'

Beispiel

```
dev_close_window ()
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
dev_error_var (Error, 1)
dev_set_check ('~give_error')
FileName := 'wrong_name'
read_image (Image, FileName)
ReadError := Error
if (ReadError != H_MSG_TRUE)
  write_string (WindowHandle, 'wrong file name: '+FileName)
endif
* Now the program will stop with an exception
dev_set_check ('give_error')
read_image (Image, FileName)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_check den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

dev_error_var

Siehe auch

set_check, try, catch, endtry

Modul

Foundation

dev_set_color (: : ColorName :)

Setzen einer oder mehrerer Ausgabefarben.

dev_set_color legt eine oder mehrere Farben fest, die zur Darstellung von Regionen, XLDs und anderen geometrischen Objekten im Grafikfenster verwendet werden. Die verfügbaren Farben können mit dem Operator query_color erfragt werden. Darüber hinaus kann ColorName auch über ein RGB-Tripel oder RGBA-Quadrupel in der Form *'#rrggbb'* bzw. *'#rrggbbaa'* gesetzt werden, wobei 'rr', 'gg', 'bb' und 'aa' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind. 'aa' ist der Alpha Wert der Farbe und kann genutzt werden, um transparente Regionen anzuzeigen.

Weitere Informationen finden sich in der Beschreibung des Operators set_color. Im Gegensatz zu jenem Operator gelten die Farbeinstellungen jedoch auch für alle nachträglich geöffneten Grafikfenster.

Die Farbeinstellungen bleiben bis zum nächsten Aufruf von dev_set_color oder dev_set_colored gültig, sofern sie nicht interaktiv modifiziert werden.

Color name	75% alpha	50% alpha	25% alpha
'black'	'#000000c0'	'#00000080'	'#00000040'
'white'	'#ffffffc0'	'#ffffff80'	'#ffffff40'
'red'	'#ff0000c0'	'#ff000080'	'#ff000040'
'green'	'#00ff00c0'	'#00ff0080'	'#00ff0040'
'blue'	'#0000ffc0'	'#0000ff80'	'#0000ff40'
'dim gray'	'#696969c0'	'#69696980'	'#69696940'
'gray'	'#bebebec0'	'#bebebe80'	'#bebebe40'
'light gray'	'#d3d3d3c0'	'#d3d3d380'	'#d3d3d340'
'cyan'	'#00ffffc0'	'#00ffff80'	'#00ffff40'
'magenta'	'#ff00ffc0'	'#ff00ff80'	'#ff00ff40'
'yellow'	'#ffff00c0'	'#ffff0080'	'#ffff0040'
'medium slate blue'	'#7b68eec0'	'#7b68ee80'	'#7b68ee40'
'coral'	'#ff7f50c0'	'#ff7f5080'	'#ff7f5040'
'slate blue'	'#6a5acdc0'	'#6a5acd80'	'#6a5acd40'
'spring green'	'#00ff7fc0'	'#00ff7f80'	'#00ff7f40'
'orange red'	'#ff4500c0'	'#ff450080'	'#ff450040'
'dark olive green'	'#556b2fc0'	'#556b2f80'	'#556b2f40'
'pink'	'#ffc0cbc0'	'#ffc0cb80'	'#ffc0cb40'
'cadet blue'	'#5f9ea0c0'	'#5f9ea080'	'#5f9ea040'
'goldenrod'	'#daa520c0'	'#daa52080'	'#daa52040'
'orange'	'#ffa500c0'	'#ffa50080'	'#ffa50040'
'gold'	'#ffd700c0'	'#ffd70080'	'#ffd70040'
'forest green'	'#228b22c0'	'#228b2280'	'#228b2240'
'cornflower blue'	'#6495edc0'	'#6495ed80'	'#6495ed40'
'navy'	'#000080c0'	'#00008080'	'#00008040'
'turquoise'	'#40e0d0c0'	'#40e0d080'	'#40e0d040'
'dark slate blue'	'#483d8bc0'	'#483d8b80'	'#483d8b40'
'light blue'	'#add8e6c0'	'#add8e680'	'#add8e640'
'indian red'	'#cd5c5cc0'	'#cd5c5c80'	'#cd5c5c40'
'violet red'	'#d02090c0'	'#d0209080'	'#d0209040'
'light steel blue'	'#b0c4dec0'	'#b0c4de80'	'#b0c4de40'
'medium blue'	'#0000cdc0'	'#0000cd80'	'#0000cd40'
'khaki'	'#f0e68cc0'	'#f0e68c80'	'#f0e68c40'
'violet'	'#ee82eec0'	'#ee82ee80'	'#ee82ee40'
'firebrick'	'#b22222c0'	'#b2222280'	'#b2222240'
'midnight blue'	'#191970c0'	'#19197080'	'#19197040'

Beispiele für mögliche Farbstrings

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **ColorName** (input_control) ..... string(-array) ⇝ *string*
Namen der Ausgabefarben.
Defaultwert: 'white'
Wertevorschläge: ColorName ∈ {'white', 'black', 'gray', 'red', 'green', 'blue', '#003075', '#e53019', '#ffb529'}

Beispiel

```
read_image(Image,'mreut')
dev_set_draw('fill')
dev_set_color('red')
threshold(Image,Region,180,255)
dev_set_color('green')
threshold(Image,Region,0,179)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_color den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_open_window, query_color, query_all_colors

Nachfolger

dev_display

Alternativen

dev_set_colored

Siehe auch

dev_set_draw, dev_set_line_width, set_color

Modul

Foundation

dev_set_colored (: : NumColors :)

Setzen einer bunten Ausgabe.

dev_set_colored legt fest, dass Tupel von Regionen, XLDs und anderen geometrischen Objekten im Grafikfenster in einer Auswahl von NumColors vordefinierten Farben dargestellt werden. Gültige Werte für NumColors können mit dem Operator query_colored erfragt werden.

Weitere Informationen finden sich in der Beschreibung des Operators set_colored. Im Gegensatz zu jenem Operator gelten die Farbeinstellungen jedoch auch für alle nachträglich geöffneten Grafikfenster.

Die Farbeinstellungen bleiben bis zum nächsten Aufruf von dev_set_color oder dev_set_colored gültig, sofern sie nicht interaktiv modifiziert werden.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **NumColors** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Ausgabefarben.
Defaultwert: 6
Werteliste: NumColors ∈ {3, 6, 12}

Beispiel

```
read_image(Image,'monkey')
threshold(Image,Region,128,255)
dev_set_colored(6)
connection(Region,Regions)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_colored den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_open_window

Nachfolger

dev_display

Alternativen

dev_set_color

Siehe auch

dev_set_draw, dev_set_line_width, set_colored

Modul

Foundation

dev_set_contour_style (: : Style :)

Setzen des Füllstils der Konturanzeige.

dev_set_contour_style setzt den Füllstil der Konturanzeige. Dabei werden folgende Werte für Style unterstützt:

- *'stroke'*: Nur die Linie der Kontur wird gezeichnet
- *'fill'*: Die von der Kontur eingeschlossene Fläche wird ausgefüllt.
- *'stroke_and_fill'*: Die Linie der Kontur wird gezeichnet und die von ihr eingeschlossene Fläche ausgefüllt. Dies macht insbesondere bei größeren Liniendicken einen Unterschied, siehe set_line_width.

Bei allen Stilen wird die aktuelle Zeichenfarbe verwendet.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **Style** (input_control) . . . . . string ⇝ *string*
Füllstil der Konturanzeige.
Defaultwert: 'stroke'
Werteliste: Style ∈ {'stroke', 'fill', 'stroke_and_fill'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_contour_style den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

dev_display

Siehe auch

set_contour_style

Modul

Foundation

dev_set_draw (: : DrawMode :)

Ausfüllmodus für Regionen setzen.

dev_set_draw legt den Ausfüllmodus für Regionen fest. Ist DrawMode auf *'fill'* gesetzt, werden Regionen ausgefüllt dargestellt. In der Einstellung *'margin'* werden nur Konturen gezeichnet, deren Darstellung von dev_set_line_width sowie set_line_style beeinflusst wird.

Weitere Informationen finden sich in der Beschreibung des Operators set_draw. Im Gegensatz zu jenem Operator gelten die Zeicheneinstellungen jedoch auch für alle nachträglich geöffneten Grafikfenster.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **DrawMode** (input_control) ..... string ⇝ *string*
Ausfüllmodus für Regionen.
Defaultwert: 'fill'
Werteliste: DrawMode ∈ {'fill', 'margin'}

Beispiel

```
read_image(Image,'monkey')
threshold(Image,Region,128,255)
dev_clear_window ()
dev_set_color('red')
dev_set_draw('fill')
dev_display(Region)
dev_set_color('white')
dev_set_draw('margin')
dev_display(Region)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_draw den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

dev_set_line_width, dev_display

Siehe auch

set_draw

Modul

Foundation

dev_set_line_width (: : LineWidth :)

Festlegen der Strichstärke für die Konturdarstellung.

dev_set_line_width legt die Strichstärke in Pixeln fest, die zur Darstellung von Regionenkonturen (im Modus *'margin'*), von XLDs und anderen geometrischen Ausgaben (z.B. mit disp_region, disp_line usw.) verwendet wird.

Weitere Informationen finden sich in der Beschreibung des Operators set_line_width. Im Gegensatz zu jenem Operator gilt die neue Strichstärke jedoch auch für alle nachträglich geöffneten Grafikfenster.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **LineWidth** (input_control) ..... integer ⇝ *integer*
Strichstärke für Konturdarstellung.
Defaultwert: 1
Restriktion: LineWidth >= 1

Beispiel

```
read_image(Image,'monkey')
threshold(Image,Region,128,255)
dev_set_draw('margin')
dev_set_line_width(5)
dev_clear_window ()
dev_display(Region)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_line_width den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

dev_display

Siehe auch

set_line_width, query_line_width

Modul

Foundation

dev_set_lut (: : LutName :)

Setzen einer Farbtabelle.

dev_set_lut legt die Farbtabelle des aktiven Grafikfensters fest. Mithilfe einer Farbtabelle wird die Transformation eines Grauwertes eines einkanaligen Bildes auf einen Grauwert oder eine Farbe für die Darstellung festgelegt. query_lut liefert die Namen aller Farbtabellen.

Weitere Informationen finden sich in der Beschreibung des Operators set_lut. Im Gegensatz zu jenem Operator gilt die neue Farbtabelle jedoch auch für alle nachträglich geöffneten Grafikfenster.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **LutName** (input_control) . . . . . filename.read ⇝ *string*
Name der Farbtabelle, Werte der Farbtabelle (RGB) oder der Name der Datei.
Defaultwert: 'default'
Wertevorschläge: `LutName` ∈ {'default', 'linear', 'inverse', 'sqr', 'inv_sqr', 'cube', 'inv_cube', 'sqrt', 'inv_sqrt', 'cubic_root', 'inv_cubic_root', 'color1', 'color2', 'color3', 'color4', 'three', 'six', 'twelve', 'twenty_four', 'rainbow', 'temperature', 'cyclic_gray', 'cyclic_temperature', 'hsi', 'change1', 'change2', 'change3'}
Dateiendung: `.lut`

Beispiel

```
read_image(Image,'mreut')
dev_set_lut('inverse')
* For true color only:
dev_display(Image)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dev_set_lut` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

`dev_display`

Siehe auch

`set_lut`

Modul

Foundation

dev_set_paint (: : Mode :)

Modus für die Darstellung von Grauwerten festlegen.

`dev_set_paint` legt den Modus für die Darstellung von Bildobjekten im Grafikfenster fest.

Weitere Informationen finden sich in der Beschreibung des Operators `set_paint`. Im Gegensatz zu jenem Operator gilt der neue Ausgabemodus jedoch auch für alle nachträglich geöffneten Grafikfenster.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „`HDevelop User's Guide`" im Kapitel `Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows`.

Parameter

▷ **Mode** (input_control) . . . . . string-array ⇝ *string* / integer
Name des Grauwertmodus, eventuell mit Parameter.
Defaultwert: 'default'
Wertevorschläge: `Mode` ∈ {'default', '3d_plot'}

Beispiel

```
read_image(Image,'fabrik')
dev_set_paint('3d_plot')
dev_display(Image)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dev_set_paint` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_open_window

Nachfolger

dev_set_color, dev_display

Siehe auch

set_paint

Modul

Foundation

dev_set_part (: : Row1, Column1, Row2, Column2 :)

Anpassung des darzustellenden Bildausschnitts.

dev_set_part legt den Bildausschnitt fest, der im Grafikfenster angezeigt wird. Die Parameter Row1 und Column1 beziehen sich auf die obere linke Ecke, die Parameter Row2 und Column2 auf die untere rechte Ecke des darzustellenden Bildes.

Weitere Informationen finden sich in der Beschreibung des Operators set_part. Wird Row1 größer als Row2 bzw. Column1 größer als Column2 gewählt, so wird der Bildausschnitt in der entsprechenden Richtung zurückgesetzt und das Bild in der ursprünglichen Höhe bzw. Breite dargestellt. Dieses Verhalten ist bei Verwendung des Operators set_part außerhalb von HDevelop nicht möglich.

Der Bildausschnitt wird darüber hinaus automatisch zurückgesetzt, sobald ein neues Programm geladen, ein Programmreset durchgeführt oder ein neues Bild abweichender Größe dargestellt wird.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

- ▷ **Row1** (input_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilennummer des linken oberen Eckpunkts des Bildausschnitts.
  Defaultwert: 0
- ▷ **Column1** (input_control) ..... rectangle.origin.x ⇝ *integer*
  Spaltennummer des linken oberen Eckpunkts des Bildausschnitts.
  Defaultwert: 0
- ▷ **Row2** (input_control) ..... rectangle.corner.y ⇝ *integer*
  Zeilennummer des rechten unteren Eckpunkts des Bildausschnitts.
  Defaultwert: 128
- ▷ **Column2** (input_control) ..... rectangle.corner.x ⇝ *integer*
  Spaltennummer des rechten unteren Eckpunkts des Bildausschnitts
  Defaultwert: 128

Beispiel

```
read_image (Image, 'fabrik')
for i := 1 to 240 by 10
  dev_set_part (i, i, 511-i, 511-i)
  dev_display (Image)
endfor
dev_set_part (1, 1, -1, -1)
dev_display (Image)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_part den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

dev_display

Siehe auch

set_part

Modul

Foundation

dev_set_preferences (: : PreferenceNames, PreferenceValues :)

Setzen von HDevelop-Einstellungen aus einem Programm heraus.

dev_set_preferences ermöglicht das Setzen ausgewählter HDevelop-Einstellungen aus dem Programm heraus. Die folgenden Einstellungen werden unterstützt:

'graphics_window_context_menu': Legt fest, ob ein Rechtsklick im Grafikfenster das Kontextmenü öffnet oder nicht. Standardmäßig ist das Kontextmenü aktiviert. Es ist sinnvoll, das Kontextmenü zu deaktivieren, wenn die rechte Maustaste für andere Aktionen im Grafikfenster verwendet wird, z.B. zum Verschieben oder Vergrößern von 3D-Objekten.

Werteliste: *'false'*, *'true'*.

Defaultwert: *'true'*.

'graphics_window_mouse_wheel': Legt fest, ob das Mausrad im Grafikfenster zum Zoomen genutzt werden kann. Standardmäßig ist das Mausrad aktiviert. Es kann aber sinnvoll sein, das Zoomen mit dem Mausrad zu deaktivieren, wenn beispielsweise 3D-Objekte dargestellt werden sollen und das Heranzoomen an ein Objekt interaktiv mit Hilfe von 3D-Darstellungsoperatoren erfolgen soll.

Werteliste: *'false'*, *'true'*.

Defaultwert: *'true'*.

'graphics_window_tool_tip': Legt fest, ob im Grafikfenster beim Drücken der Ctrl-Taste ein Tooltipp mit der Mausposition und dem Grauwert unter der Maus erscheinen soll oder nicht. Standardmäßig ist der Tooltipp aktiviert. Es kann aber sinnvoll sein, den Tooltipp zu unterdrücken, um bei der programmbasierten Interaktionen den Status der Ctrl-Taste mit verwenden zu können.

Werteliste: *'false'*, *'true'*.

Defaultwert: *'true'*.

'suppress_handled_exceptions_dlg': Legt fest, ob bei einer Exception, die innerhalb eines try-catch Blocks geworfen wird, der Fehler-Dialog unterdrückt werden soll oder nicht. Diese Option wird persistent in die HDevelop.ini Datei geschrieben und kann auch über den Einstellungen Dialog / Allgemeine Optionen / Erfahrene Anwender konfiguriert werden. Wird eine Exception in einer Programmzeile geworfen, die nicht von einem try-catch Block umgeben ist, wird immer ein Fehlerdialog geöffnet.

Werteliste: *'false'*, *'true'*.

Defaultwert: *'false'* (Jede Änderung wird persistent gespeichert).

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **PreferenceNames** (input_control) . . . . . attribute.name-array ⇝ *string*
Auswahl der zu setzenden Präferenzen.
Defaultwert: 'graphics_window_context_menu'
Werteliste: PreferenceNames ∈ {'graphics_window_context_menu', 'graphics_window_mouse_wheel', 'graphics_window_tool_tip', 'suppress_handled_exception_dlg'}

▷ **PreferenceValues** (input_control) . . . . . attribute.value-array ⇝ *string*
Neu einzustellende Werte für die ausgewählten Präferenzen.
Defaultwert: 'false'
Werteliste: PreferenceValues ∈ {'true', 'false'}

Siehe auch

dev_get_preferences

Modul

Foundation

dev_set_shape (: : Shape :)

Festlegen der Ausgabeform für Regionen.

dev_set_shape legt die Ausgabeform für die Darstellung von Regionen fest. Die verfügbaren Formen lassen sich mit query_shape erfragen.

Die folgenden Ausgabeformen werden unterstützt:

'original': Die Form wird unverändert dargestellt. Allerdings können Einstellungen wie dev_set_line_width die Darstellung beeinflussen. Dies gilt auch für alle anderen Modi.

'outer_circle': Jede Region wird durch den kleinsten umschließenden Kreis dargestellt (siehe smallest_circle).

'inner_circle': Jede Region wird durch den größten Inkreis dargestellt (siehe inner_circle).

'ellipse': Jede Region wird durch eine Ellipse mit denselben Momenten und derselben Orientierung dargestellt (siehe elliptic_axis).

'rectangle1': Jede Region wird durch das kleinste umschließende Rechteck dargestellt, das parallel zu den Koordinatenachsen liegt (siehe smallest_rectangle1).

'rectangle2': Jede Region wird durch das kleinste umschließende Rechteck dargestellt (siehe smallest_rectangle2).

'convex': Jede Region wird durch ihre konvexe Hülle dargestellt (siehe shape_trans).

'icon': Jede Region wird durch das mittels set_icon festgelegte Icon im Schwerpunkt der Region dargestellt.

Weitere Informationen finden sich in der Beschreibung des Operators set_shape. Im Gegensatz zu jenem Operator gilt die neue Ausgabeform jedoch auch für alle nachträglich geöffneten Grafikfenster.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide“ im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **Shape** (input_control) ..... string ⇝ *string*
Art der Regionenausgabe.
Defaultwert: 'original'
Werteliste: Shape ∈ {'original', 'convex', 'outer_circle', 'inner_circle', 'rectangle1', 'rectangle2', 'ellipse', 'icon'}

Beispiel

```
read_image(Image,'monkey')
threshold(Image,Region,128,255)
connection(Region,Regions)
dev_set_shape('rectangle1')
dev_set_draw('margin')
dev_set_line_width(5)
dev_clear_window()
dev_display(Regions)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_shape den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

dev_display, dev_set_color

Siehe auch

set_shape, dev_set_line_width

Modul

Foundation

dev_set_tool_geometry (: : ToolId, Row, Column, Width, Height :)

Setzt die Position und Dimension des Tools ToolId.

dev_set_tool_geometry setzt die Position und die Dimension des Tools mit der id ToolId. Anstatt der ToolId kann auch der Name verwendet werden, der beim Öffnen mit dev_open_tool angeben werden muss. Allerdings wird dann ein beliebiges Fenster, zu dem der Parameter ToolId passt, verwendet.

Mit den Parametern Row und Column kann die Position des Tools geändert werden. Beachten Sie, dass die unter Bearbeiten ▷ Einstellungen ▷ Allgemeine Optionen ▷ Allgemeine Optionen ▷ Window open offset angegebenen Versatzwerte zum Zeileindex bzw. zum Spaltenindex addiert werden. Weitere Informationen finden Sie im Kapitel „Menu Edit“ im „HDevelop User's Guide“. Bei *'default'* wird die Position nicht verändert.

Mit den Parametern Width und Height kann die Größe des Tools geändert werden. Bei *'default'* wird die Größe nicht verändert.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

- ▷ **ToolId** (input_control) ..... dev_tool ⇝ *string*
  Tool-Identifikator.
- ▷ **Row** (input_control) ..... rectangle.origin.y ⇝ *integer* / string
  Zeile der oberen linken Ecke.
  Defaultwert: 'default'
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Column** (input_control) ..... rectangle.origin.x ⇝ *integer* / string
  Spalte der oberen linken Ecke.
  Defaultwert: 'default'
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Width** (input_control) ..... rectangle.extent.x ⇝ *integer* / string
  Breite des Tools.
  Defaultwert: 'default'
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: Width > 0
- ▷ **Height** (input_control) ..... rectangle.extent.y ⇝ *integer*
  Höhe des Tools.
  Defaultwert: 'default'
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: Height > 0

Beispiel

```
dev_open_tool ('zoom_window', 0, 0, Width, 'default', ToolId)
dev_close_tool (ToolId)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_window den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_open_tool

Nachfolger

dev_show_tool, dev_close_tool

Alternativen

dev_show_tool

Siehe auch

dev_show_tool

Modul

Foundation

dev_set_window (: : WindowHandle :)

Aktivieren eines Grafikfensters.

dev_set_window aktiviert das Grafikfenster, das dem übergebenen Fensterhandle entspricht. Der Aufruf ist äquivalent zum Drücken des Aktivierungsknopfes in der Aktionsleiste des Grafikfensters.

Achtung

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide" im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

Beispiel

```
dev_open_window (1, 1, 200, 200, 'black', WindowID1)
dev_open_window (1, 220, 200, 200, 'black', WindowID2)
read_image(Image,'monkey')
dev_set_window(WindowID1)
dev_display(Image)
dev_set_window(WindowID2)
dev_display(Image)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_set_window den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

dev_open_window

Nachfolger

dev_display

Modul

Foundation

dev_set_window_extents (: : Row, Column, Width, Height :)

Position und Größe des aktiven Grafikfensters verändern.

dev_set_window_extents setzt für das aktive schwebende Grafikfenster die Position und/oder die Größe neu.

Die Parameter Row und Column bestimmen die neue Position des Fensters (linke obere Ecke). Beachten Sie, dass die unter Bearbeiten ▷ Einstellungen ▷ Allgemeine Optionen ▷ Allgemeine Optionen ▷ Window open offset angegebenen Versatzwerte zum Zeileindex bzw. zum Spaltenindex addiert werden. Weitere Informationen finden Sie im Kapitel „Menu Edit“ im „HDevelop User's Guide“. Negative Koordinaten werden ignoriert, d.h. das Fenster wird nicht in Richtung der negativen Koordinate(n) verschoben.

Die Parameter Width und Height bestimmen die neue Größe des Fensters, und zwar die Größe des inneren Teils des Grafikfensters – der eigentlichen Ausgabefläche für die ikonischen Objekte. Negative Werte für die Größe werden ebenfalls ignoriert, d.h. in dieser Richtung ändert sich die Größe nicht.

Achtung

Dieser Operator funktioniert nur für einzelne schwebende Grafikfenster, d.h. Grafikfenster, die weder angedockt noch mit Registerkarten versehen sind.

Ein HDevelop Grafikfenster darf nicht mit set_window_extents verschoben oder verändert werden. Dies darf nur mit dem Operator dev_set_window_extents geschehen.

Beim Export von HDevelop-Programmen ist zu beachten, dass der für diesen Operator generierte Code sich anders verhalten kann, als der entsprechende HALCON-Operator. Eine detaillierte Beschreibung der Besonderheiten, die beim Export von grafischen HDevelop-Operatoren zu beachten sind, findet sich im „HDevelop User's Guide“ im Kapitel Code Export ▷ General Aspects of Code Generation ▷ Graphics Windows.

Parameter

- ▷ **Row** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
  Zeilenindex der linken oberen Ecke.
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ Row
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: Row >= 0 || Row == -1
- ▷ **Column** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
  Spaltenindex der linken oberen Ecke.
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ Column
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: Column >= 0 || Column == -1
- ▷ **Width** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
  Breite des Fensters.
  Defaultwert: 256
  Typischer Wertebereich: 0 ≤ Width
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: Width > 0 || Width == -1
- ▷ **Height** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
  Höhe des Fensters.
  Defaultwert: 256
  Typischer Wertebereich: 0 ≤ Height
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: Height > 0 || Height == -1

Beispiel

```
dev_close_window ()
read_image (For5, 'for5')
get_image_size (For5, Width, Height)
dev_open_window (0, 0, Width, Height, 'black', WindowHandle)
dev_display (For5)
```

```
stop ()
dev_set_window_extents (-1,-1,Width/2,Height/2)
dev_display (For5)
stop ()
dev_set_window_extents (200,200,-1,-1)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dev_set_window_extents` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

`dev_display`, `dev_set_lut`, `dev_set_color`, `dev_set_draw`, `dev_set_part`

Siehe auch

`set_window_extents`

Modul

Foundation

dev_show_tool (: : ToolId, Action :)

Zeigt das angegebene Tool.

`dev_show_tool` führt die Aktion `Action` für das Tool `ToolId` aus.

Anstatt der ToolId kann auch der Name verwendet werden, der beim Öffnen mit `dev_open_tool` angeben werden muss. Allerdings wird dann ein beliebiges Fenster, zu dem der Parameter `ToolId` passt, verwendet.

`Action`

'show': Aktiviert das Tool.

'normal': Zeigt das Tool, bringt es in den Vordergrund und aktiviert es.

'minimize': Minimiert das Tool.

'maximize': Maximiert das Tool.

'hide': Verbirgt das Tool.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

- ▷ **ToolId** (input_control) ..... dev_tool ⇝ *string*
  Tool-Identifikator.
- ▷ **Action** (input_control) ..... string ⇝ *string*
  Auszuführende Aktion.

Beispiel

```
dev_open_tool ('matching_assistant', 0, 0, 'default','default', ToolId)
dev_show_tool (ToolId, 'minimize')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dev_set_window` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Vorgänger

`dev_set_tool_geometry`, `dev_open_tool`

Siehe auch

`dev_set_tool_geometry`

Modul

Foundation

dev_update_pc (: : DisplayMode :)

Aktualisierung des PCs beim Programmablauf ein- oder ausschalten.

dev_update_pc legt das Verhalten des PCs während der Programmausführung fest. Ist DisplayMode auf *'on'* gesetzt, was der Standardeinstellung entspricht, wird der PC innerhalb der ausgewählten Prozedur jeweils links vom gerade ausgeführten Operator angezeigt. Zusätzlich läuft der Programmtext gegebenenfalls mit, damit der aktuelle Operator sichtbar ist.

Ist DisplayMode auf *'off'* gesetzt, wird der PC während der Programmausführung nicht angezeigt, und der Programmtext läuft nicht automatisch mit. Sobald das Programm angehalten wird, werden der PC und der aktuelle Operator wieder angezeigt.

Um die Ausführungszeit einer Operatorsequenz zu messen, sollten alle update-Optionen ausgeschaltet sein, um den Einfluss der für die GUI-Aktualisierungen in HDevelop benötigten Laufzeit zu minimieren. Zu diesem Zwecke können die Operatoren dev_update_pc, dev_update_time, dev_update_var und dev_update_window sowie die Prozeduren dev_update_on und dev_update_off verwendet werden.

Diese Option kann auch über den Dialog Einstellungen gesetzt werden: Bearbeiten ▷ Einstellungen ▷ Laufzeiteinstellungen ▷ Ausführungsmarke aktualisieren.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **DisplayMode** (input_control) . . . . . string ⇝ *string*
Modus für die Darstellung.
Defaultwert: 'off'
Werteliste: DisplayMode ∈ {'on', 'off'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_update_pc den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Siehe auch

dev_update_time, dev_update_window, dev_update_var

Modul

Foundation

dev_update_time (: : DisplayMode :)

Zeitmessung von Operatoren ein- oder ausschalten.

dev_update_time legt fest, ob die Ausführungszeit eines Operators angezeigt wird.

Um die Ausführungszeit einer Operatorsequenz zu messen, sollten alle update-Optionen ausgeschaltet sein, um den Einfluss der für die GUI-Aktualisierungen in HDevelop benötigten Laufzeit zu minimieren. Zu diesem Zwecke können die Operatoren dev_update_pc, dev_update_time, dev_update_var und dev_update_window sowie die Prozeduren dev_update_on und dev_update_off verwendet werden.

Diese Option kann auch über den Dialog Einstellungen gesetzt werden: Bearbeiten ▷ Einstellungen ▷ Laufzeiteinstellungen ▷ Laufzeit anzeigen.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **DisplayMode** (input_control) . . . . . string ⇝ *string*
Modus für die Ausgabe.
Defaultwert: 'off'
Werteliste: DisplayMode ∈ {'on', 'off'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_update_time den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Siehe auch

dev_update_pc, dev_update_window, dev_update_var

Modul

Foundation

dev_update_var (: : DisplayMode :)

Aktualisierung des Variablenfensters beim Programmablauf ein- oder ausschalten.

dev_update_var legt das Aktualisierungsverhalten des Variablenfensters während der Programmausführung fest. Ist DisplayMode auf *'on'* gesetzt, was der Standardeinstellung entspricht, wird der Inhalt des Variablenfensters (ikonische und Steuervariablen) nach jeder Variablenänderung durch das Programm aktualisiert.

Ist DisplayMode auf *'off'* gesetzt, werden die Werte von Steuervariablen und die Icons von ikonischen Variablen erst nach einem Programmstopp aktualisiert.

Um die Ausführungszeit einer Operatorsequenz zu messen, sollten alle update-Optionen ausgeschaltet sein, um den Einfluss der für die GUI-Aktualisierungen in HDevelop benötigten Laufzeit zu minimieren. Zu diesem Zwecke können die Operatoren dev_update_pc, dev_update_time, dev_update_var und dev_update_window sowie die Prozeduren dev_update_on und dev_update_off verwendet werden.

Diese Option kann auch über den Dialog Einstellungen gesetzt werden: Bearbeiten ▷ Einstellungen ▷ Laufzeiteinstellungen ▷ Variablen aktualisieren.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **DisplayMode** (input_control) ..... string ⇝ *string*
Modus für die Ausgabe.
Defaultwert: 'off'
Werteliste: DisplayMode ∈ {'on', 'off'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dev_update_var den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Siehe auch

dev_update_pc, dev_update_window, dev_update_time

Modul

Foundation

dev_update_window (: : DisplayMode :)

Automatische Ausgabe von ikonischen Ausgabeobjekten ins Grafikfenster während der Programmausführung ein- oder ausschalten.

dev_update_window legt fest, ob alle ikonischen Objekte, die von Operatoraufrufen zurückgeliefert werden, im aktiven Grafikfenster angezeigt werden (DisplayMode = *'on'*, Standardeinstellung) oder nicht (DisplayMode = *'off'*).

Die gewählte Einstellung hat keinen Einfluss auf die Objektausgabe im Einzelschrittmodus. Die Ausführung einzelner Operatoren führt immer zur Darstellung der ikonischen Ausgabeobjekte im aktiven Grafikfenster.

Diese Einstellung sollte auf *'off'* gesetzt werden, wenn nur ausgewählte Objekte im Grafikfenster angezeigt werden sollen. Diese sollten dann mit dev_display dargestellt werden.

Um die Ausführungszeit einer Operatorsequenz zu messen, sollten alle update-Optionen ausgeschaltet sein, um den Einfluss der für die GUI-Aktualisierungen in HDevelop benötigten Laufzeit zu minimieren. Zu diesem Zwecke können die Operatoren dev_update_pc, dev_update_time, dev_update_var und dev_update_window sowie die Prozeduren dev_update_on und dev_update_off verwendet werden.

Diese Option kann auch über den Dialog `Einstellungen` gesetzt werden: `Bearbeiten` ▷ `Einstellungen` ▷ `Laufzeiteinstellungen` ▷ `Grafikfenster aktualisieren`.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Parameter

▷ **`DisplayMode`** (input_control) . . . . . string ⇝ *string*
Modus für die Ausgabe.
Defaultwert: 'off'
Werteliste: `DisplayMode` ∈ {'on', 'off'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dev_update_window` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Nachfolger

`dev_display`

Siehe auch

`dev_update_pc`, `dev_update_var`, `dev_update_time`

Modul

Foundation

Kapitel 10

Filter

Dieses Kapitel beschreibt Operatoren, die zum Filtern verwendet werden können.

Filter sind ein fester Bestandteil von fast jeder Aufgabe der Bildverarbeitung. Beispielsweise kann `mean_image` zum Glätten von Bildern verwendet werden, `edges_sub_pix` extrahiert sub-pixel genau Kanten, und `fft_image` kann verwendet werden, um die Fourier-Transformation eines Bildes zu berechnen.

In diesem Kapitel werden wir einen genaueren Blick auf spezielle Probleme werfen: Der Verwendung eines Bildes mit einer reduzierten Domäne und der Problematik, was passiert, wenn mehrere Filter nacheinander auf ein Bild angewendet werden.

Filter, Masken und Domänen

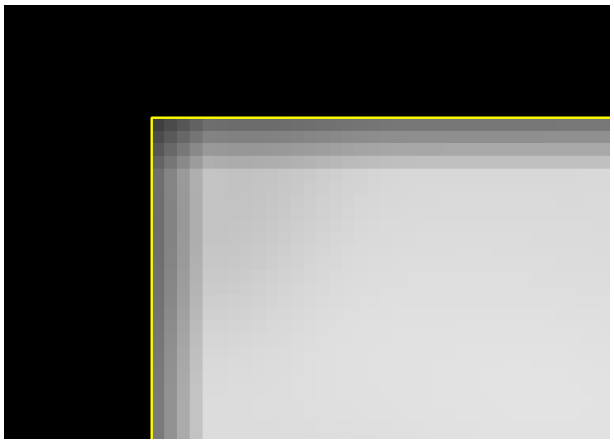
Wenn ein Filter, der eine Maske nutzt, auf ein Bild mit reduzierter Domäne angewendet wird, ist das Ergebnis an den Grenzen der Domäne eventuell überraschend, da Grauwerte, die eigentlich außerhalb der Domäne liegen, auch als Input für den Filter benutzt werden. Um dies besser zu verstehen, müssen wir uns die Definition der Domäne in diesem speziellen Fall genauer ansehen: Für einen Filter definiert die Domäne, für welche Input-Pixel Output-Pixel berechnet werden müssen. Dennoch können auch Pixel außerhalb der Domäne (die innerhalb der Bildmatrix liegen) für die Verarbeitung genutzt werden.

Probleme, die von Grauwerten außerhalb der Bilddomäne verursacht werden

Operatoren, die ein Bild als Ergebnis zurückgeben, liefern nur Resultate für Pixel zurück, die in der Eingabedomäne vorhanden waren.

Aus Performanzgründen werden Pixel außerhalb der Bilddomäne 'undefiniert'. Diese undefinierten Pixel können von System zu System unterschiedlich sein, zum Beispiel abhängig davon, ob Parallelisierung aktiv ist oder nicht. Es wird lediglich garantiert, dass sie auf identischen Systemen bei wiederholter Programmausführung konsistent sind. In bestimmten Fällen können diese 'undefinierten' Pixel zu Problemen führen. Das Vergrößern des resultierenden Bildes auf die komplette Domäne mit `full_domain` führt zu Artefakten außerhalb der ursprünglichen Domäne.

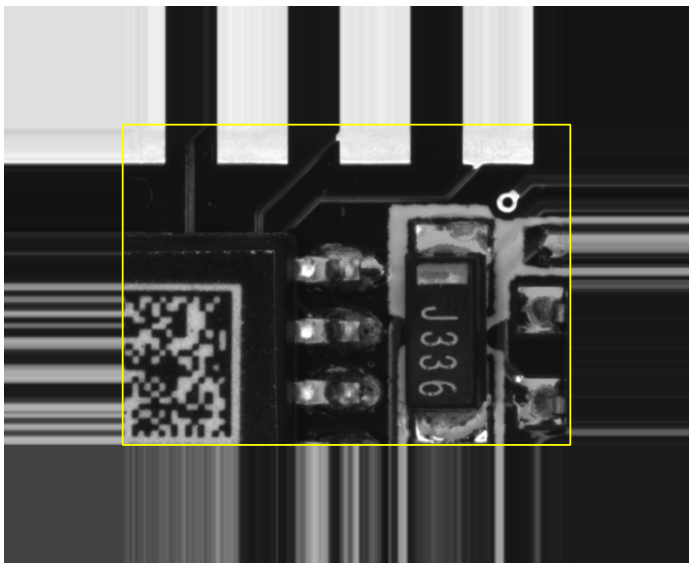
Ein anderer Grund für Probleme sind undefinierte Pixel außerhalb der Domäne, falls zwei oder mehr Filter nacheinander angewendet werden, da die Filter auch die undefinierten Pixel an der Grenze der Domäne in ihre Berechnung mit einbeziehen. Das bedeutet, dass der Fehler mit jedem nachfolgenden Filter größer wird, von der Grenze beginnend zur Mitte hin. Im Folgenden werden vier verschiedene Methoden betrachtet, wie diese Probleme gelöst werden können.



Artefakte an der Grenze der Domäne, nachdem nacheinander zwei Filter angewendet wurden.

1. Fehler, die durch undefinierte Pixel verursacht werden, können leicht vermieden werden, indem beispielsweise eine Domäne gewählt wird, die bedeutend größer ist als der Bildteil, der tatsächlich benötigt wird. Nach Anwenden des Filters kann die Domäne auf die ursprüngliche Größe reduziert werden (beispielsweise mit den Operatoren `erosion_rectangle1` und `reduce_domain`). Damit werden die Bereiche des Bildes, die falsche Werte enthalten, ausgeschlossen, und erhöhen damit nicht den Fehler für die nächsten Filter.

2. Eine andere Möglichkeit ist, die Domäne genauso groß wie benötigt festzulegen, und dann den Operator `expand_domain_gray` aufzurufen. Dann werden die Pixel innerhalb der Grenze nach außen kopiert. Damit werden Fehler vermieden, die von undefinierten Pixel außerhalb der Domäne verursacht werden. Danach kann die Domäne wieder auf ihre ursprüngliche Größe reduziert werden. Dieser Prozess sollte for jedem folgenden Filter wiederholt werden. Beachten Sie dabei, dass diese Option die Laufzeit beträchtlich erhöht.



Das Resultat von `expand_domain_gray` sieht aus, als ob die Grenze mehrmals kopiert und außen angefügt wird.

3. Falls die Laufzeit kein Thema ist, kann vor der Anwendung des ersten Filters der Operator `full_domain` aufgerufen werden. Dann ist das gesamte Bild als Domäne definiert, und undefinierte Pixel werden komplett vermieden.

4. Alternativ dazu kann auch der Operator `crop_domain` aufgerufen werden, bevor ein Filter eingesetzt wird. Dieser schneidet das Bild auf die Größe der Domäne zu - die Domäne deckt damit ein komplettes neues Bild ab. Beachten Sie allerdings, dass sich damit das Koordinatensystem des zugeschnittenen Bildes im Vergleich zum ursprünglichen Bild ändert, was alle folgenden Anwendungen betrifft, die sich auf das Bildkoordinatensystem beziehen (z. B. die Berechnung des Schwerpunktes).

10.1 Arithmetik

abs_diff_image (Image1, Image2 : ImageAbsDiff : Mult :)

Berechnen der absoluten Differenz von zwei Bildern.

abs_diff_image bildet die absolute Differenz zweier Bilder. Die Grauwerte g' des Ausgabebildes ImageAbsDiff werden aus den Grauwerten $(g1, g2)$ der Eingabebilder Image1 und Image2 werden dabei wie folgt berechnet:

$$g' = |(g1 - g2)| * \texttt{Mult}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten.

Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Es ist zu beachten, dass die Laufzeit des Operators von der Wahl von Mult abhängt. Für Mult = 1 ist eine spezielle Optimierung implementiert. Da andere Werte von Mult keine zusätzliche Information liefern können, sollte in der Anwendung Mult = 1 verwendet werden. Andere Werte von Mult sollten lediglich für Visualisierungszwecke verwendet werden. Für byte-, int1-, int2- und uint2-Bilder sind außerdem für Mult = 1 spezielle Optimierungen unter Verwendung von SIMD Technologie implementiert. Ob diese tatsächlich zur Anwendung kommen, kann über den Systemparameter *'mmx_enable'* (siehe set_system) gesteuert werden. Falls *'mmx_enable'* auf *'true'* gesetzt wird (und die verwendete Hardware SIMD Technologie unterstützt), wird die Berechnung intern unter Verwendung von SIMD Technologie durchgeführt.

abs_diff_image kann für byte-, int1-, int2-, uint2-, int4-, und real-Bildern auf OpenCL-Geräten ausgeführt werden. Da OpenCL 1.0 aber nur einfache genaue Fließkommaunterstützung voraussetzt und nicht alle Rundungsarten unterstützt werden, kann das Ergebnis der OpenCL-Implementierung geringfügig von dem der skalaren oder der SIMD-Implementierung abweichen.

Achtung

Es ist zu beachten, dass der Beschleunigungseffekt von SIMD-Technologie bei großen, kompakten Eingaberegionen am größten ist. Im Einzelfall kann aber die Ausführung von abs_diff_image, abhängig von den Eingaberegionen und den Hardwareeigenschaften, mit SIMD Technologie länger dauern als ohne. In diesem Falle lässt sich die Verwendung von SIMD-Technologie durch set_system(::'mmx_enable','false':) verhindern.

Parameter

- ▷ **Image1** (input_object) . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real
  Eingabebild 1.
- ▷ **Image2** (input_object) . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real
  Eingabebild 2.
- ▷ **ImageAbsDiff** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real
  Betrag der Differenz der Eingabebilder.
- ▷ **Mult** (input_control) . . . . . number ⇝ *real* / integer
  Skalierungsfaktor.
  Defaultwert: 1.0
  Wertevorschläge: Mult ∈ {1.0, 2.0, 3.0, 4.0}
  Restriktion: Mult > 0

Ergebnis

abs_diff_image liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`threshold`

Alternativen

`sub_image`

Siehe auch

`add_image`, `scale_image`, `dyn_threshold`

Modul

Foundation

abs_image (Image : ImageAbs : :)

Berechnen des Absolutbetrags eines Bildes.

Der Operator `abs_image` berechnet für Bilder eines beliebigen Typs den Absolutbetrag der Grauwerte und legt das Ergebnis in `ImageAbs` ab. Bei komplexen Bildern wird das Powerspektrum als 'real'-Bild berechnet. `abs_image` erzeugt bei Bildern ohne Vorzeichen eine logische Kopie.

`abs_image` kann für int1-, int2-, int4-, real-, und complex-Bilder auf OpenCL-Geräten ausgeführt werden. Da OpenCL 1.0 aber nur einfache genaue Fließkommaunterstützung voraussetzt und nicht alle Rundungsarten unterstützt werden, kann das Ergebnis der OpenCL-Implementierung geringfügig von dem der skalaren oder der SIMD-Implementierung abweichen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* int1 / int2 / int4 / int8 / real / complex Eingabebild(er).
- ▷ **ImageAbs** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* int1 / int2 / int4 / int8 / real Ergebnisbild(er).

Beispiel

```
convert_image_type (Image, ImageInt2, 'int2')
texture_laws (ImageInt2, ImageTexture, 'el', 2, 5)
abs_image (ImageTexture, ImageTexture)
```

Ergebnis

`abs_image` liefert den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

`convert_image_type`, `power_byte`

Modul

Foundation

acos_image (Image : ArccosImage : :)

Berechnen des Arkuskosinus eines Bildes.

acos_image berechnet den Arkuskosinus des Eingabebildes Image und liefert das Ergebnis in ArccosImage zurück. Die Winkel in ArccosImage werden im Bogenmaß abgespeichert. Falls Image Grauwerte außerhalb des Definitionsbereiches $[-1, 1]$ der Arkuskosinusfunktion enthält, werden die entsprechenden Grauwerte in ArccosImage auf 0 gesetzt.

Achtung

acos_image kann auf OpenCL-Geräten ausgeführt werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* real
  Eingabebild.
- ▷ **ArccosImage** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* real
  Ergebnisbild.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

asin_image, atan_image, atan2_image, tan_image, sin_image, cos_image

Modul

Foundation

add_image (Image1, Image2 : ImageResult : Mult, Add :)

Addieren zweier Bilder.

add_image addiert zwei Bilder. Die Grauwerte $(g1, g2)$ der Eingabebilder (Image1 und Image2) werden dabei wie folgt transformiert:

$$g' := (g1 + g2) * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten. Dies gilt nicht im Fall von cyclic- und direction-Bildern. Das Ergebnisbild wird in ImageResult abgelegt.

Es ist möglich, byte-Bilder mit int2, uint2 oder int4 Bildern bzw. int4 mit int2 oder uint2 Bildern zu addieren. In diesem Fall werden int2, uint2 oder int4 bzw. int4 Bilder als Ergebnis erzeugt.

Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Es ist zu beachten, dass die Laufzeit des Operators von der Wahl der Steuerparameter abhängt. Für häufig verwendete Parameterkombinationen sind spezielle Optimierungen implementiert. Für byte, int2. uint2 und int4 Bilder sind außerdem spezielle Optimierungen unter Verwendung von SIMD Technologie implementiert. Ob diese tatsächlich zur Anwendung kommen, kann über den Systemparameter *'mmx_enable'* (siehe set_system) gesteuert werden. Falls *'mmx_enable'* auf *'true'* gesetzt wird (und die verwendete Hardware SIMD Technologie unterstützt), wird die Berechnung intern unter Verwendung von SIMD Technologie durchgeführt.

add_image kann für byte-, int1-, int2-, uint2-, int4-, real-, direction-, cyclic-, und complex-Bilder auf OpenCL-Geräten ausgeführt werden. Da OpenCL 1.0 aber nur einfache genaue Fließkommaunterstützung voraussetzt und

nicht alle Rundungsarten unterstützt werden, kann das Ergebnis der OpenCL-Implementierung geringfügig von dem der skalaren oder der SIMD-Implementierung abweichen.

Achtung

Es ist zu beachten, dass der Beschleunigungseffekt von SIMD Technologie bei großen, kompakten Eingaberegionen am größten ist. Im Einzelfall kann aber die Ausführung von `add_image`, abhängig von der Eingaberegion und den Hardwareeigenschaften, mit SIMD Technologie länger dauern als ohne. In diesem Falle lässt sich die Verwendung von SIMD Technologie durch `set_system(::'mmx_enable','false':)` verhindern.

Parameter

- ▷ **`Image1`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex
  Bild(er) 1.
- ▷ **`Image2`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex
  Bild(er) 2.
- ▷ **`ImageResult`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex
  Ergebnisbil(der) durch die Bildpunktaddition.
- ▷ **`Mult`** (input_control) ..... number ⇝ *real* / integer
  Faktor zur Grauwertanpassung.
  Defaultwert: 0.5
  Wertevorschläge: `Mult` ∈ {0.2, 0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 5.0}
  Typischer Wertebereich: -255.0 ≤ `Mult` ≤ 255.0
  Minimale Schrittweite: 0.001
  Empfohlene Schrittweite: 0.1
- ▷ **`Add`** (input_control) ..... number ⇝ *real* / integer
  Wert für Grauwertausgleich.
  Defaultwert: 0
  Wertevorschläge: `Add` ∈ {0, 64, 128, 255, 512}
  Typischer Wertebereich: -512.0 ≤ `Add` ≤ 512.0
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 1.0

Beispiel

```
read_image(Image1,'fabrik')
dev_display (Image1)
read_image(Image2,'monkey')
dev_display (Image2)
add_image(Image1,Image2,Result,0.5,10.0)
dev_display (Result)
```

Ergebnis

`add_image` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

sub_image, mult_image

Siehe auch

sub_image, mult_image

Modul

Foundation

asin_image (Image : ArcsinImage : :)

Berechnen des Arkussinus eines Bildes.

asin_image berechnet den Arkussinus des Eingabebildes Image und liefert das Ergebnis in ArcsinImage zurück. Die Winkel in ArcsinImage werden im Bogenmaß abgespeichert. Falls Image Grauwerte außerhalb des Definitionsbereiches $[-1, 1]$ der Arkussinusfunktion enthält, werden die entsprechenden Grauwerte in ArcsinImage auf 0 gesetzt.

Achtung

asin_image kann auf OpenCL-Geräten ausgeführt werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* real
  Eingabebild.
- ▷ **ArcsinImage** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* real
  Ergebnisbild.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

acos_image, atan_image, atan2_image, tan_image, sin_image, cos_image

Modul

Foundation

atan2_image (ImageY, ImageX : ArctanImage : :)

Berechnen des Arkustangens zweier Bilder.

atan2_image berechnet den Arkustangens ImageY/ImageX der Eingabebilder ImageY und ImageX, wobei das Vorzeichen der Grauwerte berücksichtigt wird, um den Quadranten des Ergebnisses zu bestimmen, und liefert das Ergebnis in ArctanImage zurück. Die Winkel in ArctanImage werden im Bogenmaß abgespeichert.

Achtung

atan2_image kann für int1-, int2-, int4-, und real-Bilder auf OpenCL-Geräten ausgeführt werden. Da Ergebnis kann von dem auf der CPU errechneten abweichen.

Parameter

- ▷ **ImageY** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* int1 / int2 / int4 / int8 / real
  Eingabebild 1.
- ▷ **ImageX** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* int1 / int2 / int4 / int8 / real
  Eingabebild 2.

▷ **ArctanImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* real
Ergebnisbild.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

atan_image

Siehe auch

acos_image, asin_image, tan_image, sin_image, cos_image

Modul

Foundation

atan_image (Image : ArctanImage : :)

Berechnen des Arkustangens eines Bildes.

atan_image berechnet den Arkustangens des Eingabebildes Image und liefert das Ergebnis in ArctanImage zurück. Die Winkel in ArctanImage werden im Bogenmaß abgespeichert.

Achtung

atan_image kann auf OpenCL-Geräten ausgeführt werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* real
Eingabebild.

▷ **ArctanImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* real
Ergebnisbild.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

atan2_image

Siehe auch

acos_image, asin_image, tan_image, sin_image, cos_image

Modul

Foundation

cos_image (Image : CosImage : :)

Berechnen des Kosinus eines Bildes.

cos_image berechnet den Kosinus des Eingabebildes Image und liefert das Ergebnis vom Pixeltyp real in CosImage zurück. Falls Image vom Pixeltyp direction oder cyclic ist, werden die in Image gespeicherten Winkel intern ins Bogenmaß umgerechnet. Falls Image vom Pixeltyp real ist, wird angenommen, dass die Winkel in Image im Bogenmaß vorliegen. Für Bilder vom Pixeltyp direction führt ein Wert von 255 im Eingabebild zu einem Wert von 0 im Ausgabebild.

Achtung

cos_image kann auf OpenCL-Geräten ausgeführt werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* direction / cyclic / real
  Eingabebild.
- ▷ **CosImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* real
  Ergebnisbild.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

sin_image, tan_image, asin_image, acos_image, atan_image, atan2_image

Modul

Foundation

div_image (Image1, Image2 : ImageResult : Mult, Add :)

Dividieren zweier Bilder.

div_image dividiert zwei Bilder. Die Grauwerte $(g1, g2)$ der Eingabebilder (Image1) werden dabei wie folgt transformiert:

$$g' := g1/g2 * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten. Wird durch 0 dividiert, wird das Ergebnis auf 0 gesetzt.

Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

div_image kann für byte-, int1-, int2-, uint2-, int4-, real-, und complex-Bilder auf OpenCL-Geräten ausgeführt werden. Da OpenCL 1.0 aber nur einfache genaue Fließkommaunterstützung voraussetzt und nicht alle Rundungsarten unterstützt werden, kann das Ergebnis der OpenCL-Implementierung geringfügig von dem der skalaren Implementierung abweichen.

Parameter

- ▷ **Image1** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / complex
  Bild(er) 1.
- ▷ **Image2** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / complex
  Bild(er) 2.
- ▷ **ImageResult** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / complex
  Ergebnisbild(er) durch die Division.

▷ **Mult** (input_control) . . . . . number ⇝ *real* / integer
Faktor für Graubereichsanpassung.
Defaultwert: 255
Wertevorschläge: `Mult` ∈ {0.1, 0.2, 0.5, 1.0, 2.0, 3.0, 10, 100, 500, 1000}
Typischer Wertebereich: -1000 ≤ `Mult` ≤ 1000
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 1

▷ **Add** (input_control) . . . . . number ⇝ *real* / integer
Wert für Graubereichsanpassung.
Defaultwert: 0
Wertevorschläge: `Add` ∈ {0.0, 128.0, 256.0, 1025}
Typischer Wertebereich: -1000 ≤ `Add` ≤ 1000
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 1.0

Beispiel

```
read_image(Image1,'fabrik')
dev_display (Image1)
read_image(Image2,'monkey')
dev_display (Image2)
div_image (Image1, Image2, Result, 200, 10)
dev_display (Result)
```

Ergebnis

`div_image` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`add_image`, `sub_image`, `mult_image`

Siehe auch

`add_image`, `sub_image`, `mult_image`

Modul

Foundation

exp_image (Image : ExpImage : Base :)

Berechnen der Potenzierung eines Bildes.

`exp_image` berechnet die Potenzierung des Eingabebildes `Image` zur Basis `Base` und liefert das Ergebnis in `ExpImage` zurück. Falls `Image` Grauwerte enthält, für die das Ergebnis der Potenzierung den Wertebereich von `ExpImage` überschreiten würde (z.B. > 88.722839 für `Base` $= 'e'$), werden die entsprechenden Grauwerte in `ExpImage` auf den größten repräsentierbaren Wert (d.h. 3.4028235×10^{38}) gesetzt.

Achtung

`exp_image` kann für byte-, int1-, int2-, uint2-, int4-, und real-Bilder auf OpenCL-Geräten ausgeführt werden. Da Ergebnis kann von dem auf der CPU errechneten abweichen.

Parameter

▷ **Image** (input_object) .... (multichannel-)image(-array) ⇝ *object :* byte / int1 / uint2 / int2 / int4 / int8 / real
Eingabebild.

▷ **ExpImage** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* real
Ergebnisbild.

▷ **Base** (input_control) ..... number ⇝ *string* / integer / real
Basis der Potenzierung.
Defaultwert: 'e'
Wertevorschläge: `Base` ∈ {'e', 2, 10}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

`pow_image`, `log_image`

Modul

Foundation

gamma_image (Image : GammaImage : Gamma, Offset, Threshold, MaxGray, Encode :)

Berechnen der Gamma-Kodierung oder Gamma-Dekodierung eines Bildes.

`gamma_image` führt eine allgemeine Gamma-Kodierung oder Gamma-Dekodierung des Eingabebildes `Image` durch und liefert das Ergebnisbild in `GammaImage` zurück. Eine allgemeine Gamma-Kodierung kann wie folgt beschrieben werden:

$$G' = \begin{cases} m\left((1+o)(G/m)^{\gamma} - o\right), & t < G/m \leq 1 \\ sG, & 0 \leq G/m \leq t \end{cases}$$

Hierbei ist G' der Gamma-kodierte Grauwert, G der lineare Grauwert, γ = `Gamma`, o = `Offset`, m = `MaxGray`, t = `Threshold` und s ein Faktor, der aus `Gamma`, `Offset`, `Threshold` und `MaxGray` so berechnet wird, dass der lineare und exponentielle Teil der Transformation stetig sind.

In analoger Weise kann eine allgemeine Gamma-Dekodierung wie folgt beschrieben werden:

$$G = \begin{cases} m\left(\frac{G'/m+o}{1+o}\right)^{1/\gamma}, & t' < G'/m \leq 1 \\ G'/s, & 0 \leq G'/m \leq t' \end{cases}$$

Hierbei haben die Variablen dieselbe Bedeutung wie bei der Gamma-Kodierung und es gilt $t' = st$.

So kann zum Beispiel die Gamma-Kodierung oder Gamma-Dekodierung, die von der sRGB-Norm gefordert wird, durch `Gamma` = $1.0/2.4$, `Offset` = 0.055 und `Threshold` = 0.0031308 erhalten werden, während die Gamma-Kodierung oder Gamma-Dekodierung, die von der HDTV-Videonorm gefordert wird, durch `Gamma` = 0.45, `Offset` = 0.099 und `Threshold` = 0.018 erhalten werden kann. In jedem Fall sollte `MaxGray` auf einen Wert gesetzt werden, der zum Bildtyp von `Image` passt (z.B. 255.0 für Byte-Bilder).

Achtung

`gamma_image` kann für byte-, uint2-, und real-Bilder auf OpenCL Geräten ausgeführt werden. Da die Operation in einfach statt wie auf der CPU doppelt genauer Fließkommadarstellung berechnet wird, kann das Ergebnis der OpenCL-Implementierung geringfügig von dem der CPU-Implementierung abweichen.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **GammaImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **Gamma** (input_control) . . . . . real ⇝ *real*
  Gamma-Koeffizient des exponentiellen Teils der Transformation.
  Defaultwert: 0.416666666667
  Wertevorschläge: `Gamma` ∈ {0.416666666667, 0.45}
- ▷ **Offset** (input_control) . . . . . real ⇝ *real*
  Versatz des exponentiellen Teils der Transformation.
  Defaultwert: 0.055
  Wertevorschläge: `Offset` ∈ {0.055, 0.099}
- ▷ **Threshold** (input_control) . . . . . real ⇝ *real*
  Grauwert, für den die Transformation von linear nach exponentiell wechselt.
  Defaultwert: 0.0031308
  Wertevorschläge: `Threshold` ∈ {0.0031308, 0.018}
- ▷ **MaxGray** (input_control) . . . . . number ⇝ *real* / integer
  Maximaler Grauwert des Eingabebildtyps.
  Defaultwert: 255.0
  Wertevorschläge: `MaxGray` ∈ {1.0, 255.0, 511.0, 1023.0, 4095.0, 16383.0, 65535.0}
- ▷ **Encode** (input_control) . . . . . string ⇝ *string*
  Berechnung einer Gamma-Kodierung, falls *'true'*, ansonsten einer Gamma-Dekodierung.
  Defaultwert: 'true'
  Werteliste: `Encode` ∈ {'true', 'false'}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`pow_image`

Siehe auch

`sqrt_image`, `exp_image`, `log_image`

Literatur

Erik Reinhard, Erum Arif Khan, Ahmet Oguz Akyüz, Garret M. Johnson: „Color Imaging: Fundamentals and Applications." A K Peters, Ltd., 2008.

CEI/IEC 61966-2-1:1999: „Colour management – Default RGB colour space – sRGB."

ITU-R BT.709-5: „Parameter values for the HDTV standards for production and international programme exchange."

Modul

Foundation

`invert_image ( Image : ImageInvert : : )`

Invertieren eines Bildes.

`invert_image` invertiert die Grauwerte eines Bildes. Bei Bildern vom Typ 'byte' und 'cyclic' berechnet sich das Ergebnis als:

$$g' = 255 - g$$

Bilder vom Typ 'direction' werden durch

$$g' = (g + 90) \text{ modulo } 180$$

transformiert. Bei Typen mit Vorzeichen werden die Werte negiert. Das Ergebnisbild hat den gleichen Pixeltyp wie das Eingabebild.

Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

`invert_image` kann für byte-, direction-, cyclic-, int1-, int2-, uint2-, int4-, und real-Bilder auf OpenCL-Geräten ausgeführt werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real

Eingabebild(er).

▷ **ImageInvert** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real

Bild(er) mit invertierten Grauwerten.

Beispiel

```
read_image(Orig,'fabrik')
invert_image(Orig,Invert)
dev_display(Invert)
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

watersheds

Alternativen

scale_image

Siehe auch

scale_image, add_image, sub_image

Modul

Foundation

log_image (Image : LogImage : Base :)

Berechnen des Logarithmus eines Bildes.

`log_image` berechnet den Logarithmus des Eingabebildes `Image` zur Basis `Base` und liefert das Ergebnis in `LogImage` zurück. Falls `Image` Grauwerte außerhalb des Definitionsbereiches der Logarithmusfunktion enthält (also ≤ 0), werden die entsprechenden Grauwerte in `LogImage` auf 0 gesetzt.

Achtung

`log_image` kann für byte-, int1-, int2-, uint2-, int4-, und real-Bilder auf OpenCL-Geräten ausgeführt werden. Da Ergebnis kann von dem auf der CPU errechneten abweichen.

Parameter

▷ **Image** (input_object) . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / uint2 / int2 / int4 / int8 / real
Eingabebild.

▷ **LogImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* real
Ergebnisbild.

▷ **Base** (input_control) . . . . . number ⇝ *string* / integer / real
Basis des Logarithmus.
Defaultwert: 'e'
Wertevorschläge: `Base` ∈ {'e', 2, 10}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

`pow_image`, `exp_image`

Modul

Foundation

max_image (Image1, Image2 : ImageMax : :)

Pixelweises Berechnen des Maximums von zwei Bildern.

`max_image` ermittelt (pixelweise) das Maximum der Bilder `Image1` und `Image2`. Das Ergebnis wird in dem Bild `ImageMax` abgelegt. Das Ergebnisbild hat den gleichen Pixeltyp wie das Eingabebild. Sollen mehrere Bilder(paare) bei einem Aufruf bearbeitet werden, so wird jeweils das i-te Bild aus `Image1` mit dem i-ten Bild aus `Image2` verglichen. Die Anzahl der Bilder in beiden Eingabeparametern muss also gleich sein. Zu jedem Eingabepaar wird ein Ausgabebild erzeugt.

`max_image` kann für byte-, int1-, int2-, uint2-, int4-, real-, direction-, und cyclic-Bilder auf OpenCL-Geräten ausgeführt werden.

Achtung

Die beiden Eingabebilder müssen den gleichen Typ und die gleiche Größe haben.

Parameter

▷ **Image1** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic
Bild(er) 1.

▷ **Image2** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic
Bild(er) 2.

▷ **ImageMax** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic
Ergebnisbild(er) durch die Maximierung.

Beispiel

```
read_image(Image1,'monkey')
read_image(Image2,'fabrik')
max_image(Image1,Image2,Max)
dev_display (Max)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `max_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabbilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

`min_image`

Modul

Foundation

min_image (Image1, Image2 : ImageMin : :)

Pixelweises Berechnen des Minimums von zwei Bildern.

`min_image` ermittelt (pixelweise) das Minimum der Bilder `Image1` und `Image2`. Das Ergebnis wird in dem Bild `ImageMin` abgelegt. Das Ergebnisbild hat den gleichen Pixeltyp wie das Eingabebild. Sollen mehrere Bilder(paare) bei einem Aufruf bearbeitet werden, so wird jeweils das i-te Bild aus `Image1` mit dem i-ten Bild aus `Image2` verglichen. Die Anzahl der Bilder in beiden Eingabeparametern muss also gleich sein. Zu jedem Eingabepaar wird ein Ausgabebild erzeugt.

`min_image` kann für byte-, int1-, int2-, uint2-, int4-, real-, direction-, und cyclic-Bilder auf OpenCL-Geräten ausgeführt werden.

Parameter

▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic

Bild(er) 1.

▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic

Bild(er) 2.

▷ **ImageMin** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic

Ergebnisbild(er) durch das Minimierung.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `min_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

gray_erosion

Siehe auch

max_image

Modul

Foundation

mult_image (Image1, Image2 : ImageResult : Mult, Add :)

Multiplizieren zweier Bilder.

mult_image multipliziert zwei Bilder. Die Grauwerte $(g1, g2)$ der Eingabebilder (Image1) werden dabei wie folgt transformiert:

$$g' := g1 * g2 * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten. Dies gilt nicht im Fall von cyclic- und direction-Bildern.

Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

mult_image kann für byte-, int1-, int2-, uint2-, int4-, real-, direction-, cyclic-, und complex-Bildern auf OpenCL-Geräten ausgeführt werden. Da OpenCL 1.0 aber nur einfache genaue Fließkommaunterstützung voraussetzt und nicht alle Rundungsarten unterstützt werden, kann das Ergebnis der OpenCL-Implementierung geringfügig von dem der skalaren Implementierung abweichen.

Parameter

▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex

Bild(er) 1.

▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex

Bild(er) 2.

▷ **ImageResult** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex

Ergebnisbild(er) durch das Produkt.

▷ **Mult** (input_control) ..... number ⇝ *real* / integer

Faktor für Grauwertbereichsanpassung.
Defaultwert: 0.005
Wertevorschläge: Mult ∈ {0.001, 0.01, 0.5, 1.0, 2.0, 3.0, 5.0, 10.0}
Typischer Wertebereich: -255.0 ≤ Mult ≤ 255.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1

▷ **Add** (input_control) ..... number ⇝ *real* / integer

Wert für Grauwertbereichsanpassung.
Defaultwert: 0
Wertevorschläge: Add ∈ {0.0, 128.0, 256.0}
Typischer Wertebereich: -512.0 ≤ Add ≤ 512.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 1.0

Beispiel

```
read_image(Image1,'fabrik')
dev_display (Image1)
read_image(Image2,'monkey')
```

```
dev_display (Image2)
mult_image (Image1, Image2, Result, 0.01, 10)
dev_display (Result)
```

Ergebnis

`mult_image` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`add_image`, `sub_image`, `div_image`

Siehe auch

`add_image`, `sub_image`, `div_image`

Modul

Foundation

pow_image (Image : PowImage : Exponent :)

Berechnen der Potenzierung eines Bildes.

`pow_image` berechnet die Potenzierung des Eingabebildes `Image` mit dem Exponenten `Exponent` und liefert das Ergebnis in `PowImage` zurück. Falls `Image` Grauwerte enthält, für die das Ergebnis der Potenzierung den Wertebereich von `PowImage` überschreiten würde (z.B. > 7131.55017 für `Exponent` $= 10$), werden die entsprechenden Grauwerte in `PowImage` auf den größten repräsentierbaren Wert (d.h. 3.4028235×10^{38}) gesetzt. Falls `Image` Grauwerte enthält, für die die Potenzierung nicht durchgeführt werden kann (d.h. falls die Grauwerte negativ sind und `Exponent` keine ganze Zahl ist), werden die entsprechenden Grauwerte in `PowImage` auf 0 gesetzt.

Achtung

`pow_image` kann für byte-, int1-, int2-, uint2-, int4-, und real-Bilder auf OpenCL-Geräten ausgeführt werden. Da Ergebnis kann von dem auf der CPU errechneten abweichen.

Parameter

▷ **Image** (input_object) . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / uint2 / int2 / int4 / int8 / real
Eingabebild.

▷ **PowImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* real
Ergebnisbild.

▷ **Exponent** (input_control) . . . . . number ⇝ *real* / integer
Exponent der Potenzierung.
Defaultwert: 2
Wertevorschläge: `Exponent` ∈ {0.25, 0.5, 2, 3, 4}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

gamma_image

Siehe auch

sqrt_image, exp_image, log_image

Modul

Foundation

scale_image (Image : ImageScaled : Mult, Add :)

Skalieren von Grauwerten.

scale_image skaliert die Eingabebilder (Image) mittels folgender Transformation:

$$g' := g * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten. Dies gilt nicht im Fall von cyclic- und direction-Bildern.

Dieser Operator kann z.B. dafür verwendet werden, die Grauwerte eines Bildes, also das Intervall [GMin,GMax], auf das volle Intervall [0:255] abzubilden, indem man die Parameter wie folgt wählt:

$$\texttt{Mult} = \frac{255}{\text{GMax} - \text{GMin}} \qquad \texttt{Add} = -\texttt{Mult} * \text{GMin}$$

Min und GMax können z.B. mit dem Operator min_max_gray bestimmt werden.

Es ist zu beachten, dass die Laufzeit des Operators von der Wahl der Steuerparameter abhängt. Für häufig verwendete Parameterkombinationen sind spezielle Optimierungen implementiert. Weiterhin sind für int2 und uint2 Bilder spezielle Optimierungen mit Festkommaarithmetik und für byte, int2 und uint2 Bilder spezielle Optimierungen unter Verwendung von SIMD Technologie implementiert. Ob diese tatsächlich zur Anwendung kommen, kann über die Systemparameter *'int_zooming'* und *'mmx_enable'* (siehe set_system) gesteuert werden. Falls *'int_zooming'* auf *'true'* gesetzt wird, wird die intern Berechnung mit Festkommaarithmetik durchgeführt, was zu wesentlich kürzeren Laufzeiten führt. Allerdings ist hier die Genauigkeit der berechneten Grauwerte geringer. Die Abweichung zur genaueren Berechnung (mit *'int_zooming'* = *'false'*) ist normalerweise kleiner als zwei Grauwerte. Falls *'mmx_enable'* auf *'true'* gesetzt wird (und die verwendete Hardware SIMD Technologie unterstützt), wird die Berechnung intern unter Verwendung von SIMD Technologie durchgeführt. In diesem Fall wird der Wert von *'int_zooming'* ignoriert.

scale_image kann für byte-, int1-, int2-, uint2-, int4-, real-, direction-, cyclic-, und complex-Bilder auf OpenCL-Geräten ausgeführt werden. Da OpenCL 1.0 aber nur einfache genaue Fließkommaunterstützung voraussetzt und nicht alle Rundungsarten unterstützt werden, kann das Ergebnis der OpenCL-Implementierung geringfügig von dem der skalaren oder der SIMD-Implementierung abweichen.

Achtung

Es ist zu beachten, dass der Beschleunigungseffekt von SIMD Technologie bei großen, kompakten Eingaberegionen am größten ist. Im Einzelfall kann aber die Ausführung von scale_image, abhängig von der Eingaberegion und den Hardwareeigenschaften, mit SIMD Technologie länger dauern als ohne. In diesem Falle lässt sich die Verwendung von SIMD Technologie durch set_system(::'mmx_enable','false':) verhindern.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex

Bild(er), dessen/deren Grauwerte skaliert werden sollen.

▷ **ImageScaled** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex

Ergebnisbild(er) durch die Skalierung

▷ **Mult** (input_control) . . . . . number ⇝ *real* / integer
Grauwertskalierung.
Defaultwert: 0.01
Wertevorschläge: Mult ∈ {0.001, 0.003, 0.005, 0.008, 0.01, 0.02, 0.03, 0.05, 0.08, 0.1, 0.5, 1.0}
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1

▷ **Add** (input_control) . . . . . number ⇝ *real* / integer
Grauwertverschiebung.
Defaultwert: 0
Wertevorschläge: Add ∈ {0, 10, 50, 100, 200, 500}
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 1.0

Beispiel

```
* Complement of the gray values:
scale_image(Image,Invert,-1.0,255.0)
```

Ergebnis

scale_image liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

min_max_gray

Alternativen

mult_image, add_image, sub_image

Siehe auch

min_max_gray

Modul

Foundation

sin_image (Image : SinImage : :)

Berechnen des Sinus eines Bildes.

sin_image berechnet den Sinus des Eingabebildes Image und liefert das Ergebnis vom Pixeltyp real in SinImage zurück. Falls Image vom Pixeltyp direction oder cyclic ist, werden die in Image gespeicherten Winkel intern ins Bogenmaß umgerechnet. Falls Image vom Pixeltyp real ist, wird angenommen, dass die Winkel in Image im Bogenmaß vorliegen. Für Bilder vom Pixeltyp direction führt ein Wert von 255 im Eingabebild zu einem Wert von 0 im Ausgabebild.

Achtung

sin_image kann auf OpenCL-Geräten ausgeführt werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* direction / cyclic / real
Eingabebild.

▷ **SinImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* real
Ergebnisbild.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

`cos_image`, `tan_image`, `asin_image`, `acos_image`, `atan_image`, `atan2_image`

Modul

Foundation

sqrt_image (Image : SqrtImage : :)

Berechnen der Wurzel eines Bildes.

`sqrt_image` berechnet die Wurzel des Eingabebildes `Image` punktweise und liefert das Ergebnis vom selben Pixeltyp in `SqrtImage` zurück. Besitzt der Pixeltyp von `Image` ein Vorzeichen, so werden negative Pixelwerte auf 0 in `SqrtImage` abgebildet.

`sqrt_image` kann für byte-, int1-, int2-, uint2-, int4 und real-Bilder auf OpenCL-Geräten ausgeführt werden.

Parameter

▷ **Image** (input_object) . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real
Eingabebild

▷ **SqrtImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real
Ergebnisbild

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

`pow_image`

Modul

Foundation

sub_image (ImageMinuend, ImageSubtrahend : ImageSub : Mult, Add :)

Subtrahieren zweier Bilder.

sub_image subtrahiert zwei Bilder. Die Grauwerte $(g1, g2)$ der Eingabebilder (ImageMinuend und ImageSubtrahend) werden dabei wie folgt transformiert:

$$g' := (g1 - g2) * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten. Dies gilt nicht im Fall von cyclic- und direction-Bildern.

Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Es ist zu beachten, dass die Laufzeit des Operators von der Wahl der Steuerparameter abhängt. Für häufig verwendete Parameterkombinationen sind spezielle Optimierungen implementiert. Für byte, int2 und uint2 Bilder sind außerdem spezielle Optimierungen unter Verwendung von SIMD Technologie implementiert. Ob diese tatsächlich zur Anwendung kommen, kann über den Systemparameter *'mmx_enable'* (siehe set_system) gesteuert werden. Falls *'mmx_enable'* auf *'true'* gesetzt wird (und die verwendete Hardware SIMD Technologie unterstützt), wird die Berechnung intern unter Verwendung von SIMD Technologie durchgeführt.

sub_image kann für byte-, int1-, int2-, uint2-, int4-, real-, direction-, cyclic-, und complex-Bilder auf OpenCL-Geräten ausgeführt werden. Da OpenCL 1.0 aber nur einfache genaue Fließkommaunterstützung voraussetzt und nicht alle Rundungsarten unterstützt werden, kann das Ergebnis der OpenCL-Implementierung geringfügig von dem der skalaren oder der SIMD-Implementierung abweichen.

Achtung

Es ist zu beachten, dass der Beschleunigungseffekt von SIMD Technologie bei großen, kompakten Eingaberegionen am größten ist. Im Einzelfall kann aber die Ausführung von sub_image, abhängig von der Eingaberegion und den Hardwareeigenschaften, mit SIMD Technologie länger dauern als ohne. In diesem Falle lässt sich die Verwendung von SIMD Technologie durch set_system(::'mmx_enable','false':) verhindern.

Parameter

▷ **ImageMinuend** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex

Minuend(en).

▷ **ImageSubtrahend** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex

Subtrahend(en).

▷ **ImageSub** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / int8 / real / direction / cyclic / complex

Ergebnisbild(er) durch die Subtraktion.

▷ **Mult** (input_control) ..... number ⇝ *real* / integer

Korrekturfaktor.
Defaultwert: 1.0
Wertevorschläge: Mult ∈ {0.0, 1.0, 2.0, 3.0, 4.0}
Typischer Wertebereich: -255.0 ≤ Mult ≤ 255.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1

▷ **Add** (input_control) ..... number ⇝ *real* / integer

Korrekturwert.
Defaultwert: 128.0
Wertevorschläge: Add ∈ {0.0, 128.0, 256.0}
Typischer Wertebereich: -512.0 ≤ Add ≤ 512.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 1.0

Beispiel

```
read_image(Image1,'fabrik')
dev_display (Image1)
read_image(Image2,'monkey')
```

```
dev_display (Image2)
sub_image (Image1, Image2, Result, 1, 100)
dev_display (Result)
```

Ergebnis

`sub_image` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`dual_threshold`

Alternativen

`mult_image`, `add_image`

Siehe auch

`add_image`, `mult_image`, `dyn_threshold`, `check_difference`

Modul

Foundation

tan_image (Image : TanImage : :)

Berechnen des Tangens eines Bildes.

`tan_image` berechnet den Tangens des Eingabebildes `Image` und liefert das Ergebnis vom Pixeltyp real in `TanImage` zurück. Falls `Image` vom Pixeltyp direction oder cyclic ist, werden die in `Image` gespeicherten Winkel intern ins Bogenmaß umgerechnet. Falls `Image` vom Pixeltyp real ist, wird angenommen, dass die Winkel in `Image` im Bogenmaß vorliegen. Für Bilder vom Pixeltyp direction führt ein Wert von 255 im Eingabebild zu einem Wert von 0 im Ausgabebild.

Achtung

`tan_image` kann auf OpenCL-Geräten ausgeführt werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* direction / cyclic / real
Eingabebild.

▷ **TanImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* real
Ergebnisbild.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

sin_image, cos_image, asin_image, acos_image, atan_image, atan2_image

Modul

Foundation

10.2 Bit

bit_and (Image1, Image2 : ImageAnd : :)

Bitweises AND aller Pixel der Eingabebilder.

bit_and berechnet das bitweise „and“ aller Pixel der Eingabebilder. Die Semantik der „and“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, unsigned short, int/long). Die Bilder müssen die gleiche Bildgröße und den gleichen Pixeltyp haben. Es werden die Bildpunkte innerhalb des Definitionsbereiches des Bildes im ersten Parameter bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

- ▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

  Eingabebild(er) 1.
- ▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

  Eingabebild(er) 2.
- ▷ **ImageAnd** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

  Ergebnis(se) der AND-Operation.

Beispiel

```
read_image(Image1,'fabrik')
dev_display (Image1)
read_image(Image2,'monkey')
dev_display (Image2)
bit_and(Image1,Image2,ImageBitAnd)
dev_display (ImageBitAnd)
```

Ergebnis

Sind die Bilder korrekt (Typ und Anzahl), dann liefert bit_and den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

bit_mask, add_image, max_image

Siehe auch

bit_mask, add_image, max_image

Modul

Foundation

bit_lshift (Image : ImageLShift : Shift :)

Links-Shift aller Pixel des Bildes.

bit_lshift berechnet einen bitweisen „links-Shift“ aller Pixel des Eingabebildes. Die Semantik der „links-Shift“ Operation entspricht der von C („«“) für die jeweiligen Typen (signed char, unsigned char, short, unsigned short, int/long). Tritt ein Überlauf auf, so wird das Ergebnis auf den maximalen Wert des jeweiligen Pixeltyps begrenzt. Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Eingabebild(er).

▷ **ImageLShift** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Ergebnis(se) der Shift-Operation.

▷ **Shift** (input_control) ..... integer ⇝ *integer*
Shift-Wert.
Defaultwert: 3
Wertevorschläge: Shift ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 31}
Typischer Wertebereich: 0 ≤ Shift ≤ 31
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Shift >= 1 && Shift <= 31

Beispiel

```
read_image(&ByteImage,"fabrik");
convert_image_type(ByteImage,&Int2Image,"int2");
bit_lshift(Int2Image,&FullInt2Image,8);
```

Ergebnis

Sind die Bilder korrekt (Typ) und hat Shift eine gültigen Wert, dann liefert bit_lshift den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

scale_image

Siehe auch

bit_rshift

Modul

Foundation

bit_mask (Image : ImageMask : BitMask :)

Logisches „AND“ jedes Pixels mit einer Bitmaske.

bit_mask führt eine „and“ Operation von jedem Pixel mit einer festen Maske durch. Die Semantik der „and“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, unsigned short, int/long). Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Eingabebild(er).

▷ **ImageMask** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Ergebnis(se) der Kombination mit Mask.

▷ **BitMask** (input_control) ..... integer ⇝ *integer*
Bitfeld.
Defaultwert: 128
Wertevorschläge: BitMask ∈ {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096}

Ergebnis

Sind die Bilder korrekt (Typ), dann liefert bit_mask den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

threshold, bit_or

Alternativen

bit_slice

Siehe auch

bit_and, bit_lshift

Modul

Foundation

bit_not (Image : ImageNot : :)

Komplementieren aller Bits der Pixel.

bit_not berechnet das bitweise „complement“ aller Pixel des Eingabebildes. Die Semantik der „complement“ Operation entspricht der von C („∼“) für die jeweiligen Typen (signed char, unsigned char, short, unsigned short, int/long). Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Eingabebild(er).

▷ **ImageNot** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Ergebnis(se) der Komplement-Operation.

Beispiel

```
read_image(Image,'monkey')
dev_display(Image)
bit_not(Image,ImageBitNot)
dev_display(ImageBitNot)
```

Ergebnis

Sind die Bilder korrekt (Typ), dann liefert `bit_not` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`bit_or`, `bit_and`, `add_image`

Siehe auch

`bit_slice`, `bit_mask`

Modul

Foundation

bit_or (Image1, Image2 : ImageOr : :)

Bitweises OR aller Pixel der Eingabebilder.

`bit_or` berechnet das bitweise „or“ aller Pixel der Eingabebilder. Die Semantik der „or“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, unsigned short, int/long). Die Bilder müssen die gleiche Bildgröße und den gleichen Pixeltyp haben. Es werden die Bildpunkte innerhalb des Definitionsbereiches des Bildes im ersten Parameter bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

Eingabebild(er) 1.

▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

Eingabebild(er) 2.

▷ **ImageOr** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

Ergebnis(se) der OR-Operation.

Beispiel

```
read_image(Image1,'fabrik')
dev_display (Image1)
read_image(Image2,'monkey')
dev_display (Image2)
bit_or(Image1,Image2,ImageBitOr)
dev_display (ImageBitOr)
```

Ergebnis

Sind die Bilder korrekt (Typ und Anzahl), dann liefert `bit_or` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`bit_and`, `add_image`

Siehe auch

`bit_xor`, `bit_and`

Modul

Foundation

`bit_rshift ( Image : ImageRShift : Shift : )`

Rechts-Shift aller Pixel des Bildes.

`bit_rshift` berechnet einen bitweisen „rechts-Shift“ aller Pixel des Eingabebildes. Die Semantik der „rechts-Shift“ Operation entspricht der von C („»“) für die jeweiligen Typen (signed char, unsigned char, short, unsigned short, int/long). Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Eingabebild(er).

▷ **`ImageRShift`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Ergebnis(se) der Shift-Operation.

▷ **`Shift`** (input_control) ..... integer ⇝ *integer*
Shift-Wert.
Defaultwert: 3
Wertevorschläge: `Shift` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 31}
Typischer Wertebereich: 0 ≤ `Shift` ≤ 31
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Shift >= 1 && Shift <= 31

Beispiel

```
bit_rshift(Int2Image,&ReducedInt2Image,8);
convert_image_type(ReducedInt2Image,&ByteImage,"byte");
```

Ergebnis

Sind die Bilder korrekt (Typ) und hat `Shift` eine gültigen Wert, dann liefert `bit_rshift` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

scale_image

Siehe auch

bit_lshift

Modul

Foundation

bit_slice (Image : ImageSlice : Bit :)

Extraktion eines Bits aus den Pixeln.

bit_slice extrahiert eine Bitebene aus dem Eingabebild. Die Semantik der „and“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, unsigned short, int/long). Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Eingabebild(er).

▷ **ImageSlice** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4
Ergebnis(se) der Extraktion.

▷ **Bit** (input_control) ..... integer ⇝ *integer*
Auszuwählendes Bit.
Defaultwert: 8
Wertevorschläge: Bit ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 32}
Typischer Wertebereich: 1 ≤ Bit ≤ 32
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Bit >= 1 && Bit <= 32

Beispiel

```
read_image(Image,'fabrik')
for I:= 1 to 8 by 1
    bit_slice (Image, ImageSlice, I)
    threshold (ImageSlice, Region, 1, 255)
    dev_display (Region)
endfor
```

Ergebnis

Sind die Bilder korrekt (Typ) und hat Bit eine gültigen Wert, dann liefert bit_slice den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`threshold`, `bit_or`

Alternativen

`bit_mask`

Siehe auch

`bit_and`, `bit_lshift`

Modul

Foundation

bit_xor (Image1, Image2 : ImageXor : :)

Bitweise XOR aller Pixel der Eingabebilder.

`bit_xor` berechnet das bitweise „xor“ aller Pixel der Eingabebilder. Die Semantik der „xor“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, unsigned short, int/long). Die Bilder müssen die gleiche Bildgröße und den gleichen Pixeltyp haben. Es werden die Bildpunkte innerhalb des Definitionsbereiches des Bildes im ersten Parameter bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

Eingabebild(er) 1.

▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

Eingabebild(er) 2.

▷ **ImageXor** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4

Ergebnis(se) der XOR-Operation.

Beispiel

```
read_image(Image1,'fabrik')
dev_display (Image1)
read_image(Image2,'monkey')
dev_display (Image2)
bit_xor(Image1,Image2,ImageBitXor)
dev_display (ImageBitXor)
```

Ergebnis

Sind die Bilder korrekt (Typ und Anzahl), dann liefert `bit_xor` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

- Automatisch parallelisiert auf Domainebene.

Alternativen

bit_or, bit_and, add_image

Siehe auch

bit_or, bit_and

Modul

Foundation

10.3 FFT

convol_fft (ImageFFT, ImageFilter : ImageConvol : :)

Faltung (Convolution) mit einem Filter im Frequenzraum.

convol_fft führt eine Filterung der (fouriertransformierten) Eingabebilder im Frequenzraum durch. Eine Filterung im Frequenzraum bedeutet eine Multiplikation der Pixel des komplexen Bildes ImageFFT mit den zugehörigen Pixel des Filters ImageFilter.

Achtung

Die Filterung erfolgt immer für das gesamte Bild, d.h., der Definitionsbereich des Eingabebildes wird ignoriert.

Parameter

- ▷ **ImageFFT** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Komplexes Eingabebild.
- ▷ **ImageFilter** (input_object) ..... (multichannel-)image ⇝ *object :* real / complex
  Filter im Frequenzraum.
- ▷ **ImageConvol** (output_object) ..... image(-array) ⇝ *object :* complex
  Ergebnis der Filterung (Real- und Imaginärteile).

Beispiel

```
gen_highpass(Highpass,0.2,'n','dc_edge',Width,Height)
fft_generic(Image,ImageFFT,'to_freq',-1,'none','dc_edge','complex')
convol_fft(ImageFFT,Highpass,ImageConvol)
fft_generic(ImageConvol,ImageResult,'from_freq',1,'none','dc_edge','byte')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert convol_fft den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

fft_image, fft_generic, rft_generic, gen_highpass, gen_lowpass, gen_bandpass, gen_bandfilter

Nachfolger

power_byte, power_real, power_ln, fft_image_inv, fft_generic, rft_generic

Alternativen

convol_gabor

Siehe auch

gen_gabor, gen_highpass, gen_lowpass, gen_bandpass, convol_gabor, fft_image_inv

Modul

Foundation

convol_gabor (ImageFFT, GaborFilter : ImageResultGabor, ImageResultHilbert : :)

Faltung mit einem Gaborfilter im Frequenzraum.

convol_gabor berechnet die Faltung des Gaborfilters GaborFilter (siehe gen_gabor) und seiner Hilbert-transformierten mit einem bereits fouriertransformierten Bild im Frequenzraum. Das Ausgabebild ist komplex.

Achtung

Die Filterung erfolgt immer für das gesamte Bild, d.h., der Definitionsbereich des Eingabebildes wird ignoriert.

Parameter

- ▷ **ImageFFT** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Eingabebild (Real- und Imaginärteile).
- ▷ **GaborFilter** (input_object) ..... multichannel-image ⇝ *object :* real
  Gabor/Hilbert-Filter.
- ▷ **ImageResultGabor** (output_object) ..... image(-array) ⇝ *object :* complex
  Ergebnis Gaborfilterung.
- ▷ **ImageResultHilbert** (output_object) ..... image(-array) ⇝ *object :* complex
  Ergebnis Hilbertfilterung.

Beispiel

```
gen_gabor(Filter,1.4,0.4,1.0,1.5,'n','dc_edge',512,512)
fft_generic(Image,ImageFFT,'to_freq',-1,'none','dc_edge','complex')
convol_gabor(ImageFFT,Filter,Gabor,Hilbert)
fft_generic(Gabor,GaborInv,'from_freq',1,'none','dc_edge','byte')
fft_generic(Hilbert,HilbertInv,'from_freq',1,'none','dc_edge','byte')
energy_gabor(GaborInv,HilbertInv,Energy)
```

Ergebnis

Sind die Bilder vom richtigen Typ, dann liefert convol_gabor den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

fft_image, fft_generic, gen_gabor

Nachfolger

power_byte, power_real, power_ln, fft_image_inv, fft_generic

Alternativen

convol_fft

Siehe auch

convol_image

Modul

Foundation

correlation_fft (ImageFFT1, ImageFFT2 : ImageCorrelation : :)

Korrelation zweier Bilder im Frequenzraum.

correlation_fft führt eine Korrelation der fouriertransformierten Eingabebilder im Frequenzraum durch. Eine Korrelation im Frequenzraum bedeutet eine Multiplikation von ImageFFT1 mit dem komplex-konjugierten Bild von ImageFFT2. Es ist zu beachten, dass die korrekte Skalierung der Korrelation im Ortsraum nur erreicht werden kann, falls für die Hintransformation die Operatoren fft_generic oder rft_generic mit Norm = *'none'* und für die Rücktransformation fft_generic oder rft_generic mit Norm = *'n'* verwendet werden. Falls ImageFFT1 und ImageFFT2 dieselbe Anzahl Bilder enthalten, werden die entsprechenden Bilder paarweise korreliert. Ansonsten darf ImageFFT2 nur ein einziges Bild enthalten. In diesem Fall wird die Korrelation für jedes Bild von ImageFFT1 mit ImageFFT2 ausgeführt.

Achtung

Die Filterung erfolgt immer im Vollbild (Region wird ignoriert).

Parameter

- ▷ **ImageFFT1** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Fouriertransformiertes Eingabebild 1.
- ▷ **ImageFFT2** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Fouriertransformiertes Eingabebild 2.
  Parameteranzahl: ImageFFT2 == ImageFFT1 || ImageFFT2 == 1
- ▷ **ImageCorrelation** (output_object) ..... image(-array) ⇝ *object :* complex
  Korrelation der Eingabebilder im Frequenzraum.

Beispiel

```
* Compute the auto-correlation of an image.
get_image_size(Image,Width,Height)
rft_generic(Image,ImageFFT,'to_freq','none','complex',Width)
correlation_fft(ImageFFT,ImageFFT,Correlation)
rft_generic(Correlation,AutoCorrelation,'from_freq','n','real',Width)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert correlation_fft den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

fft_generic, fft_image, rft_generic

Nachfolger

fft_generic, fft_image_inv, rft_generic

Alternativen

phase_correlation_fft

Modul

Foundation

deserialize_fft_optimization_data (: : SerializedItemHandle :)

Deserialisieren von serialisierten FFT-Laufzeit-Optimierungsdaten.

deserialize_fft_optimization_data deserialisiert Daten zur Optimierung der Laufzeit der FFT, welche mit serialize_fft_optimization_data serialisiert wurden (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierten Daten werden in dem Handle SerializedItemHandle übergeben. Die Optimierungsdaten müssen zuvor mit optimize_fft_speed bestimmt und mit serialize_fft_optimization_data serialisiert worden sein. Falls die gespeicherten Daten für die später zu verwendenden Bildgrößen berechnet worden sind, erübrigt sich somit ein erneuter Aufruf von optimize_fft_speed. Es ist zu beachten, dass die Daten nur auf demselben Rechner verwendet werden sollten, auf dem sie mit optimize_fft_speed berechnet worden sind. Andernfalls werden die Laufzeiten nicht optimal sein. Außerdem ist zu beachten, dass Optimierungsdaten, die mit Standard HALCON erzeugt worden sind, mit Parallel HALCON nicht verwendet werden können (und umgekehrt).

deserialize_fft_optimization_data beeinflusst die Laufzeit der folgenden Operatoren, die die FFT verwenden: fft_generic, fft_image, fft_image_inv, sfs_pentland, sfs_mod_lr, sfs_orig_lr, wiener_filter.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameter korrekt, dann liefert deserialize_fft_optimization_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_fft_optimization_data

Nachfolger

fft_generic, fft_image, fft_image_inv, rft_generic, wiener_filter, wiener_filter_ni, photometric_stereo, sfs_pentland, sfs_mod_lr, sfs_orig_lr

Alternativen

optimize_fft_speed, optimize_rft_speed

Siehe auch

serialize_fft_optimization_data

Modul

Foundation

energy_gabor (ImageGabor, ImageHilbert : Energy : :)

Energie eines zweikanaligen Bildes.

energy_gabor berechnet den lokalen Kontrast (Energy) der zwei Eingabebilder. Die Energie des Bildes ergibt sich dann zu

$$\texttt{Energy} = \text{Kanal1}^2 + \text{Kanal2}^2 \ .$$

Häufig geht der Bestimmung der Energie die Faltung eines Bildes mit einem Gaborfilter und der Hilberttransformierten des Gaborfilters voraus (siehe convol_gabor). Als Kanal 1 wird energy_gabor dann das Gaborgefilterte Bild, in den Ortsraum zurück transformiert (siehe fft_image_inv) übergeben, als Kanal 2 das rücktransformierte Ergebnis der Faltung mit der Hilberttransformierten. Die lokale Energie ist ein Maß für den lokalen Kontrast von Strukturen (Kanten, Linien) in Bildern.

Parameter

- ▷ **ImageGabor** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / real
  Eingabe 1. Kanal (typisch: Gaborbild).
- ▷ **ImageHilbert** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / real
  Eingabe 2. Kanal (typisch: Hilbertbild).
- ▷ **Energy** (output_object) ..... image(-array) ⇝ *object :* real
  Bild mit der lokalen Energie.

Beispiel

```
fft_image(Image,&FFT);
gen_gabor(&Filter,1.4,0.4,1.0,1.5,512);
convol_gabor(FFT,Filter,&Gabor,&Hilbert);
fft_image_inv(Gabor,&GaborInv);
fft_image_inv(Hilbert,&HilbertInv);
energy_gabor(GaborInv,HilbertInv,&Energy);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `energy_gabor` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`gen_gabor`, `convol_gabor`, `fft_image_inv`

Modul

Foundation

fft_generic (Image : ImageFFT : Direction, Exponent, Norm, Mode, ResultType :)

Schnelle Fouriertransformation.

`fft_generic` berechnet die schnelle Fouriertransformation des Eingabebildes `Image`. Da in der Literatur mehrere Definitionen der Hin- und Rücktransformation der Fouriertransformation existieren, erlaubt diese Prozedur dem Benutzer die Auswahl der für seine Bedürfnisse geeigneten Version.

Die allgemeine Version einer Fouriertransformation sieht wie folgt aus:

$$F(m,n) = \frac{1}{c} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} e^{s2\pi i(km/M+ln/N)} f(k,l)$$

In der Literatur herrscht keine Einigkeit, ob das Vorzeichen s im Exponenten für die Hintransformation mit 1 oder -1 zu besetzen ist. Ebenso wenig Klarheit herrscht über den Normierungsfaktor c. Dieser wird manchmal für die Hintransformation mit 1 angegeben, manchmal mit MN, manchmal wird er auch (für die unitäre FFT) zu $\sqrt{MN}$ gesetzt. Insbesondere in der Bildverarbeitung wird außerdem manchmal der konstante Term der FFT (im englischen DC-Term genannt) in die Bildmitte verschoben.

`fft_generic` erlaubt es, diese Auswahlmöglichkeiten individuell anzupassen. Mit `Direction` kann die logische Richtung der FFT bestimmt werden. (Dieser Parameter ist nicht überflüssig; Er dient dazu, zu unterscheiden,

wie das Bild verschoben werden muss, falls der DC-Term in der Mitte des Bildes liegen soll.) Mögliche Werte sind *'to_freq'* und *'from_freq'*. Mit dem Parameter `Exponent` kann der Exponent der Transformation festgelegt werden. Er kann die Werte 1 und -1 annehmen. Mit `Norm` kann die gewünschte Normierung angegeben werden. Zulässige Werte sind *'none'*, *'sqrt'* und *'n'*. Mit `Mode` kann angegeben werden, wo der DC-Term der FFT liegen soll. Dabei kann zwischen *'dc_center'* und *'dc_edge'* gewählt werden.

In jedem Fall ist auf die konsistente Verwendung der Parameter zu achten. D.h., die Normierungsfaktoren bei der Hin- und Rücktransformation müssen aufmultipliziert MN ergeben. Bei der Rücktransformation ist ein anderes Vorzeichen für den Exponenten zu wählen als bei der Hintransformation. `Mode` muss für beide Transformationen gleich gewählt werden.

Eine sinnvolle Kombination ist z.B. *'(to_freq,-1,n,dc_edge)'* für die Hintransformation und *'(from_freq,1,none,dc_edge)'* für die Rücktransformation. In diesem Fall kann die FFT als Interpolation mit trigonometrischen Basisfunktionen interpretiert werden. Eine andere mögliche Kombination ist *'(to_freq,-1,sqrt,dc_center)'* und *'(from_freq,1,sqrt,dc_center)'*.

Der Parameter `ResultType` erlaubt es, bei der Rücktransformation (`Direction` = *'from_freq'*) den Ergebnisbildtyp festzulegen. Bei der Hintransformation (`Direction` = *'to_freq'*) muss `ResultType` auf *'complex'* gesetzt werden.

Achtung

Die Berechnung erfolgt immer für das gesamte Bild, d.h., der Definitionsbereich des Eingabebildes wird ignoriert.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex
  Eingabebild im Ortsraum.
- ▷ **`ImageFFT`** (output_object) ..... image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex
  Fouriertransformiertes Bild.
- ▷ **`Direction`** (input_control) ..... string ⇝ *string*
  Hin- oder Rücktransformation.
  Defaultwert: 'to_freq'
  Werteliste: `Direction` ∈ {'to_freq', 'from_freq'}
- ▷ **`Exponent`** (input_control) ..... integer ⇝ *integer*
  Vorzeichen des Exponenten.
  Defaultwert: -1
  Werteliste: `Exponent` ∈ {-1, 1}
- ▷ **`Norm`** (input_control) ..... string ⇝ *string*
  Normierung des Ergebnisbildes.
  Defaultwert: 'sqrt'
  Werteliste: `Norm` ∈ {'none', 'sqrt', 'n'}
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Position der Nullfrequenz im Frequenzraum.
  Defaultwert: 'dc_center'
  Werteliste: `Mode` ∈ {'dc_center', 'dc_edge'}
- ▷ **`ResultType`** (input_control) ..... string ⇝ *string*
  Bildtyp des Ausgabebildes.
  Defaultwert: 'complex'
  Werteliste: `ResultType` ∈ {'complex', 'byte', 'int1', 'int2', 'uint2', 'int4', 'real', 'direction', 'cyclic'}

Beispiel

```
/* simulation of fft_image */
void my_fft(Hobject In, Hobject *Out)
{
  fft_generic(In,Out,"to_freq",-1,"sqrt","dc_center","complex");
}

/* simulation of fft_image_inv */
void my_fft_image_inv(Hobject In, Hobject *Out)
```

```
{
  fft_generic(In,Out,"from_freq",1,"sqrt","dc_center","byte");
}
```

Ergebnis

Sind alle Parameterwerte korrekt, dann liefert `fft_generic` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`optimize_fft_speed`, `read_fft_optimization_data`

Nachfolger

`convol_fft`, `correlation_fft`, `phase_correlation_fft`, `convol_gabor`, `convert_image_type`, `power_byte`, `power_real`, `power_ln`, `phase_deg`, `phase_rad`, `energy_gabor`

Alternativen

`fft_image`, `fft_image_inv`, `rft_generic`

Modul

Foundation

`fft_image` (Image : ImageFFT : :)

Schnelle Fouriertransformation.

`fft_image` berechnet die Fouriertransformierte des Eingabebildes (`Image`), vollzieht also den Übergang vom Orts- in den Frequenzraum. Die Berechnung erfolgt mittels des Fast-Fourier-Algorithmus. Dies entspricht dem Aufruf von

```
fft_generic(Image,ImageFFT,'to_freq',-1,'sqrt','dc_center','complex')
```

.

Achtung

Die Berechnung erfolgt immer für das gesamte Bild, d.h., der Definitionsbereich des Eingabebildes wird ignoriert.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / real
Eingabebild im Ortsraum

▷ **ImageFFT** (output_object) ..... image(-array) ⇝ *object :* complex
Fouriertransformiertes Bild.

Ergebnis

Ist der Pixeltyp korrekt, dann liefert `fft_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

optimize_fft_speed, read_fft_optimization_data

Nachfolger

convol_fft, convol_gabor, convert_image_type, power_byte, power_real, power_ln, phase_deg, phase_rad

Alternativen

fft_generic, rft_generic

Siehe auch

fft_image_inv

Modul

Foundation

fft_image_inv (Image : ImageFFTInv : :)

Schnelle inverse Fouriertransformation.

fft_image_inv führt auf den Eingabebildern die inverse Fouriertransformation aus, vollzieht also den Übergang vom Frequenzraum zurück in den Ortsraum. Das Ausgabebild ist vom Type *'byte'*. Dies entspricht dem Aufruf von

```
fft_generic(Image,ImageFFT,'from_freq',1,'sqrt','dc_center','byte')
```

.

Achtung

Die Berechnung erfolgt immer für das gesamte Bild, d.h., der Definitionsbereich des Eingabebildes wird ignoriert.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Eingabebild (Frequenzraum).
- ▷ **ImageFFTInv** (output_object) ..... image(-array) ⇝ *object :* byte
  Rücktransformierte Bilder (Ortsraum).

Ergebnis

Ist der Typ der Bildes korrekt, dann liefert fft_image_inv den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

convol_fft, convol_gabor, fft_image, optimize_fft_speed, read_fft_optimization_data

Nachfolger

convert_image_type, energy_gabor

Alternativen

fft_generic, rft_generic

Siehe auch

fft_image, fft_generic, energy_gabor

Modul

Foundation

gen_bandfilter (: ImageFilter : MinFrequency, MaxFrequency, Norm, Mode, Width, Height :)

Erzeugen eines idealen Bandfilters.

gen_bandfilter erzeugt einen idealen Bandfilter im Frequenzraum. Die Frequenzen geben die Parameter MinFrequency und MaxFrequency vor. Die Frequenzen werden als Bruchteil der maximalen (horizontalen und vertikalen) Frequenz, die in einem Bild der Größe Width × Height möglich ist, angegeben, d.h. MinFrequency und MaxFrequency sollten zwischen 0 und 1 liegen. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit Norm der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls fft_generic verwendet wird, falls Norm = *'n'* verwendet wird. Mit Mode kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls fft_generic verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls fft_image und fft_image_inv zur Filterung verwendet werden, muss Norm = *'none'* und Mode = *'dc_center'* verwendet werden. Falls rft_generic verwendet wird, muss Mode = *'rft'* verwendet werden. Das Ergebnis ist ein Bild, bei dem ein Ring um das Zentrum auf 0 gesetzt ist. Alle anderen Pixel haben als Wert den Normierungsfaktor.

Parameter

- ▷ **ImageFilter** (output_object) ..... image ⇝ *object* : real
  Bandfilter im Frequenzraum.
- ▷ **MinFrequency** (input_control) ..... real ⇝ *real*
  Minimale Frequenz.
  Defaultwert: 0.1
  Wertevorschläge: MinFrequency ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: MinFrequency >= 0
- ▷ **MaxFrequency** (input_control) ..... real ⇝ *real*
  Maximale Frequenz.
  Defaultwert: 0.2
  Wertevorschläge: MaxFrequency ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: MaxFrequency >= 0 && MaxFrequency >= MinFrequency
- ▷ **Norm** (input_control) ..... string ⇝ *string*
  Normierung des Filters.
  Defaultwert: 'none'
  Werteliste: Norm ∈ {'none', 'n'}
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Position der Nullfrequenz im Frequenzraum.
  Defaultwert: 'dc_center'
  Werteliste: Mode ∈ {'dc_center', 'dc_edge', 'rft'}
- ▷ **Width** (input_control) ..... integer ⇝ *integer*
  Breite des Bildes (Filters).
  Defaultwert: 512
  Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}
- ▷ **Height** (input_control) ..... integer ⇝ *integer*
  Höhe des Bildes (Filters).
  Defaultwert: 512
  Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
* Filtering with maximum efficiency with fft_generic.
```

```
gen_bandpass(Bandfilter,0.2,0.4,'n','dc_edge',Width,Height)
fft_generic(Image,ImageFFT,'to_freq',-1,'none','dc_edge','complex')
convol_fft(ImageFFT,Bandfilter,ImageConvol)
fft_generic(ImageConvol,ImageResult,'from_freq',1,'none','dc_edge','byte')
```

Ergebnis

Sind die Parameter korrekt, dann liefert `gen_bandfilter` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`convol_fft`

Alternativen

`gen_circle`, `paint_region`

Siehe auch

`gen_highpass`, `gen_lowpass`, `gen_bandpass`, `gen_gauss_filter`, `gen_mean_filter`, `gen_derivative_filter`

Modul

Foundation

gen_bandpass (: ImageBandpass : MinFrequency, MaxFrequency, Norm, Mode, Width, Height :)

Erzeugen eines idealen Bandpassfilters.

`gen_bandpass` erzeugt einen idealen Bandpassfilter im Frequenzraum. Die Frequenzen geben die Parameter `MinFrequency` und `MaxFrequency` vor. Die Frequenzen werden als Bruchteil der maximalen (horizontalen und vertikalen) Frequenz, die in einem Bild der Größe `Width` × `Height` möglich ist, angegeben, d.h. `MinFrequency` und `MaxFrequency` sollten zwischen 0 und 1 liegen. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit `Norm` der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls `fft_generic` verwendet wird, falls `Norm` = *'n'* verwendet wird. Mit `Mode` kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls `fft_generic` verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls `fft_image` und `fft_image_inv` zur Filterung verwendet werden, muss `Norm` = *'none'* und `Mode` = *'dc_center'* verwendet werden. Falls `rft_generic` verwendet wird, muss `Mode` = *'rft'* verwendet werden. Das Ergebnis ist ein Bild, das mit Ausnahme eines Ringes auf Null gesetzt ist.

Parameter

▷ **ImageBandpass** (output_object) ..... image ⇝ *object :* real
Bandpass im Frequenzraum.

▷ **MinFrequency** (input_control) ..... real ⇝ *real*
Minimale Frequenz.
Defaultwert: 0.1
Wertevorschläge: `MinFrequency` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: MinFrequency >= 0

▷ **MaxFrequency** (input_control) ..... real ⇝ *real*
Maximale Frequenz.
Defaultwert: 0.2
Wertevorschläge: `MaxFrequency` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: MaxFrequency >= 0 && MaxFrequency >= MinFrequency

▷ **Norm** (input_control) ..... string ⇝ *string*
Normierung des Filters.
Defaultwert: 'none'
Werteliste: Norm ∈ {'none', 'n'}

▷ **Mode** (input_control) ..... string ⇝ *string*
Position der Nullfrequenz im Frequenzraum.
Defaultwert: 'dc_center'
Werteliste: Mode ∈ {'dc_center', 'dc_edge', 'rft'}

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
* Filtering with maximum efficiency with fft_generic.
gen_bandpass(Bandpass,0.2,0.4,'n','dc_edge',Width,Height)
fft_generic(Image,ImageFFT,'to_freq',-1,'none','dc_edge','complex')
convol_fft(ImageFFT,Bandpass,ImageConvol)
fft_generic(ImageConvol,ImageResult,'from_freq',1,'none','dc_edge','byte')
```

Ergebnis

Sind die Parameter korrekt, dann liefert gen_bandpass den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

convol_fft

Siehe auch

gen_highpass, gen_lowpass, gen_bandfilter, gen_gauss_filter, gen_mean_filter, gen_derivative_filter

Modul

Foundation

gen_derivative_filter (: ImageDerivative : Derivative, Exponent, Norm, Mode, Width, Height :)

Erzeugen eines Ableitungsfilters im Frequenzraum.

gen_derivative_filter erzeugt einen Ableitungsfilter im Frequenzraum. Die zu berechnende Ableitung wird mit Derivative festgelegt. Mit Exponent wird der bei der Rücktransformation verwendete Exponent festgelegt. Dieser muss dem bei fft_generic verwendeten Wert entsprechen. Falls fft_image_inv zur Rücktransformation verwendet wird, muss Exponent = *1* verwendet werden. Da das Ableitungsbild im Normalfall negative Werte enthält, sollte aber sowieso immer fft_generic zur Rücktransformation verwendet werden. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit Norm der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls fft_generic verwendet wird, falls Norm = *'n'* verwendet wird. Mit Mode kann festgelegt werden, wo der Nullfrequenz der FFT liegen

soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls `fft_generic` verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls `fft_image` und `fft_image_inv` zur Filterung verwendet werden, muss `Norm` = *'none'* und `Mode` = *'dc_center'* verwendet werden. Falls `rft_generic` verwendet wird, muss `Mode` = *'rft'* verwendet werden.

Parameter

▷ **`ImageDerivative`** (output_object) ..... image ⇝ *object :* complex
Ableitungsfilter als Bild in Frequenzraum.

▷ **`Derivative`** (input_control) ..... string ⇝ *string*
Zu berechnende Ableitung.
Defaultwert: 'x'
Wertevorschläge: `Derivative` ∈ {'x', 'y', 'xx', 'xy', 'yy', 'xxx', 'xxy', 'xyy', 'yyy'}

▷ **`Exponent`** (input_control) ..... integer ⇝ *integer*
Exponent, der bei der Rücktransformation verwendet wird.
Defaultwert: 1
Wertevorschläge: `Exponent` ∈ {-1, 1}

▷ **`Norm`** (input_control) ..... string ⇝ *string*
Normierung des Filters.
Defaultwert: 'none'
Werteliste: `Norm` ∈ {'none', 'n'}

▷ **`Mode`** (input_control) ..... string ⇝ *string*
Position der Nullfrequenz im Frequenzraum.
Defaultwert: 'dc_center'
Werteliste: `Mode` ∈ {'dc_center', 'dc_edge', 'rft'}

▷ **`Width`** (input_control) ..... integer ⇝ *integer*
Breite des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}

▷ **`Height`** (input_control) ..... integer ⇝ *integer*
Höhe des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: `Height` ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
* Generate a smoothed derivative filter.
gen_gauss_filter (ImageGauss, Sigma, Sigma, 0, 'n', 'dc_edge', 512, 512)
convert_image_type (ImageGauss, ImageGaussComplex, 'complex')
gen_derivative_filter (ImageDerivX, 'x', 1, 'none', 'dc_edge', 512, 512)
mult_image (ImageGaussComplex, ImageDerivX, ImageDerivXGauss, 1, 0)
* Filter an image with the smoothed derivative filter.
fft_generic (Image, ImageFFT, 'to_freq', -1, 'none', 'dc_edge', 'complex')
convol_fft (ImageFFT, ImageDerivXGauss, Filtered)
fft_generic (Filtered, ImageX, 'from_freq', 1, 'none', 'dc_edge', 'real')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_derivative_filter` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fft_image`, `fft_generic`, `rft_generic`

Nachfolger

`convol_fft`

Siehe auch

fft_image_inv, gen_gauss_filter, gen_mean_filter, gen_lowpass, gen_bandpass, gen_bandfilter, gen_highpass

Modul

Foundation

gen_filter_mask (: ImageFilter : FilterMask, Scale, Width, Height :)

Eintragen einer Filtermaske im Ortsraum in ein real-Bild.

gen_filter_mask trägt eine Filtermaske im Ortsraum in ein Bild ein. Das Zentrum des Filters kommt in der Mitte des Bildes zu liegen. Der Parameter Scale gibt an, um wie viel die Filterkoeffizienten zu multiplizieren sind (dies führt zu einer stärkeren Antwort bei der Fouriertransformation). Die Filtermatrix, die durch FilterMask definiert ist, kann dabei entweder aus einer Datei oder einem Tupel generiert werden. Das Format für die Filtermatrix ist bei convol_image beschrieben. Beispieldateien sind im Directory „filter“ unter dem HALCON-Homedirectory zu finden. Die Operation kann gut zur Visualisierung von Filter (durch anschließende Fouriertransformation) verwendet werden.

Parameter

▷ **ImageFilter** (output_object) ..... image ⇝ *object :* real
Filter im Ortsraum.

▷ **FilterMask** (input_control) ..... filename.read(-array) ⇝ *string* / integer
Filtermaske als Dateiname oder Tupel.
Defaultwert: 'gauss'
Wertevorschläge: FilterMask ∈ {'gauss', 'laplace4', 'laplace8', 'lowpas_3_3', 'lowpas_5_5', 'lowpas_7_7', 'lowpas_9_9', 'sobel_c', 'sobel_l'}

▷ **Scale** (input_control) ..... real ⇝ *real*
Skalierungsfaktor.
Defaultwert: 1.0
Wertevorschläge: Scale ∈ {0.3, 0.5, 0.75, 1.0, 1.25, 1.5, 2.0}
Typischer Wertebereich: 0.001 ≤ Scale ≤ 10.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1
Restriktion: Scale > 0.0

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
* If the filter should be read from a file:
gen_filter_mask (Filter, 'lowpas_3_3', 1.0, 512, 512)
* If the filter should be directly passed as a tuple:
gen_filter_mask (Filter, [3,3,9,1,1,1,1,1,1,1,1,1], 1.0, 512, 512)
fft_image (Filter, FilterFFT)
dev_set_paint ('3d_plot')
dev_display (FilterFFT)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`fft_image`, `fft_generic`

Siehe auch

`convol_image`

Modul

Foundation

gen_gabor (: ImageFilter : Angle, Frequency, Bandwidth, Orientation, Norm, Mode, Width, Height :)

Erzeugen eines Gaborfilters.

`gen_gabor` erzeugt einen Gaborfilter, dessen Bandpassverhalten und die zugehörigen Signumwerte für die Berechnung der Hilberttransformierten durch die Eingabeparameter einstellbar ist. Dazu wird ein symmetrischer Filter im Frequenzraum errechnet, der sich durch die Parameter `Angle`, `Frequency`, `Bandwidth` und `Orientation` so einstellen lässt, dass ein bestimmter Frequenzbereich in einem bestimmten Richtungsbereich eines Bildes im Ortsraum durch die Faltung im Ortsfrequenzraum herausgefiltert wird.

Die Parameter `Frequency` (Mittenfrequenz = Abstand vom Mittelpunkt) und `Orientation` (Winkel) geben das Zentrum des Filters an. Größere Werte von `Frequency` ergeben höhere Frequenzen. Ein Wert von 0 bei `Orientation` erzeugt eine „Sichel“ die horizontal ausgerichtet ist (Öffnung nach „unten“). Höhere Winkelwerte lassen die Sichel in mathematisch positiver Richtung rotieren.

Mit den Parametern `Angle` und `Bandwidth` kann die Größe des Richtungsbereiches und des Frequenzbereiches geändert werden. Je größer der Parameter `Angle` ist, desto kleiner wird der Winkelbereich (kürzere Sichel), das heißt, desto kleiner wird der Richtungsbereich. Je größer der Parameter `Bandwidth`, desto kleiner wird der Frequenzbereich (dünnere Sichel).

Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit `Norm` der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls `fft_generic` verwendet wird, falls `Norm` = *'n'* verwendet wird. Mit `Mode` kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll. Falls `fft_generic` verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls `fft_image` und `fft_image_inv` zur Filterung verwendet werden, muss `Norm` = *'none'* und `Mode` = *'dc_center'* verwendet werden. Es ist zu beachten, dass `gen_gabor` keinen Filter für `rft_generic` erzeugen kann.

Die Berechnung liefert ein zweikanaliges Real-Bild, welches im ersten Kanal den Gaborfilter und im zweiten Kanal den zugehörigen Hilbertfilter enthält.

Parameter

▷ **ImageFilter** (output_object) . . . . . multichannel-image ⇝ *object :* real
Gabor- und Hilbertfilter.

▷ **Angle** (input_control) . . . . . real ⇝ *real*
Winkel, indirekt proportional zur Größe des Richtungsbereichs.
Defaultwert: 1.4
Wertevorschläge: `Angle` ∈ {1.0, 1.2, 1.4, 1.6, 2.0, 2.5, 3.0, 5.0, 6.0, 10.0, 20.0, 30.0, 50.0, 70.0, 100.0}
Typischer Wertebereich: 1.0 ≤ `Angle` ≤ 500.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1

▷ **Frequency** (input_control) . . . . . real ⇝ *real*
Abstand des Filterschwerpunktes vom Mittelpunkt.
Defaultwert: 0.4
Wertevorschläge: `Frequency` ∈ {0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.50, 0.55, 0.60, 0.65, 0.699}
Typischer Wertebereich: 0.0 ≤ `Frequency` ≤ 0.7
Minimale Schrittweite: 0.00001
Empfohlene Schrittweite: 0.005

▷ **Bandwidth** (input_control) . . . . . real ⇝ *real*
Bandbreite, indirekt proportional zur Größe des Frequenzbereichs.
Defaultwert: 1.0
Wertevorschläge: `Bandwidth` ∈ {0.1, 0.3, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0, 50.0}
Typischer Wertebereich: 0.05 ≤ `Bandwidth` ≤ 100.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1

▷ **Orientation** (input_control) . . . . . real ⇝ *real*
Richtung, legt zusammen mit Frequency die Lage des Filterschwerpunktes fest.
Defaultwert: 1.5
Wertevorschläge: `Orientation` ∈ {0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.14}
Typischer Wertebereich: 0.0 ≤ `Orientation` ≤ 3.1416
Minimale Schrittweite: 0.0001
Empfohlene Schrittweite: 0.05

▷ **Norm** (input_control) . . . . . string ⇝ *string*
Normierung des Filters.
Defaultwert: 'none'
Werteliste: `Norm` ∈ {'none', 'n'}

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Position der Nullfrequenz im Frequenzraum.
Defaultwert: 'dc_center'
Werteliste: `Mode` ∈ {'dc_center', 'dc_edge'}

▷ **Width** (input_control) . . . . . integer ⇝ *integer*
Breite des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}

▷ **Height** (input_control) . . . . . integer ⇝ *integer*
Höhe des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: `Height` ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
gen_gabor(Filter,1.4,0.4,1.0,1.5,'n','dc_edge',512,512)
fft_generic(Image,ImageFFT,'to_freq',-1,'none','dc_edge','complex')
convol_gabor(ImageFFT,Filter,Gabor,Hilbert)
fft_generic(Gabor,GaborInv,'from_freq',1,'none','dc_edge','byte')
fft_generic(Hilbert,HilbertInv,'from_freq',1,'none','dc_edge','byte')
energy_gabor(GaborInv,HilbertInv,Energy)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_gabor` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fft_image`, `fft_generic`

Nachfolger

`convol_gabor`

Alternativen

`gen_bandpass`, `gen_bandfilter`, `gen_highpass`, `gen_lowpass`

Siehe auch

`fft_image_inv`, `energy_gabor`

Modul

Foundation

gen_gauss_filter (: ImageGauss : Sigma1, Sigma2, Phi, Norm, Mode, Width, Height :)

Erzeugen eines Gaußfilters im Frequenzraum.

`gen_gauss_filter` erzeugt einen (evtl. anisotropen) Gaußfilter im Frequenzraum. Die Standardabweichungen (d.h. die Stärke der Glättung) des Gaußfilters im Ortsraum werden durch `Sigma1` und `Sigma2` festgelegt. Dabei ist `Sigma1` die Standardabweichung in Richtung der durch den Winkel `Phi` gegebenen Hauptrichtung des Filters im Ortsraum. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit `Norm` der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls `fft_generic` verwendet wird, falls `Norm` = *'n'* verwendet wird. Mit `Mode` kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls `fft_generic` verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls `fft_image` und `fft_image_inv` zur Filterung verwendet werden, muss `Norm` = *'none'* und `Mode` = *'dc_center'* verwendet werden. Falls `rft_generic` verwendet wird, muss `Mode` = *'rft'* verwendet werden.

Parameter

- ▷ **ImageGauss** (output_object) . . . . . image ⇝ *object :* real
  Gaußfilter als Bild in Frequenzraum.
- ▷ **Sigma1** (input_control) . . . . . real ⇝ *real*
  Standardabweichung des Gaußfilters in der Hauptrichtung des Filters im Ortsraum.
  Defaultwert: 1.0
  Wertevorschläge: `Sigma1` ∈ {0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0}
  Restriktion: Sigma1 >= 0
- ▷ **Sigma2** (input_control) . . . . . real ⇝ *real*
  Standardabweichung des Gaußfilters senkrecht zur Hauptrichtung des Filters im Ortsraum.
  Defaultwert: 1.0
  Wertevorschläge: `Sigma2` ∈ {0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0}
  Restriktion: Sigma2 >= 0
- ▷ **Phi** (input_control) . . . . . angle.rad ⇝ *real*
  Hauptrichtung des Filters im Ortsraum.
  Defaultwert: 0.0
  Wertevorschläge: `Phi` ∈ {0.0, 0.523599, 0.785398, 1.047198, 1.570796, 2.094395, 2.356194, 2.617994, 3.141593}
- ▷ **Norm** (input_control) . . . . . string ⇝ *string*
  Normierung des Filters.
  Defaultwert: 'none'
  Werteliste: `Norm` ∈ {'none', 'n'}
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Position der Nullfrequenz im Frequenzraum.
  Defaultwert: 'dc_center'
  Werteliste: `Mode` ∈ {'dc_center', 'dc_edge', 'rft'}
- ▷ **Width** (input_control) . . . . . integer ⇝ *integer*
  Breite des Bildes (Filters).
  Defaultwert: 512
  Wertevorschläge: `Width` ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}
- ▷ **Height** (input_control) . . . . . integer ⇝ *integer*
  Höhe des Bildes (Filters).
  Defaultwert: 512
  Wertevorschläge: `Height` ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
* Generate a smoothed derivative filter.
gen_gauss_filter (ImageGauss, Sigma, Sigma, 0, 'n', 'dc_edge', 512, 512)
convert_image_type (ImageGauss, ImageGaussComplex, 'complex')
gen_derivative_filter (ImageDerivX, 'x', 1, 'none', 'dc_edge', 512, 512)
mult_image (ImageGaussComplex, ImageDerivX, ImageDerivXGauss, 1, 0)
* Filter an image with the smoothed derivative filter.
fft_generic (Image, ImageFFT, 'to_freq', -1, 'none', 'dc_edge', 'complex')
convol_fft (ImageFFT, ImageDerivXGauss, Filtered)
fft_generic (Filtered, ImageX, 'from_freq', 1, 'none', 'dc_edge', 'real')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_gauss_filter` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fft_image`, `fft_generic`, `rft_generic`

Nachfolger

`convol_fft`

Siehe auch

`fft_image_inv`, `gen_mean_filter`, `gen_derivative_filter`, `gen_lowpass`, `gen_bandpass`, `gen_bandfilter`, `gen_highpass`

Modul

Foundation

gen_highpass (: ImageHighpass : Frequency, Norm, Mode, Width, Height :)

Erzeugen eines idealen Hochpassfilters.

`gen_highpass` erzeugt einen idealen Hochpassfilter im Frequenzraum. Die Frequenz gibt der Parameter `Frequency` vor. Die Frequenz wird als Bruchteil der maximalen (horizontalen und vertikalen) Frequenz, die in einem Bild der Größe `Width` × `Height` möglich ist, angegeben, d.h. `Frequency` sollte zwischen 0 und 1 liegen. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit `Norm` der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls `fft_generic` verwendet wird, falls `Norm` = *'n'* verwendet wird. Mit `Mode` kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls `fft_generic` verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls `fft_image` und `fft_image_inv` zur Filterung verwendet werden, muss `Norm` = *'none'* und `Mode` = *'dc_center'* verwendet werden. Falls `rft_generic` verwendet wird, muss `Mode` = *'rft'* verwendet werden. Das Ergebnis ist ein Bild, das im inneren Bereich den Wert Null hat und außerhalb dieses Bereiches auf den Normierungsfaktor gesetzt ist.

Parameter

▷ **ImageHighpass** (output_object) ..... image ⇝ *object :* real
Hochpassfilter im Frequenzraum.

▷ **Frequency** (input_control) ..... real ⇝ *real*
Trennfrequenz.
Defaultwert: 0.1
Wertevorschläge: `Frequency` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: Frequency >= 0

▷ **Norm** (input_control) ..... string ⇝ *string*
Normierung des Filters.
Defaultwert: 'none'
Werteliste: Norm ∈ {'none', 'n'}

▷ **Mode** (input_control) ..... string ⇝ *string*
Position der Nullfrequenz im Frequenzraum.
Defaultwert: 'dc_center'
Werteliste: Mode ∈ {'dc_center', 'dc_edge', 'rft'}

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
* Filtering with maximum efficiency with fft_generic.
gen_highpass(Highpass,0.2,'n','dc_edge',Width,Height)
fft_generic(Image,ImageFFT,'to_freq',-1,'none','dc_edge','complex')
convol_fft(ImageFFT,Highpass,ImageConvol)
fft_generic(ImageConvol,ImageResult,'from_freq',1,'none','dc_edge','byte')
```

Ergebnis

Sind die Parameter korrekt, dann liefert gen_highpass den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

convol_fft

Siehe auch

convol_fft, gen_lowpass, gen_bandpass, gen_bandfilter, gen_gauss_filter, gen_mean_filter, gen_derivative_filter

Modul

Foundation

gen_lowpass (: ImageLowpass : Frequency, Norm, Mode, Width, Height :)

Erzeugen eines idealen Tiefpassfilters.

gen_lowpass erzeugt einen idealen Tiefpassfilter im Frequenzraum. Die Frequenz gibt der Parameter Frequency vor. Die Frequenz wird als Bruchteil der maximalen (horizontalen und vertikalen) Frequenz, die in einem Bild der Größe Width × Height möglich ist, angegeben, d.h. Frequency sollte zwischen 0 und 1 liegen. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit Norm der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls fft_generic verwendet wird, falls Norm = *'n'* verwendet wird. Mit Mode kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls fft_generic verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls fft_image und fft_image_inv zur Filterung

verwendet werden, muss Norm = *'none'* und Mode = *'dc_center'* verwendet werden. Falls rft_generic verwendet wird, muss Mode = *'rft'* verwendet werden. Das Ergebnis ist ein Bild, das im inneren Bereich als Wert den Normierungsfaktor hat und außerhalb dieses Bereiches auf 0 gesetzt ist.

Parameter

- ▷ **ImageLowpass** (output_object) ..... image ⇝ *object :* real
  Tiefpass im Frequenzraum.
- ▷ **Frequency** (input_control) ..... real ⇝ *real*
  Trennfrequenz.
  Defaultwert: 0.1
  Wertevorschläge: Frequency ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: Frequency >= 0
- ▷ **Norm** (input_control) ..... string ⇝ *string*
  Normierung des Filters.
  Defaultwert: 'none'
  Werteliste: Norm ∈ {'none', 'n'}
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Position der Nullfrequenz im Frequenzraum.
  Defaultwert: 'dc_center'
  Werteliste: Mode ∈ {'dc_center', 'dc_edge', 'rft'}
- ▷ **Width** (input_control) ..... integer ⇝ *integer*
  Breite des Bildes (Filters).
  Defaultwert: 512
  Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}
- ▷ **Height** (input_control) ..... integer ⇝ *integer*
  Höhe des Bildes (Filters).
  Defaultwert: 512
  Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
* Filtering with maximum efficiency with fft_generic.
gen_lowpass(Lowpass,0.2,'n','dc_edge',Width,Height)
fft_generic(Image,ImageFFT,'to_freq',-1,'none','dc_edge','complex')
convol_fft(ImageFFT,Lowpass,ImageConvol)
fft_generic(ImageConvol,ImageResult,'from_freq',1,'none','dc_edge','byte')
```

Ergebnis

Sind die Parameter korrekt, dann liefert gen_lowpass den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

convol_fft

Siehe auch

gen_highpass, gen_bandpass, gen_bandfilter, gen_gauss_filter, gen_mean_filter, gen_derivative_filter

Modul

Foundation

gen_mean_filter (: ImageMean : MaskShape, Diameter1, Diameter2, Phi, Norm, Mode, Width, Height :)

Erzeugen eines Mittelwertfilters im Frequenzraum.

gen_mean_filter erzeugt einen Mittelwertfilter im Frequenzraum. Die Form des Mittelwertfilters wird mit MaskShape festgelegt. Für MaskShape = *'rectangle'* wird ein rechteckiger Mittelwertfilter erzeugt. Für MaskShape = *'ellipse'* wird ein ellipsenförmiger Mittelwertfilter erzeugt. Die Durchmesser (d.h. die Stärke der Glättung) des Mittelwertfilters im Ortsraum werden durch Diameter1 und Diameter2 festgelegt. Dabei ist Diameter1 der Durchmesser in Richtung der durch den Winkel Phi gegebenen Hauptrichtung des Filters im Ortsraum. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit Norm der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls fft_generic verwendet wird, falls Norm = *'n'* verwendet wird. Mit Mode kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls fft_generic verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls fft_image und fft_image_inv zur Filterung verwendet werden, muss Norm = *'none'* und Mode = *'dc_center'* verwendet werden. Falls rft_generic verwendet wird, muss Mode = *'rft'* verwendet werden.

Parameter

- ▷ **ImageMean** (output_object) . . . . . image ⇝ *object :* real
  Mittelwertfilter als Bild in Frequenzraum.
- ▷ **MaskShape** (input_control) . . . . . string ⇝ *string*
  Form der Filtermaske im Ortsraum.
  Defaultwert: 'ellipse'
  Werteliste: MaskShape ∈ {'ellipse', 'rectangle'}
- ▷ **Diameter1** (input_control) . . . . . real ⇝ *real*
  Durchmesser des Mittelwertfilters in der Hauptrichtung des Filters im Ortsraum.
  Defaultwert: 11.0
  Wertevorschläge: Diameter1 ∈ {3.0, 5.0, 7.0, 9.0, 11.0, 15.0, 21.0, 31.0, 51.0}
  Restriktion: Diameter1 > 0
- ▷ **Diameter2** (input_control) . . . . . real ⇝ *real*
  Durchmesser des Mittelwertfilters senkrecht zur Hauptrichtung des Filters im Ortsraum.
  Defaultwert: 11.0
  Wertevorschläge: Diameter2 ∈ {3.0, 5.0, 7.0, 9.0, 11.0, 15.0, 21.0, 31.0, 51.0}
  Restriktion: Diameter2 > 0
- ▷ **Phi** (input_control) . . . . . angle.rad ⇝ *real*
  Hauptrichtung des Filters im Ortsraum.
  Defaultwert: 0.0
  Wertevorschläge: Phi ∈ {0.0, 0.523599, 0.785398, 1.047198, 1.570796, 2.094395, 2.356194, 2.617994, 3.141593}
- ▷ **Norm** (input_control) . . . . . string ⇝ *string*
  Normierung des Filters.
  Defaultwert: 'none'
  Werteliste: Norm ∈ {'none', 'n'}
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Position der Nullfrequenz im Frequenzraum.
  Defaultwert: 'dc_center'
  Werteliste: Mode ∈ {'dc_center', 'dc_edge', 'rft'}
- ▷ **Width** (input_control) . . . . . integer ⇝ *integer*
  Breite des Bildes (Filters).
  Defaultwert: 512
  Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}
- ▷ **Height** (input_control) . . . . . integer ⇝ *integer*
  Höhe des Bildes (Filters).
  Defaultwert: 512
  Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Beispiel

```
* Generate a circular mean filter.
gen_mean_filter (FilterMean, 'ellipse', 15, 15, 0, 'n', 'dc_edge', 512, 512)
* Filter an image with the circular mean filter.
fft_generic (Image, ImageFFT, 'to_freq', -1, 'none', 'dc_edge', 'complex')
```

```
convol_fft (ImageFFT, FilterMean, Filtered)
fft_generic (Filtered, ImageMean, 'from_freq', 1, 'none', 'dc_edge', 'real')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_mean_filter den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fft_image, fft_generic, rft_generic

Nachfolger

convol_fft

Siehe auch

fft_image_inv, gen_gauss_filter, gen_derivative_filter, gen_lowpass, gen_bandpass, gen_bandfilter, gen_highpass

Modul

Foundation

gen_sin_bandpass (: ImageFilter : Frequency, Norm, Mode, Width, Height :)

Erzeugen eines Bandpassfilter in der Form einer Sinusfunktion.

gen_sin_bandpass erzeugt einen rotationsinvarianten Bandpassfilter, in der Form einer Sinusfunktion. Das Maximum der Sinusfunktion wird durch Frequency bestimmt. Die Frequenz wird als Bruchteil der maximalen (horizontalen und vertikalen) Frequenz, die in einem Bild der Größe Width × Height möglich ist, angegeben, d.h. Frequency sollte zwischen 0 und 1 liegen. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit Norm der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls fft_generic verwendet wird, falls Norm = *'n'* verwendet wird. Mit Mode kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls fft_generic verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls fft_image und fft_image_inv zur Filterung verwendet werden, muss Norm = *'none'* und Mode = *'dc_center'* verwendet werden. Falls rft_generic verwendet wird, muss *'rft'* verwendet werden. Im Ursprung ist der Filter immer Null und steigt mit der Sinusfunktion bis Frequency an und fällt dann analog ab. Es wird der Wertebereich von 0 bis π der Sinusfunktion genutzt. Alle Punkte außerhalb werden auf Null gesetzt.

Parameter

▷ **ImageFilter** (output_object) ..... image ⇝ *object : real*
Bandpassfilter als Bild in Frequenzraum.

▷ **Frequency** (input_control) ..... real ⇝ *real*
Abstand des Filtermaximums vom Mittelpunkt.
Defaultwert: 0.1
Wertevorschläge: Frequency ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: Frequency >= 0

▷ **Norm** (input_control) ..... string ⇝ *string*
Normierung des Filters.
Defaultwert: 'none'
Werteliste: Norm ∈ {'none', 'n'}

▷ **Mode** (input_control) ..... string ⇝ *string*
Position der Nullfrequenz im Frequenzraum.
Defaultwert: 'dc_center'
Werteliste: Mode ∈ {'dc_center', 'dc_edge', 'rft'}

▷ **Width** (input_control) . . . . . integer ⇝ *integer*
Breite des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}

▷ **Height** (input_control) . . . . . integer ⇝ *integer*
Höhe des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_sin_bandpass den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fft_image, fft_generic, rft_generic

Nachfolger

convol_fft

Alternativen

gen_std_bandpass

Siehe auch

fft_image_inv, gen_gauss_filter, gen_mean_filter, gen_derivative_filter, gen_bandpass, gen_bandfilter, gen_highpass, gen_lowpass

Modul

Foundation

gen_std_bandpass (: ImageFilter : Frequency, Sigma, Type, Norm, Mode, Width, Height :)

Erzeugen eines Bandpassfilter in der Form einer Gaußmaske oder Sinusfunktion.

gen_std_bandpass erzeugt einen rotationsinvarianten Bandpassfilter, dessen Bandpassverhalten und durch zugehörige Parameter Frequency und Sigma beschrieben werden: Frequency legt den Abstand vom Nullpunkt und Sigma die Ausdehnung fest. Frequency und Sigma werden als Bruchteil der maximalen (horizontalen und vertikalen) Frequenz, die in einem Bild der Größe Width × Height möglich ist, angegeben. Frequency sollten zwischen 0 und 1 liegen. Bei Type = *'gauss'* wird eine Gaußfunktion erzeugt. Sigma ist dabei die Standardabweichung. Bei Type = *'sin'* wird eine Sinusfunktion mit dem Maximum bei Frequency und der Ausdehnung Sigma berechnet. Um eine maximale Effizienz der gesamten Filterung zu erreichen, kann mit Norm der Normierungsfaktor des Filters festgelegt werden. Somit kann eine Normierung in der FFT unterbleiben, falls fft_generic verwendet wird, falls Norm = *'n'* verwendet wird. Mit Mode kann festgelegt werden, wo der Nullfrequenz der FFT liegen soll bzw. ob der Filter für die reelle FFT erzeugt werden soll. Falls fft_generic verwendet wird, kann zur Effizienzsteigerung *'dc_edge'* verwendet werden. Falls fft_image und fft_image_inv zur Filterung verwendet werden, muss Norm = *'none'* und Mode = *'dc_center'* verwendet werden. Falls rft_generic verwendet wird, muss Mode = *'rft'* verwendet werden.

Parameter

▷ **ImageFilter** (output_object) . . . . . image ⇝ *object : real*
Bandpassfilter als Bild in Frequenzraum.

▷ **Frequency** (input_control) . . . . . real ⇝ *real*
Abstand des Filtermaximums vom Mittelpunkt.
Defaultwert: 0.1
Wertevorschläge: Frequency ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: Frequency >= 0

▷ **Sigma** (input_control) ..... real ⇝ *real*
Bandbreite der Filters (Standardabweichung).
Defaultwert: 0.01
Wertevorschläge: Sigma ∈ {0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1.0}
Restriktion: Sigma >= 0

▷ **Type** (input_control) ..... string ⇝ *string*
Art des Filters.
Defaultwert: 'sin'
Werteliste: Type ∈ {'sin', 'gauss'}

▷ **Norm** (input_control) ..... string ⇝ *string*
Normierung des Filters.
Defaultwert: 'none'
Werteliste: Norm ∈ {'none', 'n'}

▷ **Mode** (input_control) ..... string ⇝ *string*
Position der Nullfrequenz im Frequenzraum.
Defaultwert: 'dc_center'
Werteliste: Mode ∈ {'dc_center', 'dc_edge', 'rft'}

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048, 4096, 8192}

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des Bildes (Filters).
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048, 4096, 8192}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_std_bandpass den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fft_image, fft_generic, rft_generic

Nachfolger

convol_fft

Alternativen

gen_sin_bandpass

Siehe auch

fft_image_inv, gen_gauss_filter, gen_mean_filter, gen_derivative_filter, gen_bandpass, gen_bandfilter, gen_highpass, gen_lowpass

Modul

Foundation

optimize_fft_speed (: : Width, Height, Mode :)

Optimierung der Laufzeit der FFT.

optimize_fft_speed bestimmt das Verfahren, das eine optimierte der Laufzeit der FFT für ein Bild der Größe Width × Height erreicht. Die so festgestellten Daten beeinflussen die Verfahren für eine andere Bildgröße nicht, so dass optimize_fft_speed mehrmals mit verschiedenen Parametern für Width und Height aufgerufen werden kann, um optimale Laufzeiten für alle in der Anwendung vorkommenden Bildgrößen zu erreichen. Der Parameter Mode legt die Gründlichkeit der Suche nach dem schnellsten Verfahren fest. Für Mode = *'standard'* wird

eine schnelle Suche verwendet, die typischerweise wenige Sekunden dauert. Das damit bestimmte Verfahren liefert sehr gute Laufzeiten, die aber nicht immer optimal sind. Für Mode = *'patient'* wird eine gründlichere Suche vorgenommen, die typischerweise mehrere Sekunden dauert und in den meisten Fällen zur optimalen Laufzeit führt. Für Mode = *'exhaustive'* wird eine vollständige Suche durchgeführt, die typischerweise mehrere Minuten benötigt und in jedem Fall die optimale Laufzeit liefert. In den meisten Anwendungen liefert Mode = *'standard'* den besten Kompromiss zwischen der Laufzeit der FFT und der Zeit, die zur Bestimmung der optimalen Laufzeit notwendig ist. Die mit optimize_fft_speed berechneten Daten können mit write_fft_optimization_data gespeichert werden und mit read_fft_optimization_data wieder eingelesen werden.

Es ist zu beachten, dass diese Optimierung nur für den jeweiligen Computer durchgeführt wird, auf dem der Operator aufgerufen wird. Die Ergebnisse sind nicht geeignet für die Übertragung und Verwendung auf anderen Computern, es sei denn sie besitzen die gleiche Konfiguration bezüglich Hardware, Software und Treiberversionen.

optimize_fft_speed beeinflusst die Laufzeit der folgenden Operatoren, die die FFT verwenden: fft_generic, fft_image, fft_image_inv, photometric_stereo, sfs_pentland, sfs_mod_lr, sfs_orig_lr, wiener_filter.

Parameter

▷ **Width** (input_control) . . . . . integer ⇝ *integer*
Breite des Bildes, für die die Laufzeit optimiert werden soll.
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048}

▷ **Height** (input_control) . . . . . integer ⇝ *integer*
Höhe des Bildes, für die die Laufzeit optimiert werden soll.
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048}

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Gründlichkeit der Suche nach der optimalen Geschwindigkeit.
Defaultwert: 'standard'
Werteliste: Mode ∈ {'standard', 'patient', 'exhaustive'}

Ergebnis

Sind die Parameter korrekt, dann liefert optimize_fft_speed den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

fft_generic, fft_image, fft_image_inv, wiener_filter, wiener_filter_ni, photometric_stereo, sfs_pentland, sfs_mod_lr, sfs_orig_lr, write_fft_optimization_data

Alternativen

read_fft_optimization_data

Siehe auch

optimize_rft_speed

Modul

Foundation

optimize_rft_speed (: : Width, Height, Mode :)

Optimierung der Laufzeit der reellen FFT.

optimize_rft_speed bestimmt das Verfahren, das eine optimierte der Laufzeit der reellen FFT für ein Bild der Größe Width × Height erreicht. Die so festgestellten Daten beeinflussen die Verfahren für eine andere Bildgröße nicht, so dass optimize_rft_speed mehrmals mit verschiedenen Parametern für Width und Height

aufgerufen werden kann, um optimale Laufzeiten für alle in der Anwendung vorkommenden Bildgrößen zu erreichen. Der Parameter Mode legt die Gründlichkeit der Suche nach dem schnellsten Verfahren fest. Für Mode = *'standard'* wird eine schnelle Suche verwendet, die typischerweise wenige Sekunden dauert. Das damit bestimmte Verfahren liefert sehr gute Laufzeiten, die aber nicht immer optimal sind. Für Mode = *'patient'* wird eine gründlichere Suche vorgenommen, die typischerweise mehrere Sekunden dauert und in den meisten Fällen zur optimalen Laufzeit führt. Für Mode = *'exhaustive'* wird eine vollständige Suche durchgeführt, die typischerweise mehrere Minuten benötigt und in jedem Fall die optimale Laufzeit liefert. In den meisten Anwendungen liefert Mode = *'standard'* den besten Kompromiss zwischen der Laufzeit der reellen FFT und der Zeit, die zur Bestimmung der optimalen Laufzeit notwendig ist. Die mit optimize_rft_speed berechneten Daten können mit write_fft_optimization_data gespeichert werden und mit read_fft_optimization_data wieder eingelesen werden.

Es ist zu beachten, dass diese Optimierung nur für den jeweiligen Computer durchgeführt wird, auf dem der Operator aufgerufen wird. Die Ergebnisse sind nicht geeignet für die Übertragung und Verwendung auf anderen Computern, es sei denn sie besitzen die gleiche Konfiguration bezüglich Hardware, Software und Treiberversionen.

optimize_rft_speed beeinflusst die Laufzeit von rft_generic.

Parameter

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Bildes, für die die Laufzeit optimiert werden soll.
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048}

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des Bildes, für die die Laufzeit optimiert werden soll.
Defaultwert: 512
Wertevorschläge: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576, 1024, 2048}

▷ **Mode** (input_control) ..... string ⇝ *string*
Gründlichkeit der Suche nach der optimalen Geschwindigkeit.
Defaultwert: 'standard'
Werteliste: Mode ∈ {'standard', 'patient', 'exhaustive'}

Ergebnis

Sind die Parameter korrekt, dann liefert optimize_rft_speed den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

rft_generic, write_fft_optimization_data

Alternativen

read_fft_optimization_data

Siehe auch

optimize_fft_speed

Modul

Foundation

phase_correlation_fft (ImageFFT1, ImageFFT2 : ImagePhaseCorrelation : :)

Phasenkorrelation zweier Bilder im Frequenzraum.

phase_correlation_fft führt eine Phasenkorrelation der fouriertransformierten Eingabebilder im Frequenzraum durch. Eine Phasenkorrelation im Frequenzraum bedeutet eine Multiplikation von ImageFFT1 mit dem komplex-konjugierten Bild von ImageFFT2 und die Division durch den Betrag dieses Produktes. Es ist zu

beachten, dass die korrekte Skalierung der Phasenkorrelation im Ortsraum nur erreicht werden kann, falls für die Hintransformation die Operatoren `fft_generic` oder `rft_generic` mit `Norm` = *'none'* und für die Rücktransformation `fft_generic` oder `rft_generic` mit `Norm` = *'n'* verwendet werden. Falls `ImageFFT1` und `ImageFFT2` dieselbe Anzahl Bilder enthalten, werden die entsprechenden Bilder paarweise phasenkorreliert. Ansonsten darf `ImageFFT2` nur ein einziges Bild enthalten. In diesem Fall wird die Phasenkorrelation für jedes Bild von `ImageFFT1` mit `ImageFFT2` ausgeführt.

Achtung

Die Filterung erfolgt immer im Vollbild (Region wird ignoriert).

Parameter

- ▷ **ImageFFT1** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Fouriertransformiertes Eingabebild 1.
- ▷ **ImageFFT2** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Fouriertransformiertes Eingabebild 2.
  Parameteranzahl: ImageFFT2 == ImageFFT1 || ImageFFT2 == 1
- ▷ **ImagePhaseCorrelation** (output_object) ..... image(-array) ⇝ *object :* complex
  Phasenkorrelation der Eingabebilder im Frequenzraum.

Beispiel

```
* Compute the phase correlation of two images.
get_image_size(Image1,Width,Height)
rft_generic(Image1,ImageFFT1,'to_freq','none','complex',Width)
rft_generic(Image2,ImageFFT2,'to_freq','none','complex',Width)
phase_correlation_fft(ImageFFT1,ImageFFT2,PhaseCorrelationFFT)
rft_generic(PhaseCorrelationFFT,PhaseCorrelation,'from_freq','n', \
            'real',Width)
* Determine the translation between the two images.
local_max_sub_pix (PhaseCorrelation, 'facet', 1, 0.02, Row, Column)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `phase_correlation_fft` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

`fft_generic`, `fft_image`, `rft_generic`

Nachfolger

`fft_generic`, `fft_image_inv`, `rft_generic`

Alternativen

`correlation_fft`

Literatur

C. D. Kuglin, D. C. Hines: „The Phase Correlation Image Alignment Method“; IEEE International Conference on Cybernetics and Society; pp. 163-165; 1975.

Modul

Foundation

phase_deg (ImageComplex : ImagePhase : :)

Phase eines komplexen Bildes in Grad.

phase_deg berechnet die Phase eines komplexen Bildes im Winkelmaß. Die Berechnung erfolgt nach folgender Formel:

$$phase = \frac{90}{\pi}\text{atan2}(Imaginärteil, Realteil) \quad .$$

Der Inhalt von ImagePhase ist also der halbe Phasenwinkel. Auf negative Phasenwinkel wird 180 addiert.

Parameter

- ▷ **ImageComplex** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Eingabebild im Frequenzraum.
- ▷ **ImagePhase** (output_object) ..... image(-array) ⇝ *object :* direction
  Phase des Bildes im Winkelmaß.

Beispiel

```
read_image(&Image,"monkey");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
phase_deg(FFT,&Phase);
disp_image(Phase,WindowHandle);
```

Ergebnis

Ist der Pixeltyp korrekt, dann liefert phase_deg den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

fft_image, fft_generic, rft_generic

Nachfolger

disp_image

Alternativen

phase_rad

Siehe auch

fft_image_inv

Modul

Foundation

phase_rad (ImageComplex : ImagePhase : :)

Phase eines komplexen Bildes im Bogenmaß.

phase_rad berechnet die Phase eines komplexen Bildes in Bogenmaß. Die Berechnung erfolgt nach folgender Formel:

$$phase = \text{atan2}(Imaginärteil, Realteil) \quad .$$

Parameter

▷ **ImageComplex** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* complex
Eingabebild im Frequenzraum.

▷ **ImagePhase** (output_object) . . . . . image(-array) ⇝ *object :* real
Phase des Bildes in Bogenmaß.

Beispiel

```
read_image(&Image,"monkey");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
phase_rad(FFT,&Phase);
disp_image(Phase,WindowHandle);
```

Ergebnis

Ist der Pixeltyp korrekt, dann liefert `phase_rad` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`fft_image`, `fft_generic`, `rft_generic`

Nachfolger

`disp_image`

Alternativen

`phase_deg`

Siehe auch

`fft_image_inv`, `fft_generic`, `rft_generic`

Modul

Foundation

power_byte (Image : PowerByte : :)

Powerspektrum eines komplexen Bildes.

`power_byte` berechnet aus dem Real- und Imaginärteil der Fouriertransformierten (vgl. `fft_image`) das Powerspektrum, d.h. die Beträge der Fouriertransformierten. Das Ergebnisbild ist vom Typ byte. Die Berechnung erfolgt nach folgender Formel:

$$\sqrt{Realteil^2 + Imaginärteil^2} \quad .$$

Es ist zu beachten, dass resultierende Grauwerte mit einem Wert über 255 bei diesem Grauwert abgeschnitten werden, da das Ergebnisbild den Typ *'byte'* hat.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
Eingabebild im Frequenzraum.

▷ **PowerByte** (output_object) ..... image(-array) ⇝ *object :* byte
Powerspektrum des Bildes.

Beispiel

```
read_image(&Image,"monkey");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
power_byte(FFT,&Power);
disp_image(Power,WindowHandle);
```

Ergebnis

Ist der Typ des Bildes korrekt, dann liefert `power_byte` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`fft_image`, `fft_generic`, `rft_generic`, `convol_fft`, `convol_gabor`

Nachfolger

`disp_image`

Alternativen

`abs_image`, `convert_image_type`, `power_real`, `power_ln`

Siehe auch

`fft_image`, `fft_generic`, `rft_generic`

Modul

Foundation

power_ln (Image : ImageResult : :)

Powerspektrum eines komplexen Bildes.

`power_ln` berechnet aus dem Real- und Imaginärteil der Fouriertransformierten (vgl. `fft_image`) das Powerspektrum, d.h. die Beträge der Fouriertransformierten. Zusätzlich wird von dem Ergebnis der natürliche Logarithmus berechnet. Das Ergebnisbild ist vom Typ real. Die Berechnung erfolgt nach folgender Formel:

$$\ln(\sqrt{Realteil^2 + Imaginärteil^2}) \ .$$

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
Eingabebild im Frequenzraum.

▷ **ImageResult** (output_object) ..... image(-array) ⇝ *object :* real
Powerspektrum des Bildes.

Beispiel

```
read_image(&Image,"monkey");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
power_ln(FFT,&Power);
disp_image(Power,WindowHandle);
```

Ergebnis

Ist der Typ des Bildes korrekt, dann liefert `power_ln` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`fft_image`, `fft_generic`, `rft_generic`, `convol_fft`, `convol_gabor`

Nachfolger

`disp_image`, `convert_image_type`, `scale_image`

Alternativen

`abs_image`, `convert_image_type`, `power_real`, `power_byte`

Siehe auch

`fft_image`, `fft_generic`, `rft_generic`

Modul

Foundation

power_real (Image : ImageResult : :)

Powerspektrum eines komplexen Bildes.

`power_real` berechnet aus dem Real- und Imaginärteil der Fouriertransformierten (vgl. `fft_image`) das Powerspektrum, d.h. die Beträge der Fouriertransformierten. Das Ergebnisbild ist vom Typ real. Die Berechnung erfolgt nach folgender Formel:

$$\sqrt{Realteil^2 + Imaginärteil^2} \quad .$$

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* complex
  Eingabebild im Frequenzraum.
- ▷ **ImageResult** (output_object) ..... image(-array) ⇝ *object :* real
  Powerspektrum des Bildes.

Beispiel

```
read_image(&Image,"monkey");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
power_real(FFT,&Power);
disp_image(Power,WindowHandle);
```

Ergebnis

Ist der Typ des Bildes korrekt, dann liefert `power_real` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`fft_image`, `fft_generic`, `rft_generic`, `convol_fft`, `convol_gabor`

Nachfolger

`disp_image`, `convert_image_type`, `scale_image`

Alternativen

`abs_image`, `convert_image_type`, `power_byte`, `power_ln`

Siehe auch

`fft_image`, `fft_generic`, `rft_generic`

Modul

Foundation

`read_fft_optimization_data ( : : FileName : )`

Laden von FFT-Laufzeit-Optimierungsdaten aus Datei.

`read_fft_optimization_data` lädt Daten zur Optimierung der Laufzeit der FFT aus der durch `FileName` angegebenen Datei. Die Optimierungsdaten müssen zuvor mit `optimize_fft_speed` bestimmt und mit `write_fft_optimization_data` gespeichert worden sein. Falls die gespeicherten Daten für die später zu verwendenden Bildgrößen berechnet worden sind, erübrigt sich somit ein erneuter Aufruf von `optimize_fft_speed`. Es ist zu beachten, dass die Daten nur auf demselben Rechner verwendet werden sollten, auf dem sie mit `optimize_fft_speed` berechnet worden sind. Falls dies nicht beachtet wird, werden die Laufzeiten nicht optimal sein. Außerdem ist zu beachten, dass Optimierungsdaten, die mit Standard HALCON erzeugt worden sind, mit Parallel HALCON nicht verwendet werden können (und umgekehrt).

`read_fft_optimization_data` beeinflusst die Laufzeit der folgenden Operatoren, die die FFT verwenden: `fft_generic`, `fft_image`, `fft_image_inv`, `sfs_pentland`, `sfs_mod_lr`, `sfs_orig_lr`, `wiener_filter`.

Parameter

▷ **`FileName`** (input_control) . . . . . filename.read ⇝ *string*
Dateiname der Optimierungsdaten.
Defaultwert: 'fft_opt.dat'

Ergebnis

Sind die Parameter korrekt, dann liefert `read_fft_optimization_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

fft_generic, fft_image, fft_image_inv, rft_generic, wiener_filter, wiener_filter_ni, photometric_stereo, sfs_pentland, sfs_mod_lr, sfs_orig_lr

Alternativen

optimize_fft_speed, optimize_rft_speed

Siehe auch

write_fft_optimization_data

Modul

Foundation

rft_generic (Image : ImageFFT : Direction, Norm, ResultType, Width :)

Schnelle reelle Fouriertransformation.

rft_generic berechnet die schnelle Fouriertransformation des Eingabebildes Image. Dabei wird im Gegensatz zu fft_generic, fft_image und fft_image_inv ausgenutzt, dass das Eingabebild der Hintransformation ein reellwertiges Bild (d.h. kein komplexes Bild) ist. In diesem Fall besitzt das komplexe Ausgabebild eine Redundanz. Die Werte in der rechten Hälfte des komplexen Bildes sind die komplexen Konjugierten der Werte an der entsprechenden Position in der linken Hälfte des Bildes. Entsprechend kann Laufzeit und Speicher eingespart werden, indem nur die linke Hälfte des komplexen Bildes berechnet und ausgegeben wird.

Der Parameter ResultType erlaubt es, bei der Rücktransformation (Direction = *'from_freq'*) den Ergebnisbildtyp festzulegen. Bei der Hintransformation (Direction = *'to_freq'*) muss ResultType auf *'complex'* gesetzt werden.

Der Parameter Direction legt fest, ob in den Frequenzraum oder aus dem Frequenzraum zurück transformiert werden soll. Für Direction = *'to_freq'* muss das Eingabebild ein reellwertiger Typ sein, d.h. es darf kein komplexes Bild als Eingabe verwendet werden. Es werden alle Bildtypen unterstützt, die sich in ein reelles Bild transformieren lassen. In diesem Fall ist die Ausgabe ein komplexes Bild der Dimension $(w/2+1) \times h$, wobei w und h die Breite und Höhe des Eingabebildes sind. In diesem Modus wird als Exponent der Transformation -1 verwendet (siehe fft_generic). Für Direction = *'from_freq'* muss das Eingabebild ein komplexes Bild sein. In diesem Fall kann nicht aus der Größe des Eingabebildes entschieden werden, welche Größe das Ausgabebild besitzen soll. Hierzu muss Width korrekt gesetzt werden, d.h. auf $2w-2$ oder auf $2w-1$, wobei w die Breite des komplexen Eingabebildes ist. In diesem Modus wird als Exponent der Transformation 1 verwendet.

Mit Norm kann die gewünschte Normierung angegeben werden. Zulässige Werte sind *'none'*, *'sqrt'* und *'n'*. In jedem Fall ist auf die konsistente Verwendung der Parameter zu achten. D.h., die Normierungsfaktoren bei der Hin- und Rücktransformation müssen aufmultipliziert wh ergeben.

Achtung

Die Berechnung erfolgt immer für das gesamte Bild, d.h., der Definitionsbereich des Eingabebildes wird ignoriert.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex

Eingabebild im Ortsraum.

▷ **ImageFFT** (output_object) ..... image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex

Fouriertransformiertes Bild.

▷ **Direction** (input_control) ..... string ⇝ *string*

Hin- oder Rücktransformation.
Defaultwert: 'to_freq'
Werteliste: Direction ∈ {'to_freq', 'from_freq'}

▷ **Norm** (input_control) ..... string ⇝ *string*

Normierung des Ergebnisbildes.
Defaultwert: 'sqrt'
Werteliste: Norm ∈ {'none', 'sqrt', 'n'}

▷ **ResultType** (input_control) ..... string ⇝ *string*
Bildtyp des Ausgabebildes.
Defaultwert: 'complex'
Werteliste: `ResultType` ∈ {'complex', 'byte', 'int1', 'int2', 'uint2', 'int4', 'real', 'direction', 'cyclic'}

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Bildes, für die die Laufzeit optimiert werden soll.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768, 1024, 2048}

Ergebnis

Sind alle Parameterwerte korrekt, dann liefert `rft_generic` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`optimize_rft_speed`, `read_fft_optimization_data`

Nachfolger

`convol_fft`, `correlation_fft`, `phase_correlation_fft`, `convert_image_type`, `power_byte`, `power_real`, `power_ln`, `phase_deg`, `phase_rad`

Alternativen

`fft_generic`, `fft_image`, `fft_image_inv`

Modul

Foundation

serialize_fft_optimization_data (: : : SerializedItemHandle)

Serialisieren von FFT-Laufzeit-Optimierungsdaten.

`serialize_fft_optimization_data` serialisiert die mit `optimize_fft_speed` bestimmten Daten zur Optimierung der Laufzeit der FFT (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_fft_optimization_data` in eine Datei geschrieben werden. Die serialisierten Daten werden in dem Handle `SerializedItemHandle` zurückgegeben und können mit `deserialize_fft_optimization_data` wieder deserialisiert werden.

Parameter

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameter korrekt, dann liefert `serialize_fft_optimization_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

optimize_fft_speed, optimize_rft_speed

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_fft_optimization_data

Siehe auch

fft_generic, fft_image, fft_image_inv, wiener_filter, wiener_filter_ni, photometric_stereo, sfs_pentland, sfs_mod_lr, sfs_orig_lr, deserialize_fft_optimization_data

Modul

Foundation

write_fft_optimization_data (: : FileName :)

Speichern von FFT-Laufzeit-Optimierungsdaten in Datei.

write_fft_optimization_data speichert die mit optimize_fft_speed bestimmten Daten zur Optimierung der Laufzeit der FFT in der durch FileName angegebenen Datei. Die Daten können mit read_fft_optimization_data wieder eingelesen werden.

Parameter

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Dateiname der Optimierungsdaten.
Defaultwert: 'fft_opt.dat'

Ergebnis

Sind die Parameter korrekt, dann liefert write_fft_optimization_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

optimize_fft_speed, optimize_rft_speed

Siehe auch

fft_generic, fft_image, fft_image_inv, wiener_filter, wiener_filter_ni, photometric_stereo, sfs_pentland, sfs_mod_lr, sfs_orig_lr, read_fft_optimization_data

Modul

Foundation

10.4 Farbe

apply_color_trans_lut (Image1, Image2, Image3 : ImageResult1, ImageResult2, ImageResult3 : ColorTransLUTHandle :)

Farbraumtransformation mit der vorab generierten Look-Up-Tabelle.

apply_color_trans_lut transformiert ein 3-kanaliges Bild von RGB in den anderen Farbraum und umgekehrt mit der vorab generierten Look-Up-Tabelle. Die drei Kanäle werden als drei getrennte Bilder sowohl ein- als auch ausgegeben.

Die weiteren Informationen zu den einzelnen Farbraumtransformationen können aus den Beschreibungen der Operatoren trans_from_rgb und trans_to_rgb entnommen werden.

Parameter

- ▷ **Image1** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild (Kanal 1).
- ▷ **Image2** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild (Kanal 2).
- ▷ **Image3** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild (Kanal 3).
- ▷ **ImageResult1** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Farbtransformiertes Ausgabebild (Kanal 1).
- ▷ **ImageResult2** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Farbtransformiertes Ausgabebild (Kanal 2).
- ▷ **ImageResult3** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Farbtransformiertes Ausgabebild (Kanal 3).
- ▷ **ColorTransLUTHandle** (input_control) ..... color_trans_lut ⇝ *handle*
  Handle der Lookup-Tabelle für die Farbtransformation.

Ergebnis

`apply_color_trans_lut` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind und die Eingabebilder erfolgreich transformiert werden konnten. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`create_color_trans_lut`

Nachfolger

`clear_color_trans_lut`

Alternativen

`trans_from_rgb`, `trans_to_rgb`

Siehe auch

`create_color_trans_lut`, `clear_color_trans_lut`

Modul

Foundation

cfa_to_rgb (CFAImage : RGBImage : CFAType, Interpolation :)

Umwandlung eines einkanaligen Farbmosaikbildes in ein RGB-Bild.

`cfa_to_rgb` wandelt ein einkanaliges Farbmosaikbild `CFAImage` in ein RGB-Bild `RGBImage` um. Farbmosaikbilder werden typischerweise von Ein-Chip-CCD-Kameras erzeugt. Die Transformation von Farbmosaikbild in RGB-Bild erfolgt typischerweise auf der Kamera selbst oder wird vom Gerätetreiber des verwendeten Framegrabbers durchgeführt. In manchen Fällen wird allerdings vom Gerätetreiber das Farbmosaikbild ohne Verarbeitung weitergereicht. In diesem Fall wird typischerweise im jeweiligen HALCON-Framegrabber-Interface die Konversion in ein RGB-Bild durchgeführt. Der Operator `cfa_to_rgb` bietet sich also hauptsächlich dann zur Verwendung an, wenn die Bilder nicht mit dem HALCON-Framegrabber-Interface (`grab_image` oder `grab_image_async`) eingezogen werden, sondern mit einer selbstgeschriebenen Funktionalität, die das Bild mit `gen_image1` oder `gen_image1_extern` an HALCON übergibt.

Bei Ein-Chip-CCD-Kameras sorgt ein Farbmosaikfilter vor dem Sensor für die Erzeugung (unterabgetasteter) Farbinformation. Der am häufigsten verwendete Farbmosaikfilter ist der sog. Bayer-Filter. Die Farbfilter vor den Sensorelementen des CCD-Chips haben hierbei folgende Anordnung:

G	B	G	B	G	B	···
R	G	R	G	R	G	···
G	B	G	B	G	B	···
R	G	R	G	R	G	···
⋮	⋮	⋮	⋮	⋮	⋮	⋱

Jeder Grauwert des Eingabebildes `CFAImage` entspricht der Helligkeit des Pixels hinter dem entsprechenden Farbfilter. In der obigen Anordnung entspricht also das Pixel (0,0) einem Grünwert, das Pixel (0,1) einem Blauwert. Die Anordnung des Bayer-Filters wird vollständig bestimmt durch die ersten zwei Elemente der ersten Bildzeile, welche mit `CFAType` festgelegt werden kann. Dies erlaubt insbesondere die korrekte Umwandlung von Farbmosaikbildern, die aus einem größeren Bild ausgeschnitten worden sind (z.B. mit `crop_part` oder `crop_rectangle1`).

Der Algorithmus, der zur Berechnung der RGB-Werte verwendet wird, wird mit `Interpolation` festgelegt. Für `Interpolation` = *'bilinear'* wird eine bilineare Interpolation durchgeführt. Dieser Algorithmus ist sehr schnell, führt aber typischerweise an starken Kanten zu „reißverschlussartigen" Artefakten und Farbartefakten. Für `Interpolation` = *'bilinear_dir'* wird eine modifizierte Variante der bilinearen Interpolation durchgeführt, die zu weniger reißverschlussartigen Artefakten, speziell bei horizontalen oder vertikalen Kanten im Bild, führen kann. Das Ergebnis kann aber immer noch Farbartefakte an starken Kanten enthalten. Die Laufzeit dieses Algorithmus' ist nur wenig länger als die der bilinearen Interpolation. Für `Interpolation` = *'bilinear_enhanced'* wird eine verbesserte Version der bilinearen Interpolation verwendet, die in den meisten Fällen deutlich weniger reißverschlussartige Artefakte und Farbartefakte erzeugt als die anderen beiden bilinearen Algorithmen. Die Laufzeit dieses Algorithmus ist deutlich länger als die der anderen beiden Algorithmen.

Falls *'mmx_enable'* auf *'true'* gesetzt wird (und die verwendete Hardware SIMD-Technologie unterstützt), wird die Berechnung für byte-Bilder für `Interpolation` = *'bilinear'* und `Interpolation` = *'bilinear_dir'* intern unter Verwendung von SIMD-Technologie durchgeführt.

`cfa_to_rgb` kann für `Interpolation` = *'bilinear'* und `Interpolation` = *'bilinear_dir'* auf OpenCL-Geräten ausgeführt werden. Dabei sollte die Breite des Eingabebildes ein vielfaches von vier (für byte-Bilder) bzw. zwei (für uint2-Bilder) betragen, da die Berechnung sonst um ein vielfaches langsamer ist.

Parameter

- ▷ **`CFAImage`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **`RGBImage`** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / uint2
  Ausgabebild.
- ▷ **`CFAType`** (input_control) ..... string ⇝ *string*
  Art des Farbmosaiks.
  Defaultwert: 'bayer_gb'
  Werteliste: `CFAType` ∈ {'bayer_gb', 'bayer_gr', 'bayer_bg', 'bayer_rg'}
- ▷ **`Interpolation`** (input_control) ..... string ⇝ *string*
  Art der Interpolation.
  Defaultwert: 'bilinear'
  Werteliste: `Interpolation` ∈ {'bilinear', 'bilinear_dir', 'bilinear_enhanced'}

Ergebnis

`cfa_to_rgb` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result', <Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

gen_image1_extern, gen_image1, grab_image

Nachfolger

decompose3

Siehe auch

trans_from_rgb

Modul

Foundation

clear_color_trans_lut (: : ColorTransLUTHandle :)

Löscht die Look-Up-Tabelle für die Farbraumtransformation.

clear_color_trans_lut löscht die Look-Up-Tabelle, die mit dem Operator create_color_trans_lut erstellt wurde. Nach dem Aufruf des Operators clear_color_trans_lut wird das Handle ColorTransLUTHandle ungültig.

Parameter

▷ **ColorTransLUTHandle** (input_control) ..... color_trans_lut ⇝ *handle*
Handle der Look-Up-Tabelle für die Farbtransformation.

Ergebnis

clear_color_trans_lut liefert den Wert TRUE, falls der Speicher der Look-Up-Tabelle erfolgreich gelöscht wurde. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_color_trans_lut, apply_color_trans_lut

Siehe auch

create_color_trans_lut, apply_color_trans_lut

Modul

Foundation

create_color_trans_lut (: : ColorSpace, TransDirection,
NumBits : ColorTransLUTHandle)

Generiert eine Look-Up-Tabelle für die Farbraumtransformation von RGB Farbraum in die weiteren Farbräume.

create_color_trans_lut generiert eine Look-Up-Tabelle für die Farbraumtransformation aus dem RGB-Farbraum in andere Farbräume und umgekehrt.

Für die Transformationen werden 3-kanalige 8-Bit Farbbilder benötigt, d.h. jeder Kanal muss vom Typ byte sein. Besitzen die Eingabebilder unterschiedliche Domänen, wird nur die gemeinsame Schnittmenge aller drei Eingabebilder transformiert.

Der Parameter ColorSpace gibt den Ausgangs- oder den Zielfarbraum an. Mit dem Parameter TransDirection wird die Richtung der Transformation (’von RGB’ oder ’nach RGB’) festgelegt.

Die Verwendung von der Look-Up-Tabelle für die Farbraumtransformation beschleunigt viele Farbraumtransformationen im Vergleich zu den Operatoren `trans_from_rgb` und `trans_to_rgb`. Bei Transformationen aus / nach den Farbräumen *'yiq'*, *'argyb'*, *'yuv'*, *'ciexyz'* oder *'i1i2i3'* wird empfohlen die Operatoren `trans_from_rgb` und `trans_to_rgb` zu verwenden.

Für weitere Informationen zu den Farbraumtransformationen siehe die Beschreibungen der Operatoren `trans_from_rgb` und `trans_to_rgb`.

Parameter

▷ **`ColorSpace`** (input_control) . . . . . string ⇝ *string*
Gewünschte Farbtransformation.
Defaultwert: 'hsv'
Werteliste: `ColorSpace` ∈ {'yiq', 'yuv', 'argyb', 'ciexyz', 'ciexyz4', 'cielab', 'cielchab', 'cieluv', 'cielchuv', 'hls', 'hsi', 'hsv', 'ciexyz3', 'ciexyz4', 'hls', 'ihs', 'i1i2i3', 'lms'}

▷ **`TransDirection`** (input_control) . . . . . string ⇝ *string*
Gewünschte Richtung der Farbtransformation.
Defaultwert: 'from_rgb'
Werteliste: `TransDirection` ∈ {'from_rgb', 'to_rgb'}

▷ **`NumBits`** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Bits des Eingangsbildes.
Defaultwert: 8
Werteliste: `NumBits` ∈ {8}

▷ **`ColorTransLUTHandle`** (output_control) . . . . . color_trans_lut ⇝ *handle*
Handle der Look-Up-Tabelle für Farbtransformation.

Ergebnis

`create_color_trans_lut` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind und die Look-Up-Tabelle erzeugt werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`apply_color_trans_lut`

Siehe auch

`apply_color_trans_lut`, `clear_color_trans_lut`

Modul

Foundation

`gen_principal_comp_trans` (MultichannelImage : : : Trans, TransInv, Mean, Cov, InfoPerComp)

Berechnung der Transformationsmatrix der Hauptachsentransformation (Principal Components Analysis) von mehrkanaligen Bildern.

`gen_principal_comp_trans` berechnet die Transformationsmatrix für eine Hauptachsentransformation (Principal Components Analysis) von mehrkanaligen Bildern. Diese Transformation ist z.B. bei Bildern nützlich, die mit dem Thematic Mapper des Landsat Satelliten aufgenommen wurden. Da die einzelnen Kanäle zum Teil starke Korrelationen aufweisen, ist es wünschenswert, die Eingabebilder in Bilder zu transformieren, die möglichst wenig Korrelation aufweisen. Dies ist zum einen nützlich, um Speicherplatz zu sparen, da Bildkomponenten mit geringem Informationsgehalt vernachlässigt werden können, und zum anderen im Hinblick auf eine spätere Klassifikation.

Die Funktion gen_principal_comp_trans nimmt ein oder mehrere Eingabebilder in MultichannelImage und berechnet die Transformationsmatrix Trans für die Hauptachsentransformation, sowie ihre Inverse TransInv. Alle Eingabebilder müssen dieselbe Anzahl von Kanälen haben. Die Hauptachsentransformation wird aus allen Bildern gemeinsam berechnet. Somit erlaubt es gen_principal_comp_trans, die Statistik mehrerer Bilder auszunutzen.

Wenn n die Anzahl der Kanäle ist, sind Trans und TransInv Matrizen der Dimension $n \times (n+1)$, die eine affine Transformation der mehrkanaligen Grauwerte beschreiben. Sie können mit dem Operator linear_trans_color verwendet werden, um ein mehrkanaliges Bild zu transformieren. Als Information werden in Mean und Cov die mittleren Grauwerte der einzelnen Kanäle sowie die $n \times n$ Kovarianzmatrix der Grauwerte zurückgegeben. In dem Ausgabeparameter InfoPerComp wird der jeweilige relative Informationsgehalt der einzelnen Komponenten zurückgegeben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **MultichannelImage** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
  Mehrkanaliges Bild.
- ▷ **Trans** (output_control) ..... real-array ⇝ *real*
  Transformationsmatrix zur Berechnung der PCA.
- ▷ **TransInv** (output_control) ..... real-array ⇝ *real*
  Transformationsmatrix zur Berechnung der inversen PCA.
- ▷ **Mean** (output_control) ..... real-array ⇝ *real*
  Mittelwert der jeweiligen Kanäle.
- ▷ **Cov** (output_control) ..... real-array ⇝ *real*
  Kovarianzmatrix der Kanäle.
- ▷ **InfoPerComp** (output_control) ..... real-array ⇝ *real*
  Informationsgehalt der transformierten Kanäle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_principal_comp_trans den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

linear_trans_color

Alternativen

principal_comp

Modul

Foundation

linear_trans_color (Image : ImageTrans : TransMat :)

Affine Transformation der Farbwerte eines mehrkanaligen Bildes.

linear_trans_color führt eine affine Transformation der Farbwerte des mehrkanaligen Bildes Image aus und gibt das Ergebnis in ImageTrans zurück. Die affine Abbildung der Farbwerte wird durch die Transformationsmatrix TransMat beschrieben. Wenn n die Anzahl der Kanäle in Image und m die Anzahl der Kanäle in ImageTrans sind, ist TransMat eine homogene $m \times (n+1)$ Matrix, die zeilenweise abgespeichert ist.

Homogen heißt, dass in der linken $m \times n$ Untermatrix von TransMat eine lineare Transformation der Farbwerte enthalten ist, während in der letzten Spalte von TransMat ein konstanter Versatz der Farbwerte beschrieben wird. Die Transformationsmatrix wird typischerweise mit gen_principal_comp_trans erzeugt. Insbesondere kann ein Bild auf seine ersten m Hauptachsen projiziert werden, indem die ersten $m \times (n+1)$ Werte der Transformation, die von gen_principal_comp_trans zurückgeliefert wird, selektiert werden. Die Transformation kann aber auch direkt spezifiziert werden. So kann eine Transformation von RGB nach YIQ, die durch folgende affine Abbildung gegeben ist

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.599 & -0.276 & -0.324 \\ 0.214 & -0.522 & 0.309 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 127.5 \\ 127.5 \end{pmatrix}$$

durch folgende Wahl von TransMat erreicht werden:

$$[0.299, 0.587, 0.114, 0.0, 0.599, -0.276, -0.324, 127.5, 0.214, -0.522, 0.309, 127.5]$$

Hierbei ist zu beachten, dass die obige Transformation unnormalisiert ist, d.h. die Werte können den Bereich $[0, 255]$ verlassen. Die Transformation *'yiq'* in trans_from_rgb skaliert die Zeilen der Matrix (außer dem konstanten Versatz) noch geeignet.

Um einen Verlust von Information zu vermeiden, liefert linear_trans_color als Ausgabe ein Bild vom Typ real zurück. Falls ein anderer Bildtyp gewünscht ist, kann dieser mit convert_image_type erzeugt werden.

Achtung

linear_trans_color kann auf OpenCL-Geräten ausgeführt werden, sofern das Eingangsbild nicht mehr als neun Kanäle besitzt und auf ein Bild transformiert wird, das nicht mehr als drei Kanäle enthält. Da das Ergebnis mit einfacher Genauigkeit berechnet wird, kann es von dem mit der CPU berechneten Ergebnis abweichen.

Parameter

- ▷ **Image** (input_object) ..... multichannel-image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
  Mehrkanaliges Grauwertbild.
- ▷ **ImageTrans** (output_object) ..... multichannel-image(-array) ⇝ *object :* real
  Mehrkanaliges Ausgabebild.
- ▷ **TransMat** (input_control) ..... real-array ⇝ *real*
  Transformationsmatrix für die Farbwerte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert linear_trans_color den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

gen_principal_comp_trans

Nachfolger

convert_image_type

Alternativen

principal_comp, trans_from_rgb, trans_to_rgb

Modul

Foundation

principal_comp (MultichannelImage : PCAImage : : InfoPerComp)

Hauptachsentransformation (Principal Components Analysis) von mehrkanaligen Bildern.

principal_comp führt eine Hauptachsentransformation (Principal Components Analysis) von mehrkanaligen Bildern durch. Diese Transformation ist z.B. bei Bildern nützlich, die mit dem Thematic Mapper des Landsat Satelliten aufgenommen wurden. Da die einzelnen Kanäle zum Teil starke Korrelationen aufweisen, ist es wünschenswert, die Eingabebilder in Bilder zu transformieren, die möglichst wenig Korrelation aufweisen. Dies ist zum einen nützlich, um Speicherplatz zu sparen, da Bildkomponenten mit geringem Informationsgehalt vernachlässigt werden können, und zum anderen im Hinblick auf eine spätere Klassifikation.

Die Funktion principal_comp nimmt ein Eingabebild MultichannelImage und transformiert dieses mittels der Hauptachsentransformation in ein Ausgabebild PCAImage, das die gleiche Anzahl von Komponenten besitzt. In dem Ausgabeparameter InfoPerComp wird der jeweilige relative Informationsgehalt der einzelnen Komponenten zurückgegeben.

Achtung

principal_comp kann auf OpenCL-Geräten ausgeführt werden, sofern das Eingangsbild nicht mehr als acht Kanäle besitzt. Da das Ergebnis mit einfacher Genauigkeit berechnet wird, kann es von dem mit der CPU berechneten Ergebnis abweichen.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **MultichannelImage** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
  Mehrkanaliges Grauwertbild.
- ▷ **PCAImage** (output_object) ..... multichannel-image ⇝ *object :* real
  Mehrkanaliges Ausgabebild.
- ▷ **InfoPerComp** (output_control) ..... real-array ⇝ *real*
  Informationsgehalt der ausgegebenen Kanäle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert principal_comp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

gen_principal_comp_trans

Siehe auch

linear_trans_color

Modul

Foundation

rgb1_to_gray (RGBImage : GrayImage : :)

Berechnung eines Grauwertbildes aus einem RGB-Bild.

rgb1_to_gray transformiert ein RGB-Bild in ein Grauwertbild. Die drei Kanäle werden in den ersten drei Kanälen des Eingabebildes übergeben. Rot ist der erste, Grün der zweite und Blau der dritte Kanal. Die Transformation erfolgt nach folgender Formel:

$$k = 0.299r + 0.587g + 0.114b \ .$$

Falls eines der Eingabebilder in RGBImage ein Einkanal-Bild ist wird dessen Referenz einfach in die Ausgabe GrayImage kopiert.

Parameter

- ▷ **RGBImage** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Dreikanaliges RGB-Bild.
- ▷ **GrayImage** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / real
  Grauwertbild.

Beispiel

```
* Transformation from rgb to gray
read_image(Image,'patras')
dev_display(Image)
rgb1_to_gray(Image,GrayImage)
dev_display(GrayImage)
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

compose3

Alternativen

trans_from_rgb, rgb3_to_gray

Modul

Foundation

rgb3_to_gray (ImageRed, ImageGreen, ImageBlue : ImageGray : :)

Berechnung eines Grauwertbildes aus drei Bildern (RGB).

rgb3_to_gray transformiert ein RGB-Bild in ein Grauwertbild. Die drei Kanäle werden als drei getrennte Bilder übergeben. Die Transformation erfolgt nach folgender Formel:

$$k = 0.299r + 0.587g + 0.114b \ .$$

Parameter

- ▷ **ImageRed** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild (Rot-Kanal).
- ▷ **ImageGreen** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild (Grün-Kanal).
- ▷ **ImageBlue** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild (Blau-Kanal).
- ▷ **ImageGray** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / real
  Grauwertbild.

Beispiel

```
* Transformation from rgb to gray
read_image(Image,'patras')
dev_display(Image)
decompose3(Image, ImageR, ImageG, ImageB)
rgb3_to_gray(ImageR,ImageG,ImageB,GrayImage)
dev_display(GrayImage)
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`decompose3`

Alternativen

`rgb1_to_gray`, `trans_from_rgb`

Modul

Foundation

`trans_from_rgb` `( ImageRed, ImageGreen, ImageBlue : ImageResult1, ImageResult2, ImageResult3 : ColorSpace : )`

Transformation vom RGB- in einen anderen Farbraum.

`trans_from_rgb` transformiert ein Bild von RGB in den angegebenen Farbraum (`ColorSpace`). Die drei Kanäle werden als drei getrennte Bilder sowohl ein- als auch ausgegeben.

Die Bildtypen byte, uint2, int4 und real werden unterstützt. Dabei sollten real-Bilder nur Werte zwischen 0 und 1 erhalten. Andernfalls sind die Ergebnisse der Transformationen möglicherweise nicht sinnvoll.

Entsprechend des Datentyps der beteiligten Bilder werden noch Skalierungen vorgenommen, wenn eine der folgende Bedingungen zutrifft:

- Der Wertebereich der Farbraumgrößen wird bei byte- und uint2-Bildern generell auf den vollen Bereich von [0..255] bzw. [0..65535] abgebildet. Bei vorzeichenbehafteten Farbraumgrößen (z.B. bei CIELab) liegt aus diesem Grund der Nullpunkt nicht in der Mitte des Bereiches.
- Hue Werte werden durch Winkel im Bereich von $[0..2\pi[$ repräsentiert, welche für die jeweiligen Bildtypen unterschiedlich kodiert werden:
  - *byte*-Bilder bilden den Winkelbereich auf $[0..255]$ ab.
  - *uint2/int4*-Bilder werden in Winkelminuten $[0..21600[$ kodiert, außer für *'cielchab'* und *'cielchuv'* für *int4*-Bilder, für die sie als Winkelsekunden $[0..1296000[$ kodiert werden.
  - *real*-Bilder werden im Bogenmaß $[0..2\pi[$ kodiert, außer für *'cielchab'* und *'cielchuv'* für die die Normen ISO 11664-4:2008 und ISO 11664-5:2009 vorschreiben, dass der Hue in Grad angegeben wird.
- Sättigungswerte (Saturation) werden in Prozent angegeben, welche für die jeweiligen Bildtypen unterschiedlich kodiert werden:
  - *byte*-Bilder bilden die Sättigung auf $[0..255]$ ab.
  - *uint2/int4*-Bilder bilden die Sättigung auf $[0..10000]$ ab.
  - *real*-Bilder bilden die Sättigung auf $[0..1]$ ab.

Unterstützt werden die unten folgenden Transformationen. Dabei gilt zu beachten, dass die angegebenen Wertebereiche sich auf eine Skalierung der RGB Werte auf das Intervall [0;1] beziehen. Der Wertebereich für die jeweiligen Bildtypen ergibt sich durch Skalierung mit den entsprechenden Wertebereichen. Aufgrund verschiedener Genauigkeiten können die mit den unten angegebenen Gleichungen erhaltenen Werte leicht von den Resultaten des Operators abweichen.

'yiq'

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.599 & -0.276 & -0.324 \\ 0.214 & -0.522 & 0.309 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Wertebereich: $Y \in [0;1], I \in [-0.599;0.599], Q \in [-0.522;0.522]$

'yuv'

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= 0.493(B - Y) \\ V &= 0.877(R - Y) \end{aligned}$$

Es gilt zu beachten, dass Y, U und V nicht unabhängig voneinander sind.

Wertebereich: $Y \in [0.0;1.0], U \in [-0.436798;0.436798], V \in [-0.614777;0.614777]$

'argyb'

$$\begin{pmatrix} A \\ Rg \\ Yb \end{pmatrix} = \begin{pmatrix} 0.30 & 0.59 & 0.11 \\ 0.50 & -0.50 & 0.00 \\ 0.25 & 0.25 & -0.50 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Wertebereich: $A \in [0;1], Rg \in [-0.5;0.5], Yb \in [-0.5;0.5]$

'ciexyz'

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Die verwendeten Grundfarben entsprechen sRGB bzw. der CIE Rec. 709. Als Weißpunkt wird D65 verwendet.

Verwendete Grundfarben (x, y, z)**:** *rot*:= $\begin{pmatrix} 0.6400 \\ 0.3300 \\ 0.0300 \end{pmatrix}$, *grün*:= $\begin{pmatrix} 0.3000 \\ 0.6000 \\ 0.1000 \end{pmatrix}$,

blau:= $\begin{pmatrix} 0.1500 \\ 0.0600 \\ 0.7900 \end{pmatrix}$, *weiß*$_{65}$:= $\begin{pmatrix} 0.3127 \\ 0.3290 \\ 0.3583 \end{pmatrix}$

Wertebereich: $X \in [0;0.950456], \quad Y \in [0;1], \quad Z \in [0;1.088754]$

'hls'

```
Min := min([R, G, B])
Max := max([R, G, B])
L := (Min + Max) / 2
if (Max == Min)
   H := 0
   S := 0
else
   if (L > 0.5)
      S := (Max - Min) / (2 - Max - Min)
   else
      S := (Max - Min) / (Max + Min)
   endif
   if (R == Max)
      H := ((G - B) / (Max - Min)) * rad(60)
```

```
      elseif (G == Max)
         H := (2 + (B - R) / (Max - Min)) * rad(60)
      elseif (B == Max)
         H := (4 + (R - G) / (Max - Min)) * rad(60)
      endif
   endif
```

Wertebereich: $H \in [0; 2\pi[, L \in [0; 1], S \in [0; 1]$

'hsi'

$$\begin{pmatrix} M1 \\ M2 \\ I1 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{pmatrix} H \\ S \\ I \end{pmatrix} = \begin{pmatrix} \operatorname{atan2}(M2, M1) \\ \sqrt{M1^2 + M2^2} \\ \frac{I1}{\sqrt{3}} \end{pmatrix}$$

Wertebereich: $H \in [0; 2\pi[, S \in [0; \sqrt{\frac{2}{3}}], I \in [0; 1]$

'hsv'

```
Min := min([R, G, B])
Max := max([R, G, B])
V := Max
if (Max == Min)
   S := 0
   H := 0
else
   S := (Max - Min) / Max
   if (R == Max)
      H := ((G - B) / (Max - Min)) * rad(60)
   elseif (G == Max)
      H := (2 + (B - R) / (Max - Min)) * rad(60)
   elseif (B == Max)
      H := (4 + (R - G) / (Max - Min)) * rad(60)
   endif
endif
```

Wertebereich: $H \in [0; 2\pi[, S \in [0; 1], V \in [0; 1]$

'ihs'

```
Min := min([R, G, B])
Max := max([R, G, B])
I := (R + G + B) / 3
if (I == 0)
   H := 0
   S := 1
else
   S := 1 - Min / I
   if (S == 0)
        H := 0
   else
      A := (R + R - G - B) / 2
      B := (R - G) * (R - G) + (R - B) * (G - B)
      C := sqrt(B)
      if (C == 0)
         H := 0
      else
```

```
            H := acos(A / C)
         endif
         if (B > G)
            H := rad(360) - H
         endif
      endif
   endif
```

Wertebereich: $I \in [0;1], H \in [0;2\pi[, S \in [0;1]$

'cielab'

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{aligned} L &= 116 f(\frac{Y}{Y_w}) - 16 \\ a &= 500(f(\frac{X}{X_w}) - f(\frac{Y}{Y_w})) \\ b &= 200(f(\frac{Y}{Y_w}) - f(\frac{Z}{Z_w})) \end{aligned}$$

mit $\begin{array}{lcll} f(t) & = & t^{\frac{1}{3}}, & t > (\frac{24}{116})^3 \\ f(t) & = & \frac{841}{108} * t + \frac{16}{116}, & \text{sonst} \end{array}$

Schwarzpunkt B**:** $(R_b, G_b, B_b) = (0, 0, 0)$

Weißpunkt $W = (R_w, G_w, B_w)$**, je nach Bildtyp:** $W_{byte} = (255, 255, 255), W_{uint2} = (2^{16} - 1, 2^{16} - 1, 2^{16} - 1)$,
$W_{int4} = (2^{31} - 1, 2^{31} - 1, 2^{31} - 1), W_{real} = (1.0, 1.0, 1.0)$

Wertebereich: $L \in [0;100], a \in [-86.1813; 98.2352], b \in [-107.8617; 94.4758]$

(Für byte/uint2-Bildtypen auf den maximalen Grauwert skaliert. Im Falle von int4-Bildtypen sind L und a auf den maximalen Grauwert und b auf den minimalen Grauwert skaliert, so dass der Nullpunkt erhalten bleibt.)

'cielchab'

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{aligned} L &= 116 f(\frac{Y}{Y_w}) - 16 \\ a &= 500(f(\frac{X}{X_w}) - f(\frac{Y}{Y_w})) \\ b &= 200(f(\frac{Y}{Y_w}) - f(\frac{Z}{Z_w})) \\ C_{ab} &= \sqrt{a^2 + b^2} \\ h_{ab} &= \operatorname{atan2}(b, a) \end{aligned}$$

mit $\begin{array}{lcll} f(t) & = & t^{\frac{1}{3}}, & t > (\frac{24}{116})^3 \\ f(t) & = & \frac{841}{108} * t + \frac{16}{116}, & \text{sonst} \end{array}$

h_{ab} liegt zwischen 0° und 90° falls a und b positiv sind, zwischen 90° und 180° falls a negativ ist und b positiv ist, zwischen 180° und 270° falls a und b negativ sind und zwischen 270° und 360° falls a positiv ist und b negativ ist.

Schwarzpunkt B**:** $(R_b, G_b, B_b) = (0, 0, 0)$

Weißpunkt $W = (R_w, G_w, B_w)$**, je nach Bildtyp:** $W_{byte} = (255, 255, 255), W_{uint2} = (2^{16} - 1, 2^{16} - 1, 2^{16} - 1)$,
$W_{int4} = (2^{31} - 1, 2^{31} - 1, 2^{31} - 1), W_{real} = (1.0, 1.0, 1.0)$
Wertebereich: $L \in [0; 100], C \in [0; 133.8086], h \in [0; 360[$

(Für byte/uint2-Bildtypen auf den maximalen Grauwert skaliert. Im Fall von int4-Bildtypen sind L und C auf den maximalen Grauwert skaliert und h wird in Winkelsekunden angegeben.)

'cieluv'

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{aligned} u' &= 4X/(X + 15Y + 3Z) \\ v' &= 9X/(X + 15Y + 3Z) \\ u'_w &= 4X_w/(X_w + 15Y_w + 3Z_w) \\ v'_w &= 9X_w/(X_w + 15Y_w + 3Z_w) \\ L &= 116f(\frac{Y}{Y_w}) - 16 \\ u &= 13L(u' - u'_w) \\ v &= 13L(v' - v'_w) \end{aligned}$$

mit $\begin{array}{lcll} f(t) & = & t^{\frac{1}{3}}, & t > (\frac{24}{116})^3 \\ f(t) & = & \frac{841}{108} * t + \frac{16}{116}, & \text{sonst} \end{array}$
Schwarzpunkt B**:** $(R_b, G_b, B_b) = (0, 0, 0)$
Weißpunkt $W = (R_w, G_w, B_w)$**, je nach Bildtyp:** $W_{byte} = (255, 255, 255), W_{uint2} = (2^{16} - 1, 2^{16} - 1, 2^{16} - 1)$,
$W_{int4} = (2^{31} - 1, 2^{31} - 1, 2^{31} - 1), W_{real} = (1.0, 1.0, 1.0)$
Wertebereich: $L \in [0; 100], u \in [-83.0774; 175.0148], v \in [-134.1008; 107.3923]$

(Für byte/uint2-Bildtypen auf den maximalen Grauwert skaliert. Im Falle von int4-Bildtypen sind L und u auf den maximalen Grauwert und v auf den minimalen Grauwert skaliert, so dass der Nullpunkt erhalten bleibt.)

'cielchuv'

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{aligned} u' &= 4X/(X + 15Y + 3Z) \\ v' &= 9X/(X + 15Y + 3Z) \\ u'_w &= 4X_w/(X_w + 15Y_w + 3Z_w) \\ v'_w &= 9X_w/(X_w + 15Y_w + 3Z_w) \\ L &= 116f(\frac{Y}{Y_w}) - 16 \\ u &= 13L(u' - u'_w) \\ v &= 13L(v' - v'_w) \\ C_{uv} &= \sqrt{u^2 + v^2} \\ h_{uv} &= \operatorname{atan2}(v, u) \end{aligned}$$

mit $\begin{array}{lcll} f(t) & = & t^{\frac{1}{3}}, & t > (\frac{24}{116})^3 \\ f(t) & = & \frac{841}{108} * t + \frac{16}{116}, & \text{sonst} \end{array}$
Schwarzpunkt B**:** $(R_b, G_b, B_b) = (0, 0, 0)$

Weißpunkt $W = (R_w, G_w, B_w)$**, je nach Bildtyp:** $W_{byte} = (255, 255, 255), W_{uint2} = (2^{16} - 1, 2^{16} - 1, 2^{16} - 1)$,
$W_{int4} = (2^{31} - 1, 2^{31} - 1, 2^{31} - 1), W_{real} = (1.0, 1.0, 1.0)$
Wertebereich: $L \in [0; 100], C \in [0; 179.0402], h \in [0; 360[$

(Für byte/uint2-Bildtypen auf den maximalen Grauwert skaliert. Im Falle von int4-Bildtypen sind L und u auf den maximalen Grauwert und v auf den minimalen Grauwert skaliert, so dass der Nullpunkt erhalten bleibt.)

'i1i2i3'

$$\begin{pmatrix} I1 \\ I2 \\ I3 \end{pmatrix} = \begin{pmatrix} 0.333 & 0.333 & 0.333 \\ 1.0 & 0.0 & -1.0 \\ -0.5 & 1.0 & -0.5 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Wertebereich: $I1 \in [0; 1], I2 \in [-1; 1], I3 \in [-1; 1]$

'ciexyz2'

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.620 & 0.170 & 0.180 \\ 0.310 & 0.590 & 0.110 \\ 0.000 & 0.066 & 1.020 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Wertebereich: $X \in [0; 0.970], Y \in [0; 1.010], Z \in [0; 1.086]$

'ciexyz3'

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.618 & 0.177 & 0.205 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.056 & 0.944 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Wertebereich: $X \in [0; 1], Y \in [0; 1], Z \in [0; 1]$

'ciexyz4'

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.476 & 0.299 & 0.175 \\ 0.262 & 0.656 & 0.082 \\ 0.020 & 0.161 & 0.909 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Verwendete Grundfarben (x, y, z)**:** $red := \begin{pmatrix} 0.628 \\ 0.346 \\ 0.026 \end{pmatrix}$, $green := \begin{pmatrix} 0.268 \\ 0.588 \\ 0.144 \end{pmatrix}$, $blue := \begin{pmatrix} 0.150 \\ 0.070 \\ 0.780 \end{pmatrix}$, $white_{65} := \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

Wertebereich: $X \in [0; 0.951], \quad Y \in [0; 1], \quad Z \in [0; 1.088]$

'lms'

$$\begin{pmatrix} L \\ M \\ S \end{pmatrix} = \begin{pmatrix} 0.390469 & 0.549910 & 0.008901 \\ 0.070925 & 0.963118 & 0.001358 \\ 0.023143 & 0.128013 & 0.935976 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Dies stellt konzeptuell eine Transformation von RGB nach CIE XYZ (siehe *'ciexyz'* oben) gefolgt von einer Transformation von CIE XYZ nach LMS dar.

Wertebereich: $L \in [0; 0.949280], \quad M \in [0; 1.035401], \quad S \in [0; 1.087132]$

Achtung

Die OpenCL-Implementierung der cielab-Transformation für int4-Bilder ist aufgrund der abweichenden numerischen Präzision bei der Berechnung etwas ungenauer als die reine C Variante.

Parameter

- ▷ **ImageRed** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Eingabebild (Rot-Kanal).
- ▷ **ImageGreen** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Eingabebild (Grün-Kanal).
- ▷ **ImageBlue** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Eingabebild (Blau-Kanal).
- ▷ **ImageResult1** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Farbtransformiertes Ausgabebild (Kanal 1).
- ▷ **ImageResult2** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Farbtransformiertes Ausgabebild (Kanal 2).
- ▷ **ImageResult3** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Farbtransformiertes Ausgabebild (Kanal 3).
- ▷ **ColorSpace** (input_control) . . . . . string ⇝ *string*
  Gewünschte Farbtransformation.
  Defaultwert: 'hsv'
  Werteliste: `ColorSpace` ∈ {'cielab', 'cielchab', 'cieluv', 'cielchuv', 'hsv', 'hsi', 'yiq', 'yuv', 'argyb', 'ciexyz', 'ciexyz2', 'ciexyz3', 'ciexyz4', 'hls', 'ihs', 'i1i2i3', 'lms'}

Beispiel

```
* Transformation from rgb to hsv and conversely
read_image(Image,'patras')
dev_display(Image)
decompose3(Image, Image1, Image2, Image3)
trans_from_rgb(Image1,Image2,Image3,ImageH,ImageS,ImageV,'hsv')
trans_to_rgb(ImageH,ImageS,ImageV,ImageR,ImageG,ImageB,'hsv')
compose3(ImageR,ImageG,ImageB,Multichannel)
dev_display(Multichannel)
```

Ergebnis

`trans_from_rgb` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result', <Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`decompose3`

Nachfolger

`compose3`

Alternativen

`linear_trans_color`, `rgb1_to_gray`, `rgb3_to_gray`

Siehe auch

`trans_to_rgb`

Literatur

ITU-R BT.470-6: „Conventional Television Systems“, 1998.

ISO 11664-4:2008: „Colorimetry — Part 4: CIE 1976 L*a*b* Colour Space“, 2008.

ISO 11664-5:2009: „Colorimetry — Part 5: CIE 1976 L*u*v* Colour space and u',v' uniform chromaticity scale diagram“, 2009.

Modul

Foundation

trans_to_rgb (ImageInput1, ImageInput2, ImageInput3 : ImageRed, ImageGreen, ImageBlue : ColorSpace :)

Transformation von einem Farbraum nach RGB.

trans_to_rgb transformiert ein Bild aus einem angegebenen Farbraum (ColorSpace) nach RGB. Die drei Kanäle werden als drei getrennte Bilder sowohl ein- als auch ausgegeben.

Die Bildtypen byte, uint2, int4 und real werden unterstützt. Die Werte im angegebenen Farbraum ColorSpace müssen so vorliegen, wie sie aus einer Transformation mit trans_from_rgb entstanden wären. Andernfalls sind die Ergebnisse der Transformationen möglicherweise nicht sinnvoll.

Dies beinhaltet Skalierungen im Falle bestimmter Transformationen und/oder Bildtypen:

- Es wird vorausgesetzt, dass der Definitionsbereich der Farbraumgrößen bei byte- und uint2-Bildern generell auf den vollen Bereich von [0..255] bzw. [0..65535] abgebildet ist. Dies beinhaltet eine Verschiebung im Falle vorzeichenbehafteter Größen, so dass bei vorzeichenbehafteten Farbraumgrößen (z.B. bei CIELab) der Nullpunkt nicht unbedingt in der Mitte des Bereiches liegt.
- Hue Werte werden durch Winkel im Bereich $[0..2\pi[$ repräsentiert, welche für die jeweiligen Bildtypen unterschiedlich kodiert werden:
  - *byte*-Bilder bilden den Winkelbereich auf $[0..255]$ ab.
  - *uint2/int4*-Bilder werden in Winkelminuten $[0..21600[$ kodiert, außer für *'cielchab'* und *'cielchuv'* für *int4*-Bilder, für die sie als Winkelsekunden $[0..1296000[$ kodiert werden.
  - *real*-Bilder werden im Bogenmaß $[0..2\pi[$ kodiert, außer für *'cielchab'* und ISO 11664-5:2009 *'cielchuv'* für die die Normen ISO 11664-4:2008 und vorschreiben, dass der Hue in Grad angegeben wird.
- Sättigungswerte (Saturation) werden in Prozent angegeben, welche für die jeweiligen Bildtypen unterschiedlich kodiert werden:
  - *byte*-Bilder bilden die Sättigung auf $[0..255]$ ab.
  - *uint2/int4*-Bilder bilden die Sättigung auf $[0..10000]$ ab.
  - *real*-Bilder bilden die Sättigung auf $[0..1]$ ab.

Unterstützt werden die unten folgenden Transformationen. Dabei gilt zu beachten, dass die angegebenen Wertebereiche sich auf eine Skalierung der RGB Werte auf das Intervall [0;1] beziehen. Der Wertebereich für die jeweiligen Bildtypen ergibt sich durch Skalierung mit den entsprechenden Wertebereichen. Aufgrund verschiedener Genauigkeiten können die mit den unten angegebenen Gleichungen erhaltenen Werte leicht von den Resultaten des Operators abweichen.

'yiq'

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 & 0.947 & 0.624 \\ 1.0 & -0.267 & -0.650 \\ 1.0 & -1.109 & 1.709 \end{pmatrix} \begin{pmatrix} Y \\ I \\ Q \end{pmatrix}$$

Definitionsbereich: $Y \in [0;1], I \in [-0.599;0.599], Q \in [-0.522;0.522]$

'yuv'

$$\begin{aligned} R &= Y + \frac{1.0}{0.877}V \\ G &= Y - \frac{0.114}{0.493 * 0.587}U - \frac{0.299}{0.877 * 0.587}V \\ B &= Y + \frac{1.0}{0.493}U \end{aligned}$$

Es gilt zu beachten, dass Y, U und V nicht unabhängig voneinander sind.

Definitionsbereich: $Y \in [0.0;1.0], U \in [-0.436798;0.436798], V \in [-0.614777;0.614777]$

'argyb'

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.00 & 1.29 & 0.22 \\ 1.00 & -0.71 & 0.22 \\ 1.00 & 0.29 & -1.78 \end{pmatrix} \begin{pmatrix} A \\ Rg \\ Yb \end{pmatrix}$$

Definitionsbereich: $A \in [0;1], Rg \in [-0.5;0.5], Yb \in [-0.5;0.5]$

'ciexyz'

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Definitionsbereich: $X \in [0;0.950456], \quad Y \in [0;1], \quad Z \in [0;1.088754]$

'cielab'

$$\begin{aligned} f_y &= (L+16)/116 \\ f_x &= a/500 + f_y \\ f_z &= b/200 - f_y \end{aligned}$$

$$\begin{aligned} X &= X_w * f_x^3, && f_x > \tfrac{24}{116} \\ X &= (f_x - \tfrac{16}{116}) * X_w * \tfrac{108}{841}, && \text{sonst} \end{aligned}$$

$$\begin{aligned} Y &= Y_w * f_y^3, && f_y > \tfrac{24}{116} \\ Y &= (f_y - \tfrac{16}{116}) * Y_w * \tfrac{108}{841}, && \text{sonst} \end{aligned}$$

$$\begin{aligned} Z &= Z_w * f_z^3, && f_z > \tfrac{24}{116} \\ Z &= (f_z - \tfrac{16}{116}) * Z_w * \tfrac{108}{841}, && \text{sonst} \end{aligned}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Schwarzpunkt B**:** $(R_b, G_b, B_b) = (0,0,0)$

Weißpunkt $W = (R_w, G_w, B_w)$**, je nach Bildtyp:** $W_{byte} = (255,255,255), W_{uint2} = (2^{16}-1, 2^{16}-1, 2^{16}-1)$,
$W_{int4} = (2^{31}-1, 2^{31}-1, 2^{31}-1), W_{real} = (1.0, 1.0, 1.0)$

Definitionsbereich: $L \in [0;100], a \in [-86.1813;98.2352], b \in [-107.8617;94.4758]$

(Für byte/uint2-Bildtypen auf den maximalen Grauwert skaliert. Im Falle von int4-Bildtypen sind L und a auf den maximalen Grauwert und b auf den minimalen Grauwert skaliert, so dass der Nullpunkt erhalten bleibt.)

'cielchab'

$$\begin{aligned} f_y &= (L+16)/116 \\ f_x &= a/500 + f_y \\ f_z &= b/200 - f_y \end{aligned}$$

$$\begin{aligned} X &= X_w * f_x^3, && f_x > \tfrac{24}{116} \\ X &= (f_x - \tfrac{16}{116}) * X_w * \tfrac{108}{841}, && \text{sonst} \end{aligned}$$

$$\begin{aligned} Y &= Y_w * f_y^3, && f_y > \tfrac{24}{116} \\ Y &= (f_y - \tfrac{16}{116}) * Y_w * \tfrac{108}{841}, && \text{sonst} \end{aligned}$$

$$\begin{aligned} Z &= Z_w * f_z^3, && f_z > \tfrac{24}{116} \\ Z &= (f_z - \tfrac{16}{116}) * Z_w * \tfrac{108}{841}, && \text{sonst} \end{aligned}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Schwarzpunkt B**:** $(R_b, G_b, B_b) = (0, 0, 0)$

Weißpunkt $W = (R_w, G_w, B_w)$**, je nach Bildtyp:** $W_{byte} = (255, 255, 255), W_{uint2} = (2^{16} - 1, 2^{16} - 1, 2^{16} - 1)$,
$W_{int4} = (2^{31} - 1, 2^{31} - 1, 2^{31} - 1), W_{real} = (1.0, 1.0, 1.0)$

Definitionsbereich: $L \in [0; 100], C \in [0; 133.8086], h \in [0; 360[$

(Für byte/uint2-Bildtypen auf den maximalen Grauwert skaliert. Im Fall von int4-Bildtypen sind L und C auf den maximalen Grauwert skaliert und h wird in Winkelsekunden angegeben.)

'cieluv'

$$
\begin{array}{rcll}
f_y & = & (L+16)/116 & \\
Y & = & Y_w * f_y^3, & f_y > \frac{24}{116} \\
Y & = & (f_y - \frac{16}{116}) * Y_w * \frac{108}{841}, & \text{sonst} \\
u' & = & \frac{u}{13L} + u'_w & \\
v' & = & \frac{u}{13L} + v'_w & \\
x & = & \frac{9u'}{6u'-16v'+12} & \\
y & = & \frac{4u'}{6u'-16v'+12} & \\
X & = & \frac{xY}{y} & \\
Z & = & \frac{(1-x-y)Y}{y} &
\end{array}
$$

$$
\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}
$$

Schwarzpunkt B**:** $(R_b, G_b, B_b) = (0, 0, 0)$

Weißpunkt $W = (R_w, G_w, B_w)$**, je nach Bildtyp:** $W_{byte} = (255, 255, 255), W_{uint2} = (2^{16} - 1, 2^{16} - 1, 2^{16} - 1)$,
$W_{int4} = (2^{31} - 1, 2^{31} - 1, 2^{31} - 1), W_{real} = (1.0, 1.0, 1.0)$

Definitionsbereich: $L \in [0; 100], u \in [-83.0774; 175.0148], v \in [-134.1008; 107.3923]$

(Für byte/uint2-Bildtypen auf den maximalen Grauwert skaliert. Im Falle von int4-Bildtypen sind L und u auf den maximalen Grauwert und v auf den minimalen Grauwert skaliert, so dass der Nullpunkt erhalten bleibt.)

'cielchuv'

$$
\begin{array}{rcl}
u & = & C_{uv} \cos h_{uv} \\
v & = & C_{uv} \sin h_{uv}
\end{array}
$$

$$
\begin{array}{rcll}
f_y & = & (L+16)/116 & \\
Y & = & Y_w * f_y^3, & f_y > \frac{24}{116} \\
Y & = & (f_y - \frac{16}{116}) * Y_w * \frac{108}{841}, & \text{sonst} \\
u' & = & \frac{u}{13L} + u'_w & \\
v' & = & \frac{u}{13L} + v'_w & \\
x & = & \frac{9u'}{6u'-16v'+12} & \\
y & = & \frac{4u'}{6u'-16v'+12} & \\
X & = & \frac{xY}{y} & \\
Z & = & \frac{(1-x-y)Y}{y} &
\end{array}
$$

$$
\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}
$$

Schwarzpunkt B**:** $(R_b, G_b, B_b) = (0, 0, 0)$

Weißpunkt $W = (R_w, G_w, B_w)$**, je nach Bildtyp:** $W_{byte} = (255, 255, 255), W_{uint2} = (2^{16} - 1, 2^{16} - 1, 2^{16} - 1)$,
$W_{int4} = (2^{31} - 1, 2^{31} - 1, 2^{31} - 1), W_{real} = (1.0, 1.0, 1.0)$

Definitionsbereich: $L \in [0; 100], C \in [0; 179.0402], h \in [0; 360[$

(Für byte/uint2-Bildtypen auf den maximalen Grauwert skaliert. Im Fall von int4-Bildtypen sind L und C auf den maximalen Grauwert skaliert und h wird in Winkelsekunden angegeben.)

'hls'

```
 Hi := floor(H/rad(60))
Hf := H/rad(60) - Hi
if (L <= 0.5)
   Max := L * (S + 1)
else
   Max := L + S - (L * S)
endif
Min := 2 * L - Max
if (S == 0)
   R := L
   G := L
B := L
else
  if (Hi == 0)
     R := Max
     G := Min + Hf * (Max - Min)
     B := Min
  elseif (Hi == 1)
     R := Min + (1 - Hf) * (Max - Min)
     G := Max
     B := Min
  elseif (Hi == 2)
     R := Min
     G := Max
     B := Min + Hf * (Max - Min)
  elseif (Hi == 3)
     R := Min
     G := Min + (1 - Hf) * (Max - Min)
     B := Max
  elseif (Hi == 4)
     R := Min + Hf * (Max - Min)
     G := Min
     B := Max
  elseif (Hi == 5)
     R := Max
     G := Min
     B := Min + (1 - Hf) * (Max - Min)
  endif
endif
```

Definitionsbereich: $H \in [0; 2\pi[, L \in [0; 1], S \in [0; 1]$

'hsi'

$$\begin{aligned} M1 &= S * \cos H \\ M2 &= S * \sin H \\ I1 &= I * \sqrt{3} \end{aligned}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} \\ \frac{-1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} M1 \\ M2 \\ I1 \end{pmatrix}$$

Definitionsbereich: $H \in [0; 2\pi[, S \in [0; \sqrt{\frac{2}{3}}], I \in [0; 1]$

'hsv'

```
if (S == 0)
  R := V
  G := V
```

```
      B := V
   else
      Hi := floor(H/rad(60))
      Hf := H/rad(60) - Hi
      if (Hi == 0)
         R := V
         G := V * (1 - (S * (1 - Hf)))
         B := V * (1 - S)
      elseif (Hi == 1)
         R := V * (1 - (S * Hf))
         G := V
         B := V * (1 - S)
      elseif (Hi == 2)
         R := V * (1 - S)
         G := V
         B := V * (1 - (S * (1 - Hf)))
      elseif (Hi == 3)
         R := V * (1 - S)
         G := V * (1 - (S * Hf))
         B := V
      elseif (Hi == 4)
         R := V * (1 - (S * (1 - Hf)))
         G := V * (1 - S)
         B := V
      elseif (Hi == 5)
         R := V
         G := V * (1 - S)
         B := V * (1 - (S * Hf))
      endif
   endif
```

Definitionsbereich: $H \in [0; 2\pi[, S \in [0; 1], V \in [0; 1]$

'ciexyz4'

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 2.750 & -1.149 & -0.426 \\ -1.118 & 2.026 & 0.033 \\ 0.138 & -0.333 & 1.104 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Definitionsbereich: $X \in [0; 0.951], \quad Y \in [0; 1], \quad Z \in [0; 1.088]$

'lms'

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 2.858334 & -1.628721 & -0.024820 \\ -0.210432 & 1.158402 & 0.000321 \\ -0.041894 & -0.118163 & 1.068973 \end{pmatrix} \begin{pmatrix} L \\ M \\ S \end{pmatrix}$$

Definitionsbereich: $L \in [0; 0.949280], \quad M \in [0; 1.035401], \quad S \in [0; 1.087132]$

Achtung

Die OpenCL-Implementierung der cielab-Transformation für int4-Bilder ist aufgrund der abweichenden numerischen Präzision bei der Berechnung etwas ungenauer als die reine C Variante.

Parameter

- ▷ **ImageInput1** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Eingabebild (1. Kanal).
- ▷ **ImageInput2** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Eingabebild (2. Kanal).
- ▷ **ImageInput3** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Eingabebild (3. Kanal).

▷ **ImageRed** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
Rot-Kanal.

▷ **ImageGreen** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
Grün-Kanal.

▷ **ImageBlue** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
Blau-Kanal.

▷ **ColorSpace** (input_control) . . . . . string ⇝ *string*
Gewünschte Farbtransformation.
Defaultwert: 'hsv'
Werteliste: `ColorSpace` ∈ {'hsi', 'yiq', 'yuv', 'argyb', 'ciexyz', 'ciexyz4', 'cielab', 'cielchab', 'cieluv', 'cielchuv', 'hls', 'hsv', 'lms'}

Beispiel

```
* Transformation from rgb to hsv and conversely
read_image(Image,'patras')
dev_display(Image)
decompose3(Image, Image1, Image2, Image3)
trans_from_rgb(Image1,Image2,Image3,ImageH,ImageS,ImageV,'hsv')
trans_to_rgb(ImageH,ImageS,ImageV,ImageR,ImageG,ImageB,'hsv')
compose3(ImageR,ImageG,ImageB,Multichannel)
dev_display(Multichannel)
```

Ergebnis

`trans_to_rgb` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result', <Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`decompose3`

Nachfolger

`compose3`, `disp_color`

Alternativen

`linear_trans_color`

Siehe auch

`decompose3`

Literatur

ITU-R BT.470-6: „Conventional Television Systems“, 1998.

ISO 11664-4:2008: „Colorimetry — Part 4: CIE 1976 L*a*b* Colour space“, 2008.

ISO 11664-5:2009: „Colorimetry — Part 5: CIE 1976 L*u*v* Colour space and u',v' uniform chromaticity scale diagram“, 2009.

Modul

Foundation

10.5 Geometrische Transformationen

affine_trans_image (Image : ImageAffineTrans : HomMat2D,
Interpolation, AdaptImageSize :)

Anwenden einer beliebigen affinen 2D-Transformation auf Bilder.

affine_trans_image wendet eine beliebige affine 2D-Transformation (Skalierung, Drehung, Verschiebung, Scherung) auf die in Image übergebenen Bilder an und liefert die transformierten Bilder in ImageAffineTrans zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in HomMat2D übergeben wird. Diese kann mit Hilfe der Operatoren hom_mat2d_identity, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_translate etc. aufgebaut werden oder das Resultat von Operatoren wie vector_angle_to_rigid sein.

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, dass die *Zeilen*-Koordinate des Bildes der x-Koordinate und die *Spalten*-Koordinate des Bildes der y-Koordinate des Koordinatensystems entspricht, in dem die Transformationsmatrix definiert wurde. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Der Definitionsbereich des Eingabebildes wird ignoriert, d.h., als das volle Rechteck des Bildes angenommen. Der Definitionsbereich des Ausgabebildes entspricht der Schnittmenge des transformierten Rechtecks und des Rechtecks des Ausgabebildes.

Im allgemeinen fallen transformierte Punkte zwischen die Rasterpunkte der Bildmatrix. Es ist also eine geeignete Interpolation zwischen den Grauwerten des Eingabebildes durchzuführen. Sie dient insbesondere auch dazu, die „Klötzchenbildung“ bei Vergrößerungen bzw. Aliasing-Effekte bei Verkleinerungen zu vermeiden. Die Qualität (bzw. Geschwindigkeit) der Interpolation wird über den Parameter Interpolation gesteuert:

nearest_neighbor Nächste-Nachbar Interpolation. Der Grauwert des nächstgelegenen Bildpunktes wird verwendet (eventuell niedrigere Qualität, sehr schnell).

bilinear Bilineare Interpolation. Der Grauwert wird aus den Grauwerten der vier nächstgelegenen Pixel durch bilineare Interpolation berechnet. Falls die affine Transformation eine Skalierung < 1 enthält, wird keine Glättung durchgeführt, was zu erheblichen Aliasingeffekten führen kann (mittlere Qualität und Laufzeit).

bicubic Bikubische Interpolation. Der Grauwert wird aus den Grauwerten der 4×4 nächstgelegenen Pixel durch bikubische Interpolation berechnet. Falls die affine Transformation eine Skalierung < 1 enthält, wird keine Glättung durchgeführt, was zu erheblichen Aliasingeffekten führen kann (hohe Qualität für Vergrößerungen, langsam).

constant Bilineare Interpolation. Der Grauwert wird aus den Grauwerten der vier nächstgelegenen Pixel durch bilineare Interpolation berechnet. Falls die affine Transformation eine Skalierung < 1 enthält, wird zusätzlich eine Art Mittelwertfilter zur Vermeidung von Aliasingeffekten ausgeführt (mittlere Qualität und Laufzeit).

weighted Bilineare Interpolation. Der Grauwert wird aus den Grauwerten der vier nächstgelegenen Pixel durch bilineare Interpolation berechnet. Falls die affine Transformation eine Skalierung < 1 enthält, wird zusätzlich eine Art Gaußfilter zur Vermeidung von Aliasingeffekten ausgeführt (hohe Qualität, langsam).

Zusätzlich beeinflusst der Systemparameter *'int_zooming'* (siehe set_system) die Genauigkeit der Transformation. Falls *'int_zooming'* auf *'true'* gesetzt wird, wird die Transformation intern bei byte-, int2- und uint2-Bildern mit Festkommaarithmetik durchgeführt, was zu wesentlich kürzeren Laufzeiten führt. Allerdings ist hier die Genauigkeit der berechneten Grauwerte geringer. Für byte-Bilder sind die Differenzen zur genaueren Berechnung (mit *'int_zooming'* = *'false'*) normalerweise kleiner als zwei Grauwerte. Für int2- und uint2-Bilder gilt entsprechend, dass die Grauwertdifferenzen kleiner als 1/128 mal der dynamische Grauwertbereich des Bildes sind. Sie können also bis zu 512 Grauwerte betragen, wenn der volle Grauwertbereich von 16 Bit ausgenutzt wird. Auch die Domäne der Ergebnisbilder kann sich bei Transformationen mit Festkommaarithmetik leicht unterscheiden. Außerdem können, wenn *'int_zooming'* auf *'true'* gesetzt ist, undefinierte Grauwerte am unteren und am rechten Bildrand entstehen wenn ein großer Skalierungsfaktor angewendet wird und außerdem ein großes Ergebnisbild entsteht. Die maximale Breite B_{max} dieses Randes mit undefinierten Grauwerten kann mit der Formel $B_{max} = 0.5 \cdot S \cdot I/2^{15}$ abgeschätzt werden, wobei S den Skalierungsfaktor in einer Dimension und I die Größe des Ausgabebildes in der entsprechenden Dimension darstellt.

Für Real-Bilder hat der Parameter *'int_zooming'* keinen Einfluss, da intern immer mit Gleitkommazahlen gerechnet wird.

Die Größe des Zielbildes wird durch den Parameter AdaptImageSize gesteuert: Bei *'true'* wird die Größe des Zielbildes so gewählt, dass rechts und unten kein Clipping auftritt. Bei *'false'* hat das Zielbild die gleiche Größe wie das Eingabebild. Beachten Sie, dass unabhängig von AdaptImageSize das Zielbild grundsätzlich am linken und oberen Rand beschnitten wird, d.h. alle Bildbereiche, die nach der Transformation negative Koordinaten aufweisen, werden abgeschnitten.

Achtung

Die Region des Eingabebildes wird nicht beachtet.

affine_trans_image verwendet nicht das HALCON-Standard-Koordinatensystem (mit dem Ursprung im Mittelpunkt des linken oberen Pixels) sondern das gleiche Koordinatensystem wie affine_trans_pixel, d.h. der Ursprung liegt in der linken oberen Ecke des linken oberen Pixels. Eine Transformation mit affine_trans_image entspricht für jeden Punkt des Bildes einer Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt, siehe affine_trans_pixel). Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von dem Bild mit Operatoren wie z.B. area_center_gray abgeleitet wurden. Wenn z.B. ein rotationssymmetrisches Bild unter Verwendung von hom_mat2d_rotate um seinen mit area_center_gray bestimmten Schwerpunkt rotiert wird, so liegt das transformierte Bild nicht auf dem Originalbild. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von affine_trans_image die Transformationsmatrix HomMat2D durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
affine_trans_image(Image, ImageAffineTrans, HomMat2DAdapted, 'constant',
'false')
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild.
- ▷ **ImageAffineTrans** (output_object) .... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Transformiertes Ausgabebild.
- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Interpolation** (input_control) ..... string ⇝ *string*
  Art der Interpolation.
  Defaultwert: 'constant'
  Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}
- ▷ **AdaptImageSize** (input_control) ..... string ⇝ *string*
  Anpassung der Zielbildgröße.
  Defaultwert: 'false'
  Werteliste: AdaptImageSize ∈ {'true', 'false'}

Beispiel

```
* Reduction of an image (512 x 512 Pixels) by 50%, rotation
* by 180 degrees and translation to the upper-left corner:

read_image (Image, 'ic0')
hom_mat2d_identity(Matrix1)
hom_mat2d_scale(Matrix1,0.5,0.5,256.0,256.0,Matrix2)
hom_mat2d_rotate(Matrix2,3.14,256.0,256.0,Matrix3)
hom_mat2d_translate(Matrix3,-128.0,-128.0,Matrix4)
affine_trans_image(Image,TransImage,Matrix4,'constant','true')

* Enlarging the part of an image in the interactively
* chosen rectangular window sector:

dev_get_window (WindowHandle)
```

```
draw_rectangle2(WindowHandle,L,C,Phi,L1,L2)
hom_mat2d_identity(Matrix1)
get_system('width',Width)
get_system('height',Height)
hom_mat2d_translate(Matrix1,Height/2.0-L,Width/2.0-C,Matrix2)
hom_mat2d_rotate(Matrix2,3.14-Phi,Height/2.0,Width/2.0,Matrix3)
hom_mat2d_scale(Matrix3,Height/(2.0*L2),Width/(2.0*L1), \
                Height/2.0,Width/2.0,Matrix4)
affine_trans_image(Image,TransImage,Matrix4,'constant','true')
```

Ergebnis

Falls die Matrix HomMat2D eine affine Abbildung darstellt (d.h. keine projektive Abbildung), liefert affine_trans_image den Wert TRUE zurück. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_rotate, hom_mat2d_scale, hom_mat2d_reflect

Alternativen

affine_trans_image_size, zoom_image_size, zoom_image_factor, mirror_image, rotate_image, affine_trans_region

Siehe auch

set_part_style

Modul

Foundation

affine_trans_image_size (Image : ImageAffineTrans : HomMat2D, Interpolation, Width, Height :)

Anwenden einer beliebigen affinen 2D-Transformation auf Bilder mit Angabe der Ausgabebildgröße.

affine_trans_image_size wendet eine beliebige affine 2D-Transformation (Skalierung, Drehung, Verschiebung, Scherung) auf die in Image übergebenen Bilder an und liefert die transformierten Bilder in ImageAffineTrans zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in HomMat2D übergeben wird. Diese kann mit Hilfe der Operatoren hom_mat2d_identity, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_translate etc. aufgebaut werden oder das Resultat von Operatoren wie vector_angle_to_rigid sein.

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, dass die *Zeilen*-Koordinate des Bildes der x-Koordinate und die *Spalten*-Koordinate des Bildes der y-Koordinate des Koordinatensystems entspricht, in dem die Transformationsmatrix definiert wurde. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Die Region des Eingabebildes wird ignoriert, also als das volle Rechteck der Eingabebildmatrix angenommen. Die Region des Ausgabebildes besteht aus dem transformierten Rechteck des Eingabebildes (gegebenenfalls wird das Ausgabebild mit Nullen, d.h. „schwarz“, aufgefüllt).

Im allgemeinen fallen transformierte Punkte zwischen die Rasterpunkte der Bildmatrix. Es ist also eine geeignete Interpolation zwischen den Grauwerten des Eingabebildes durchzuführen. Sie dient insbesondere auch dazu, die „Klötzchenbildung“ bei Vergrößerungen bzw. Aliasing-Effekte bei Verkleinerungen zu vermeiden. Die Qualität (bzw. Geschwindigkeit) der Interpolation wird über den Parameter `Interpolation` gesteuert:

nearest_neighbor Nächste-Nachbar Interpolation. Der Grauwert des nächstgelegenen Bildpunktes wird verwendet (eventuell niedrigere Qualität, sehr schnell).

bilinear Bilineare Interpolation. Der Grauwert wird aus den Grauwerten der vier nächstgelegenen Pixel durch bilineare Interpolation berechnet. Falls die affine Transformation eine Skalierung < 1 enthält, wird keine Glättung durchgeführt, was zu erheblichen Aliasingeffekten führen kann (mittlere Qualität und Laufzeit).

bicubic Bikubische Interpolation. Der Grauwert wird aus den Grauwerten der 4×4 nächstgelegenen Pixel durch bikubische Interpolation berechnet. Falls die affine Transformation eine Skalierung < 1 enthält, wird keine Glättung durchgeführt, was zu erheblichen Aliasingeffekten führen kann (hohe Qualität für Vergrößerungen, langsam).

constant Bilineare Interpolation. Der Grauwert wird aus den Grauwerten der vier nächstgelegenen Pixel durch bilineare Interpolation berechnet. Falls die affine Transformation eine Skalierung < 1 enthält, wird zusätzlich eine Art Mittelwertfilter zur Vermeidung von Aliasingeffekten ausgeführt (mittlere Qualität und Laufzeit).

weighted Bilineare Interpolation. Der Grauwert wird aus den Grauwerten der vier nächstgelegenen Pixel durch bilineare Interpolation berechnet. Falls die affine Transformation eine Skalierung < 1 enthält, wird zusätzlich eine Art Gaußfilter zur Vermeidung von Aliasingeffekten ausgeführt (hohe Qualität, langsam).

Zusätzlich beeinflusst der Systemparameter *'int_zooming'* (siehe `set_system`) die Genauigkeit der Transformation. Falls *'int_zooming'* auf *'true'* gesetzt wird, wird die Transformation intern bei byte-, int2- und uint2-Bildern mit Festkommaarithmetik durchgeführt, was zu wesentlich kürzeren Laufzeiten führt. Allerdings ist hier die Genauigkeit der berechneten Grauwerte geringer. Für byte-Bilder sind die Differenzen zur genaueren Berechnung (mit *'int_zooming'* = *'false'*) normalerweise kleiner als zwei Grauwerte. Für int2- und uint2-Bilder gilt entsprechend, dass die Grauwertdifferenzen kleiner als 1/128 mal der dynamische Grauwertbereich des Bildes sind. Sie können also bis zu 512 Grauwerte betragen, wenn der volle Grauwertbereich von 16 Bit ausgenutzt wird. Auch die Domäne der Ergebnisbilder kann sich bei Transformationen mit Festkommaarithmetik leicht unterscheiden. Außerdem können, wenn *'int_zooming'* auf *'true'* gesetzt ist, undefinierte Grauwerte am unteren und am rechten Bildrand entstehen wenn ein großer Skalierungsfaktor angewendet wird und außerdem ein großes Ergebnisbild entsteht. Die maximale Breite B_{max} dieses Randes mit undefinierten Grauwerten kann mit der Formel $B_{max} = 0.5 \cdot S \cdot I/2^{15}$ abgeschätzt werden, wobei S den Skalierungsfaktor in einer Dimension und I die Größe des Ausgabebildes in der entsprechenden Dimension darstellt. Für Real-Bilder hat der Parameter *'int_zooming'* keinen Einfluss, da intern immer mit Gleitkommazahlen gerechnet wird.

Die Größe des Zielbildes wird durch die Parameter `Width` und `Height` angegeben. Beachten Sie, dass das Zielbild grundsätzlich am linken und oberen Rand beschnitten wird, d.h. alle Bildbereiche, die nach der Transformation negative Koordinaten aufweisen, werden abgeschnitten. Bei geeigneter Wahl der affinen Abbildung, insbesondere der Translation, kann somit gleichzeitig ein Teil des Bildes transformiert und ausgeschnitten werden. Dies ist z.B. bei der Verwendung des Variationsmodells nützlich (siehe `compare_variation_model`), da mit diesem Mechanismus nur die Teile des Bildes, die untersucht werden sollen, transformiert werden.

Achtung

Die Region des Eingabebildes wird nicht beachtet.

`affine_trans_image_size` verwendet nicht das HALCON-Standard-Koordinatensystem (mit dem Ursprung im Mittelpunkt des linken oberen Pixels) sondern das gleiche Koordinatensystem wie `affine_trans_pixel`, d.h. der Ursprung liegt in der linken oberen Ecke des linken oberen Pixels. Eine Transformation mit `affine_trans_image_size` entspricht für jeden Punkt des Bildes einer Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt, siehe `affine_trans_pixel`). Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von dem Bild mit Operatoren wie z.B. `area_center_gray` abgeleitet wurden. Wenn z.B. ein rotationssymmetrisches Bild unter Verwendung von `hom_mat2d_rotate` um seinen mit `area_center_gray` bestimmten Schwerpunkt rotiert wird, so liegt das transformierte Bild nicht auf dem Originalbild. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von `affine_trans_image_size` die Transformationsmatrix `HomMat2D` durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
```

```
affine_trans_image_size(Image, ImageAffineTrans, HomMat2DAdapted,
'constant', Width, Height)
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild.
- ▷ **ImageAffineTrans** (output_object) .... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Transformiertes Ausgabebild.
- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Interpolation** (input_control) ..... string ⇝ *string*
  Art der Interpolation.
  Defaultwert: 'constant'
  Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Ausgabebildes.
  Defaultwert: 640
  Wertevorschläge: `Width` ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768}
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Ausgabebildes.
  Defaultwert: 480
  Wertevorschläge: `Height` ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576}

Ergebnis

Falls die Matrix `HomMat2D` eine affine Abbildung darstellt (d.h. keine projektive Abbildung), liefert `affine_trans_image_size` den Wert TRUE zurück. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`hom_mat2d_identity`, `hom_mat2d_translate`, `hom_mat2d_rotate`, `hom_mat2d_scale`, `hom_mat2d_reflect`

Alternativen

`affine_trans_image`, `zoom_image_size`, `zoom_image_factor`, `mirror_image`, `rotate_image`, `affine_trans_region`

Siehe auch

`set_part_style`

Modul

Foundation

convert_map_type (Map : MapConverted : NewType, ImageWidth :)

Konvertieren von Abbildungsvorschriften in andere Typen.

convert_map_type konvertiert die Eingabeabbildungsvorschrift Map in einen anderen Typ. Der hauptsächliche Nutzen von convert_map_type liegt darin Abbildungsvorschriften vom Typ *'nearest_neighbor'* bzw. *'bilinear'* in zwei Koordinatenbilder (*'coord_map_sub_pix'*) zu konvertieren, da dies der einzige Typ ist, der auf Compute Devices unterstützt wird.

NewType legt die Art der Konversion fest. Die folgenden Werte sind erlaubt:

'coord_map_sub_pix' erzeugt eine Abbildungsvorschrift, die aus einem Vektorfeld besteht, welches die subpixelgenauen Koordinaten für das Mapping jedes Pixels enthält. (Der Typ der Eingabeabbildungsvorschrift muss entweder *'bilinear'* oder *'nearest_neighbor'* sein)

'bilinear' erzeugt eine bilinear interpolierende Abbildungsvorschrift (siehe map_image). (Der Typ der Eingabeabbildungsvorschrift muss *'coord_map_sub_pix'* sein)

'nearest_neighbor' erzeugt eine Nächste-Nachbar Abbildungsvorschrift (siehe map_image). (Der Typ der Eingabeabbildungsvorschrift muss *'coord_map_sub_pix'* sein)

Die Breite der Bilder auf die MapConverted angewendet werden soll muss in ImageWidth übergeben werden. Falls diese Breite identisch mit der Breite von Map ist, kann auch *'map_width'* übergeben werden.

Parameter

- ▷ **Map** (input_object) ..... (multichannel-)image ⇝ *object :* vector_field / int4 / int8 / uint2
  Eingabeabbildungsvorschrift.
- ▷ **MapConverted** (output_object) ..... (multichannel-)image ⇝ *object :* vector_field / int4 / int8 / uint2
  Konvertierte Abbildungsvorschrift.
- ▷ **NewType** (input_control) ..... string ⇝ *string*
  Typ von MapConverted.
  Defaultwert: 'coord_map_sub_pix'
  Werteliste: NewType ∈ {'coord_map_sub_pix', 'bilinear', 'nearest_neighbor'}
- ▷ **ImageWidth** (input_control) ..... integer ⇝ *integer* / string
  Breite der Bilder auf die MapConverted angewandt wird.
  Defaultwert: 'map_width'

Beispiel

```
gen_radial_distortion_map (MapFixed, CamParOriginal, CamParVirtualFixed, \
                           'bilinear')
convert_map_type(MapFixed, MapConverted, 'coord_map_sub_pix', 'map_width')
query_available_compute_devices (DeviceIdentifier)
open_compute_device (DeviceIdentifier[0], DeviceHandle)
activate_compute_device (DeviceHandle)
map_image (GrayImage, MapConverted, ImageRectifiedFixed)
```

Ergebnis

convert_map_type gibt TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_image_to_world_plane_map, gen_radial_distortion_map, gen_arbitrary_distortion_map, gen_grid_rectification_map, find_local_deformable_model

Nachfolger

map_image

Modul

Foundation

map_image (Image, Map : ImageMapped : :)

Transformieren eines Bildes unter Verwendung einer allgemeinen Abbildungsvorschrift.

map_image transformiert ein Bild Image unter Verwendung einer allgemeinen Abbildungsvorschrift Map, die z.B. mit gen_image_to_world_plane_map oder gen_radial_distortion_map erzeugt wurde. Das Mehrkanalbild Map muss dabei folgenden Aufbau besitzen:

Höhe und Breite von Map legen die Größe des Ausgabebildes ImageMapped fest. Die Anzahl der Kanäle in Map legt fest, ob bei der Transformation keine Interpolation oder bilineare Interpolation angewendet werden soll. Besteht Map lediglich aus einem Kanal, so wird bei der Transformation keine Interpolation angewendet. In diesem Kanal (Bildtyp 'int4' bzw. 'int8' in HALCON XL falls der Wertebereich von 'int4' nicht ausreicht) ist für jedes Pixel des Ergebnisbildes ImageMapped die lineare Koordinate desjenigen Pixels im Eingabebild Image gespeichert, von dem der Grauwert übernommen werden soll.

Soll zwischen den Pixeln im Eingabebild bilinear interpoliert werden, so muss Map aus einem Bild mit 5 Kanälen bestehen. Der erste Kanal (Bildtyp 'int4' bzw. 'int8') beschreibt die geometrische Transformation. Die Kanäle 2-5 (Bildtyp 'uint2') bestehen enthalten jeweils die Gewichte der vier benachbarten Pixel für die bilineare Interpolation. Die Gewichte [0...1] sind dabei auf den Wertebereich des 'uint2' Bildes skaliert und besitzen daher ganzzahlige Werte im Bereich von 0 bis 65535. Soll sich die Helligkeit des transformierten Bildes nicht von der des Eingabebildes unterscheiden, so ist darauf zu achten, dass die Summe der vier unskalierten Gewichte jeweils 1 ergibt. In diesem Zusammenhang ist auch darauf zu achten, dass der Wertebereich des Ausgabebildes ImageMapped nicht überschritten wird. Der geometrische Bezug zwischen den Kanälen 2-5 ist durch folgende Skizze ersichtlich:

2	3
4	5

Bezugspunkt der vier Pixel ist dabei jeweils das linke obere Pixel, dessen lineare Koordinate im ersten Kanal gespeichert ist.

Es ist außerdem möglich eine Map zu verwenden die aus einem Vektorfeld des semantischen Typs 'vector_field_absolute' besteht. Dieses enthält die subpixelgenauen Koordinaten der Abbildungsfunktion. Die beiden oben erläuterten Typen können mittels convert_map_type in diesen Typ überführt werden. Dieser Typ ist der einzige Typ, der auf Compute Devices unterstützt wird!

Achtung

Es ist darauf zu achten, dass der Wertebereich des Ausgabebildes ImageMapped nicht überschritten wird.

Aus Laufzeitgründen wird nicht überprüft, ob die linearen Koordinaten im ersten Kanal von Map innerhalb des Eingabebildes liegen. Es muss also darauf geachtet werden, dass diese Bedingung erfüllt ist, da es sonst zum Programmabsturz kommen kann!

map_image wird nur dann automatisch parallelisiert, wenn Map bilineare Interpolation nutzt.

map_image wird nur für eine Abbildungsvorschrift vom Typ 'coord_map_sub_pix' auf einem OpenCL-fähigen Compute Device ausgeführt. Ferner darf das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreiten.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Zu transformierendes Bild.
- ▷ **Map** (input_object) . . . . . (multichannel-)image ⇝ *object :* int4 / int8 / uint2 / vector_field
  Bild mit Transformationsdaten.
- ▷ **ImageMapped** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Transformiertes Bild.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert map_image den Wert TRUE.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

gen_image_to_world_plane_map, gen_radial_distortion_map, convert_map_type

Siehe auch

affine_trans_image, rotate_image

Modul

Foundation

mirror_image (Image : ImageMirror : Mode :)

Spiegeln von Bildern.

mirror_image spiegelt das Bild Image an einer von drei Achsen. Wenn Mode *'row'* ist, wird das Bild vertikal gespiegelt, wenn er *'column'* ist, wird es horizontal gespiegelt, und wenn er *'diagonal'* ist, wird das Bild an der Winkelhalbierenden $x = y$ gespiegelt.

Achtung

mirror_image kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet. Allerdings kann die Ausführung unter OpenCL langsamer sein als auf der CPU, insbesondere für den Modus *'row'*.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Eingabebild.
- ▷ **ImageMirror** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Gespiegeltes Ausgabebild.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Art der Spiegelung.
  Defaultwert: 'row'
  Werteliste: Mode ∈ {'row', 'column', 'diagonal'}

Beispiel

```
read_image(Image,'monkey')
dev_display (Image)
mirror_image(Image,MirrorImage,'row')
dev_display (MirrorImage)
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

hom_mat2d_rotate, hom_mat2d_reflect, affine_trans_image, rotate_image

Siehe auch

rotate_image, hom_mat2d_rotate

Modul

Foundation

polar_trans_image_ext (Image : PolarTransImage : Row, Column, AngleStart, AngleEnd, RadiusStart, RadiusEnd, Width, Height, Interpolation :)

Anwenden einer Polarkoordinatentransformation eines Kreisrings.

polar_trans_image_ext transformiert den durch den Mittelpunkt (Row, Column), die Radien RadiusStart und RadiusEnd und die Winkel AngleStart und AngleEnd definierten Kreisring im Bild Image auf dessen Polarkoordinatendarstellung im Bild PolarTransImage mit den Dimensionen Width × Height.

Das linke obere Pixel im Ausgabebild entspricht immer dem Punkt im Eingabebild, der durch RadiusStart und AngleStart spezifiziert wird. Analog entspricht das rechte untere Pixel im Ausgabebild immer dem Punkt im Eingabebild, der durch RadiusEnd und AngleEnd spezifiziert wird. Im Normalfall (AngleStart < AngleEnd und RadiusStart < RadiusEnd) wird die Polarkoordinatentransformation also in mathematisch positiver Durchlaufrichtung (entgegen dem Uhrzeigersinn) durchgeführt. Dabei liegen die Punkte mit kleineren Radien im Ausgabebild weiter oben. Durch geeignetes Vertauschen der Werte der Parameter (z.B. AngleStart > AngleEnd oder RadiusStart > RadiusEnd) kann jede beliebige Orientierung des Ausgabebildes erreicht werden.

Über den Parameter Interpolation kann dabei zwischen den Interpolationsverfahren *'bilinear'* und *'nearest_neighbor'* gewählt werden. Bei *'nearest_neighbor'* wird der Grauwert des Pixels im Ausgabebild aus dem Grauwert des nächstgelegenen Pixels im Eingabebild bestimmt. Bei *'bilinear'* wird der Grauwert des Pixels im Ausgabebild durch bilineare Interpolation aus den Grauwerten der vier nächstgelegenen Pixel im Eingabebild bestimmt. Der Modus *'bilinear'* liefert qualitativ bessere Ergebnisse, ist allerdings auch langsamer als *'nearest_neighbor'*.

Die Winkel sind frei als reelle Zahlen wählbar, Mittelpunkt und Radien können ebenfalls frei als reelle Zahlen gewählt werden. Sind diese jedoch ganze Zahlen und entspricht die Differenz von RadiusEnd und RadiusStart der Höhe Height des Zielbildes, so wird die Berechnung durch eine optimierte Routine erheblich schneller durchgeführt.

Die Radien und Winkel sind inklusive, d.h. die erste Zeile des Zielbildes entspricht dem Kreis mit dem Radius RadiusStart, die letzte Zeile dem mit dem Radius RadiusEnd. Für Vollkreise, bei denen die Differenz AngleStart - AngleEnd 2π (360 Grad) beträgt, bedeutet dies insbesondere, dass die erste Spalte des Zielbildes gleich der letzten ist.

Um diesem Verhalten vorzubeugen sollte diese Differenz $2\pi(1 - \frac{1}{\texttt{Width}})$ statt 2π betragen.

Der Aufruf:

```
polar_trans_image(Image, PolarTransImage, Row, Column, Width, Height)
```

entspricht dem Aufruf:

```
polar_trans_image_ext(Image, PolarTransImage, Row-0.5, Column-0.5,
6.2831853, 6.2831853/Width, 0, Height-1, Width, Height, 'nearest_neighbor')
```

Der Offset von 0.5 ist notwendig, da polar_trans_image keine Nearest-Neighbor Interpolation durchführt, Radien und Winkel ergeben sich durch die im vorherigen Abschnitt beschriebenen Überlegungen, da diese Werte in polar_trans_image nicht inklusive behandelt werden. Der Startwinkel ist hier größer als der Endwinkel, damit der Kreisring von polar_trans_image_ext genauso wie in polar_trans_image im Uhrzeigersinn durchlaufen wird.

polar_trans_image_ext kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet. Aus numerischen Gründen kann es dabei zu geringfügigen Abweichungen des Ergebnisses gegenüber der Ausführung auf der CPU kommen.

Für Bilder vom Typ *'byte'*, *'int2'* oder *'uint2'* kann mittels des Systemparameters *'int_zooming'* (siehe set_system) zwischen schneller Berechnung in Festkommaarithmetik (*'int_zooming'* = *'true'*) und hochgenauer Berechnung in Fließkommaarithmetik (*'int_zooming'* = *'false'*) gewählt werden. Bei der Festkommaberechnung kann es allerdings zu leichten Grauwertabweichungen und Pixeln mit undefinierten Grauwerten kommen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Achtung

Aus Geschwindigkeitsgründen wird die Region des Eingabebildes nicht beachtet. Das Ausgabebild hat immer ein komplettes Rechteck als Region.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild.
- ▷ **PolarTransImage** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Ergebnisbild.
- ▷ **Row** (input_control) ..... number ⇝ *real* / integer
  Zeilenkoordinate des Kreismittelpunktes.
  Defaultwert: 256
  Wertevorschläge: `Row` ∈ {0, 16, 32, 64, 128, 240, 256, 480, 512}
- ▷ **Column** (input_control) ..... number ⇝ *real* / integer
  Spaltenkoordinate des Kreismittelpunktes.
  Defaultwert: 256
  Wertevorschläge: `Column` ∈ {0, 16, 32, 64, 128, 256, 320, 512, 640}
- ▷ **AngleStart** (input_control) ..... angle.rad ⇝ *real*
  Winkel, der die erste Spalte des Zielbildes definiert.
  Defaultwert: 0.0
  Wertevorschläge: `AngleStart` ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853, 12.566370616}
- ▷ **AngleEnd** (input_control) ..... angle.rad ⇝ *real*
  Winkel, der die letzte Spalte des Zielbildes definiert.
  Defaultwert: 6.2831853
  Wertevorschläge: `AngleEnd` ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853, 12.566370616}
- ▷ **RadiusStart** (input_control) ..... number ⇝ *real* / integer
  Kreisradius, der die erste Zeile des Zielbildes definiert.
  Defaultwert: 0
  Wertevorschläge: `RadiusStart` ∈ {0, 16, 32, 64, 100, 128, 256, 512}
  Typischer Wertebereich: 0 ≤ RadiusStart
- ▷ **RadiusEnd** (input_control) ..... number ⇝ *real* / integer
  Kreisradius, der die letzte Zeile des Zielbildes definiert.
  Defaultwert: 100
  Wertevorschläge: `RadiusEnd` ∈ {0, 16, 32, 64, 100, 128, 256, 512}
  Typischer Wertebereich: 0 ≤ RadiusEnd
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Zielbildes.
  Defaultwert: 512
  Wertevorschläge: `Width` ∈ {256, 320, 512, 640, 800, 1024}
  Typischer Wertebereich: 0 ≤ `Width` ≤ 32767
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Zielbildes.
  Defaultwert: 512
  Wertevorschläge: `Height` ∈ {240, 256, 480, 512, 600, 1024}
  Typischer Wertebereich: 0 ≤ `Height` ≤ 32767
- ▷ **Interpolation** (input_control) ..... string ⇝ *string*
  Interpolationsverfahren für die Transformation.
  Defaultwert: 'nearest_neighbor'
  Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear'}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

- Automatisch parallelisiert auf interner Datenebene.

Siehe auch

polar_trans_image_inv, polar_trans_region, polar_trans_region_inv, polar_trans_contour_xld, polar_trans_contour_xld_inv

Modul

Foundation

polar_trans_image_inv (PolarImage : XYTransImage : Row, Column, AngleStart, AngleEnd, RadiusStart, RadiusEnd, Width, Height, Interpolation :)

Rücktransformieren eines Polarkoordinatenbildes

polar_trans_image_inv transformiert die Polarkoordinatendarstellung eines Bildes, gespeichert in PolarImage, zurück auf einen Kreisring in kartesischen Koordinaten, der durch die Radien RadiusStart und RadiusEnd sowie durch die Winkel AngleStart und AngleEnd beschrieben ist und dessen Mittelpunkt in (Row, Column) liegt. Alle diese Werte können frei aus den reellen Zahlen gewählt werden. Die Gesamtgröße des Ausgabebildes beträgt Width × Height Pixel.

Über den Parameter Interpolation kann dabei zwischen den Interpolationsverfahren *'bilinear'* und *'nearest_neighbor'* gewählt werden. Bei *'nearest_neighbor'* wird der Grauwert des Pixels im Ausgabebild aus dem Grauwert des nächstgelegenen Pixels im Eingabebild bestimmt. Bei *'bilinear'* wird der Grauwert des Pixels im Ausgabebild durch bilineare Interpolation aus den Grauwerten der vier nächstgelegenen Pixel im Eingabebild bestimmt. Der Modus *'bilinear'* liefert qualitativ bessere Ergebnisse, ist allerdings auch langsamer als *'nearest_neighbor'*.

Die Winkel und Radien sind inklusive, die erste Zeile des Eingabebildes wird also auf einen Kreis im Abstand RadiusStart vom angegebenen Mittelpunkt abgebildet, die letzte Zeile auf einen Kreis im Abstand RadiusEnd.

polar_trans_image_inv ist die exakte Umkehrfunktion zu polar_trans_image_ext.

Die Aufrufsequenz:

```
polar_trans_image_ext(Image, PolarImage, Row, Column, rad(360), 0, 0,
Radius, Width, Height, Interpolation)
polar_trans_image_inv(PolarImage, XYTransImage, Row, Column, rad(360), 0,
0, Radius, Width, Height, Interpolation)
```

liefert im Ausgabebild XYTransImage das Bild Image eingeschränkt auf den Kreis um (Row, Column) mit Radius Radius zurück.

polar_trans_image_inv kann auf OpenCL-Geräten ausgeführt werden. Dabei kann es zu geringfügigen Abweichungen des Ergebnisses gegenüber der Ausführung auf der CPU kommen.

Für Bilder vom Typ *'byte'*, *'int2'* oder *'uint2'* kann mittels des Systemparameters *'int_zooming'* (siehe set_system) zwischen schneller Berechnung in Festkommaarithmetik (*'int_zooming'* = *'true'*) und hochgenauer Berechnung in Fließkommaarithmetik (*'int_zooming'* = *'false'*) gewählt werden. Bei der Festkommaberechnung kann es aber zu leichten Grauwertabweichungen kommen. Auch die Domäne der Ergebnisbilder kann sich in diesem Fall leicht unterscheiden.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

▷ **PolarImage** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Eingabebild.

▷ **XYTransImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Ergebnisbild.

▷ **Row** (input_control) . . . . . number ⇝ *real* / integer
Zeilenkoordinate des Kreismittelpunktes.
Defaultwert: 256
Wertevorschläge: Row ∈ {0, 16, 32, 64, 128, 240, 256, 480, 512}
Typischer Wertebereich: 0 ≤ Row ≤ 32767

▷ **Column** (input_control) . . . . . number ⇝ *real* / integer
Spaltenkoordinate des Kreismittelpunktes.
Defaultwert: 256
Wertevorschläge: Column ∈ {0, 16, 32, 64, 128, 256, 320, 512, 640}
Typischer Wertebereich: 0 ≤ Column ≤ 32767

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Winkel, der die erste Spalte des Eingabebildes definiert.
Defaultwert: 0.0
Wertevorschläge: AngleStart ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853}

▷ **AngleEnd** (input_control) . . . . . angle.rad ⇝ *real*
Winkel, der die letzte Spalte des Eingabebildes definiert.
Defaultwert: 6.2831853
Wertevorschläge: AngleEnd ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853}

▷ **RadiusStart** (input_control) . . . . . number ⇝ *real* / integer
Kreisradius, der die erste Zeile des Eingabebildes definiert.
Defaultwert: 0
Wertevorschläge: RadiusStart ∈ {0, 16, 32, 64, 100, 128, 256, 512}
Typischer Wertebereich: 0 ≤ RadiusStart

▷ **RadiusEnd** (input_control) . . . . . number ⇝ *real* / integer
Kreisradius, der die letzte Zeile des Eingabebildes definiert.
Defaultwert: 100
Wertevorschläge: RadiusEnd ∈ {0, 16, 32, 64, 100, 128, 256, 512}
Typischer Wertebereich: 0 ≤ RadiusEnd

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des Zielbildes.
Defaultwert: 512
Wertevorschläge: Width ∈ {256, 320, 512, 640, 800, 1024}
Typischer Wertebereich: 0 ≤ Width ≤ 32767

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des Zielbildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {240, 256, 480, 512, 600, 1024}
Typischer Wertebereich: 0 ≤ Height ≤ 32767

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Interpolationsverfahren für die Transformation.
Defaultwert: 'nearest_neighbor'
Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear'}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Siehe auch

polar_trans_image_ext, polar_trans_region, polar_trans_region_inv, polar_trans_contour_xld, polar_trans_contour_xld_inv

Modul

Foundation

projective_trans_image (Image : TransImage : HomMat2D, Interpolation, AdaptImageSize, TransformDomain :)

Berechnen der projektiven Transformation eines Bildes.

projective_trans_image wendet die durch die homogene Transformationsmatrix HomMat2D beschriebene projektive Transformation (Homographie) auf das Eingabebild Image an und speichert das Ergebnis in das Ausgabebild TransImage.

Ist der Parameter AdaptImageSize auf *'false'* gesetzt, so erhält TransImage dieselbe Größe wie Image; ist AdaptImageSize *'true'*, so wird die Zielgröße falls möglich automatisch so gewählt, dass alle nicht-negativen Punkte des transformierten Bildes sichtbar sind.

Der Parameter Interpolation bestimmt, welche Interpolationsmethode zur Bestimmung der Grauwerte verwendet wird. Für Interpolation = *'nearest_neighbor'* wird der Grauwert aus dem nächstgelegenen Punkt im Eingabebild bestimmt. Dieser Modus ist sehr schnell, führt aber zu der typischen „Klötzchenbildung“ bei starker Vergrößerung. Für Interpolation = *'bilinear'* werden die Grauwerte bilinear interpoliert, was länger dauert, aber auch zu wesentlich besseren Ergebnissen führt.

Mit TransformDomain kann festgelegt werden, ob der Definitionsbereich von Image transformiert wird. Da die Transformation des Definitionsbereichs Zeit kostet, kann dieser Parameter dazu verwendet werden, festzulegen, ob dies erwünscht ist. Wenn TransformDomain auf *'false'* gesetzt wird, wird der Definitionsbereich des Eingabebildes ignoriert. Somit wird das komplette Bild transformiert.

Die projektive Transformationsmatrix HomMat2D kann z.B. mit vector_to_proj_hom_mat2d erzeugt werden.

Bei einer Homographie werden die zu projizierenden Punkte durch homogene Vektoren der Form (x, y, w) dargestellt. Ein euklidischer Punkt ergibt sich durch (x',y') = $(\frac{x}{w}, \frac{y}{w})$. Wie bei affine_trans_image entspricht bei projective_trans_image x der Zeilenkoordinate und y der Spaltenkoordinate. Mit dieser Konvention sind affine Transformationen ein Spezialfall der projektiven Transformationen, bei denen die letzte Zeile von HomMat2D die Form $(0, 0, c)$ besitzt.

Für Bilder vom Typ *'byte'* oder *'uint2'* kann mittels des Systemparameters *'int_zooming'* (siehe set_system) zwischen schneller Berechnung in Fließkommaarithmetik (*'int_zooming'* = *'true'*) und hochgenauer Berechnung in Fließkommaarithmetik (*'int_zooming'* = *'false'*) gewählt werden. Insbesondere für Interpolation = *'bilinear'* kann es bei der schnellen Fließkommaarithmetik zu leichten Grauwertabweichungen kommen, da der schnellere Algorithmus nur eine geringere Genauigkeit von etwa 10^{-7} mal der Größe des Bildes erreicht. Bei starken Skalierungen ist daher *'int_zooming'* = *'false'* zu empfehlen.

Achtung

Der Transformation liegt das gleiche Koordinatensystem zugrunde, das auch bei dem Operator affine_trans_pixel verwendet wird. Das bedeutet, dass die Transformation nicht mit der Transformationsmatrix HomMat2D, sondern mit einer modifizierte Version davon durchgeführt wird. Eine Transformation mit projective_trans_image entspricht für jeden Punkt (Row_i, Col_i) des Bildes der folgenden Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt):

$$\begin{pmatrix} RowTrans_i \\ ColTrans_i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & -0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \text{HomMat2D} \cdot \begin{bmatrix} 1 & 0 & +0.5 \\ 0 & 1 & +0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} Row_i \\ Col_i \\ 1 \end{pmatrix}$$

Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von dem Bild mit Operatoren wie z.B. area_center_gray abgeleitet wurden. Wenn z.B. ein rotationssymmetrisches Bild unter Verwendung von hom_mat2d_rotate um seinen mit area_center_gray bestimmten Schwerpunkt rotiert wird, so liegt das transformierte Bild nicht auf dem Originalbild. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von projective_trans_image die Transformationsmatrix HomMat2D durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
projective_trans_image(Image, TransImage, HomMat2DAdapted, 'bilinear',
'false', 'false')
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

projective_trans_image kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet und der Parameter TransformDomain auf *'false'* gesetzt wird. Das Ergebnis kann dabei geringfügig von dem auf der CPU berechneten abweichen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **TransImage** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ergebnisbild.
- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **Interpolation** (input_control) ..... string ⇝ *string*
  Interpolationsmethode für die Transformation.
  Defaultwert: 'bilinear'
  Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear'}
- ▷ **AdaptImageSize** (input_control) ..... string ⇝ *string*
  Automatische Größenanpassung für Zielbild.
  Defaultwert: 'false'
  Werteliste: AdaptImageSize ∈ {'true', 'false'}
- ▷ **TransformDomain** (input_control) ..... string ⇝ *string*
  Soll die Region des Eingabebildes auch transformiert werden?
  Defaultwert: 'false'
  Werteliste: TransformDomain ∈ {'true', 'false'}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

vector_to_proj_hom_mat2d, hom_vector_to_proj_hom_mat2d, proj_match_points_ransac, proj_match_points_ransac_guided, hom_mat3d_project

Siehe auch

projective_trans_image_size, projective_trans_contour_xld, projective_trans_region, projective_trans_point_2d, projective_trans_pixel

Modul

Foundation

projective_trans_image_size (Image : TransImage : HomMat2D, Interpolation, Width, Height, TransformDomain :)

Berechnen der projektive Transformation eines Bildes bei vorgegebener Zielgröße.

projective_trans_image_size wendet die durch die homogene Transformationsmatrix HomMat2D beschriebene projektive Transformation (Homographie) auf das Eingabebild Image an und speichert das Ergebnis in das Ausgabebild TransImage.

TransImage wird dabei auf die Zielgröße Height×Width beschnitten. Ansonsten entspricht der Operator der Alternative projective_trans_image.

Achtung

Der Transformation liegt das gleiche Koordinatensystem zugrunde, das auch bei dem Operator `affine_trans_pixel` verwendet wird. Das bedeutet, dass die Transformation nicht mit der Transformationsmatrix `HomMat2D`, sondern mit einer modifizierte Version davon durchgeführt wird. Eine Transformation mit `projective_trans_image_size` entspricht für jeden Punkt (Row_i, Col_i) des Bildes der folgenden Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt):

$$\begin{pmatrix} RowTrans_i \\ ColTrans_i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & -0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D} \cdot \begin{bmatrix} 1 & 0 & +0.5 \\ 0 & 1 & +0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} Row_i \\ Col_i \\ 1 \end{pmatrix}$$

Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von dem Bild mit Operatoren wie z.B. `area_center_gray` abgeleitet wurden. Wenn z.B. ein rotationssymmetrisches Bild unter Verwendung von `hom_mat2d_rotate` um seinen mit `area_center_gray` bestimmten Schwerpunkt rotiert wird, so liegt das transformierte Bild nicht auf dem Originalbild. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von `projective_trans_image_size` die Transformationsmatrix `HomMat2D` durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
projective_trans_image_size(Image, TransImage, HomMat2DAdapted, 'bilinear',
Width, Height, 'false')
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

`projective_trans_image_size` kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet und der Parameter `TransformDomain` auf *'false'* gesetzt wird. Das Ergebnis kann dabei geringfügig von dem auf der CPU berechneten abweichen.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **`TransImage`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ergebnisbild.
- ▷ **`HomMat2D`** (input_control) ..... hom_mat2d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **`Interpolation`** (input_control) ..... string ⇝ *string*
  Interpolationsmethode für die Transformation.
  Defaultwert: 'bilinear'
  Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear'}
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Breite des Ausgabebildes.
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Ausgabebildes.
- ▷ **`TransformDomain`** (input_control) ..... string ⇝ *string*
  Soll die Region des Eingabebildes auch transformiert werden?
  Defaultwert: 'false'
  Werteliste: `TransformDomain` ∈ {'true', 'false'}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

vector_to_proj_hom_mat2d, hom_vector_to_proj_hom_mat2d, proj_match_points_ransac, proj_match_points_ransac_guided, hom_mat3d_project

Siehe auch

projective_trans_image, projective_trans_contour_xld, projective_trans_region, projective_trans_point_2d, projective_trans_pixel

Modul

Foundation

rotate_image (Image : ImageRotate : Phi, Interpolation :)

Rotieren von Bildern um das Zentrum.

rotate_image dreht das Bild Image gegen den Uhrzeigersinn um Phi Grad um den Bildmittelpunkt und gibt das Ergebnis im Ausgabebild ImageRotate zurück. Das Ausgabebild hat die gleiche Größe wie das Eingabebild. Die einzigen Ausnahmen sind Drehungen um 90 und 270 Grad, wo Breite und Höhe vertauscht werden. Der Definitionsbereich des Eingabebildes wird ignoriert, d.h., als das volle Rechteck des Bildes angenommen. Der Definitionsbereich des Ausgabebildes entspricht der Überschneidung des transformierten Rechtecks und des Rechtecks des Ausgabebildes.

Falls Phi ein Vielfaches von 90 Grad ist, ist dieser Operator viel schneller als der allgemeine Operator affine_trans_image. Der Parameter Interpolation entspricht der Wirkung bei affine_trans_image. Bei Rotationen um 90, 180 und 270 Grad wird der Parameter ignoriert. Wenn es erforderlich ist, den Definitionsbereich ebenfalls mit zu rotieren, muss der Operator projective_trans_image genutzt werden.

Für Bilder vom Typ *'byte'*, *'int2'* oder *'uint2'* kann mittels des Systemparameters *'int_zooming'* (siehe set_system) zwischen schneller Berechnung in Festkommaarithmetik (*'int_zooming'* = *'true'*) und hochgenauer Berechnung in Fließkommaarithmetik (*'int_zooming'* = *'false'*) gewählt werden. Mit Ausnahme der Rotationen um 90, 180 oder 270 Grad kann es bei der Festkommaberechnung zu leichten Grauwertabweichungen kommen. Auch die Domäne der Ergebnisbilder kann sich in diesem Fall leicht unterscheiden.

Achtung

Der Definitionsbereich des Eingabebildes wird ignoriert, d.h., als das volle Rechteck des Bildes angenommen. Der Definitionsbereich des Ausgabebildes entspricht der Überschneidung des transformierten Rechtecks und des Rechtecks des Ausgabebildes. Der Winkel Phi wird in Grad und nicht im Bogenmaß angegeben. rotate_image wird für Rotationen um 90, 180 und 270 Grad nicht intern parallelisiert!

rotate_image kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet. Aus numerischen Gründen kann es dabei zu geringfügigen Abweichungen des Ergebnisses gegenüber der Ausführung auf der CPU kommen.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Eingabebild.

▷ **ImageRotate** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Rotiertes Ausgabebild.

▷ **Phi** (input_control) ..... angle.deg ⇝ *real* / integer
Rotationswinkel.
Defaultwert: 90
Wertevorschläge: Phi ∈ {90, 180, 270}
Typischer Wertebereich: 0 ≤ Phi ≤ 360
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.2

▷ **Interpolation** (input_control) ..... string ⇝ *string*
Art der Interpolation.
Defaultwert: 'constant'
Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}

Beispiel

```
read_image(Image,'monkey')
dev_display (Image)
rotate_image(Image,RotImage,270,'constant')
dev_display (RotImage)
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

hom_mat2d_rotate, affine_trans_image

Siehe auch

mirror_image

Modul

Foundation

zoom_image_factor (Image : ImageZoomed : ScaleWidth, ScaleHeight, Interpolation :)

Skalieren von Bildern um einen Vergrößerungsfaktor.

zoom_image_factor vergrößert das Bild Image um einen Faktor ScaleWidth in der Breite und einen Faktor ScaleHeight in der Höhe. Der Parameter Interpolation gibt dabei die Art der verwendeten Interpolationsmethode an (siehe affine_trans_image). Die Domäne des Eingabebildes wird ignoriert, d.h. es wird die volle Domäne des Bildrechtecks angenommen.

Achtung

Falls der Systemparameter *'int_zooming'* auf *'true'* gesetzt ist, kann die intern verwendete Integer-Arithmetik in den folgenden beiden Fällen zu Fehlern führen: Zum einen kann das Ergebnisbild leicht fehlerhafte Grauwerte enthalten, wenn zoom_image_factor auf ein uint2 oder int2 Bild mit hohem Dynamikbereich (das heißt, Bilder mit Werten nahe an den Grenzen der entsprechenden Typen) angewendet wird und gleichzeitig Skalierungsfaktoren kleiner als 0.5 verwendet werden. Zum zweiten können undefinierte Grauwerte am unteren und am rechten Bildrand entstehen, wenn für Interpolation ein anderer Wert als *'nearest_neighbor'* gewählt wird, ein großer Skalierungsfaktor angewendet wird und außerdem ein großes Ergebnisbild entsteht. Die maximale Breite B_{max} dieses Randes mit undefinierten Grauwerten kann mit der Formel $B_{max} = 0.5 \cdot S \cdot I/2^{15}$ abgeschätzt werden, wobei S den Skalierungsfaktor in einer Dimension und I die Größe des Ausgabebildes in der entsprechenden Dimension darstellt. In beiden Fällen wird empfohlen, den Systemparameter *'int_zooming'* mittels set_system auf *'false'* zu setzen.

Wird *'nearest_neighbor'* als Interpolationsmethode gesetzt, können die Ergebnisse für verschiedene Bildtypen leicht voneinander abweichen. Grund dafür sind Bildtyp-spezifische Optimierungen des Interpolationsverfahrens.

zoom_image_factor wird für ScaleWidth = 0.5 und ScaleHeight = 0.5 nicht intern parallelisiert. Außerdem erfolgt keine interne Parallelisierung für Interpolation=*'nearest_neighbor'*.

zoom_image_factor kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet. Aus numerischen Gründen kann es dabei zu geringfügigen Abweichungen des Ergebnisses gegenüber der Ausführung auf der CPU kommen.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Eingabebild.

▷ **ImageZoomed** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Skaliertes Ausgabebild.

▷ **ScaleWidth** (input_control) . . . . . extent.x ⇝ *real*
Faktor für Breite des Zielbildes.
Defaultwert: 0.5
Wertevorschläge: `ScaleWidth` ∈ {0.25, 0.5, 1.5, 2.0}
Typischer Wertebereich: 0.001 ≤ `ScaleWidth` ≤ 10.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1

▷ **ScaleHeight** (input_control) . . . . . extent.y ⇝ *real*
Faktor für Höhe des Zielbildes.
Defaultwert: 0.5
Wertevorschläge: `ScaleHeight` ∈ {0.25, 0.5, 1.5, 2.0}
Typischer Wertebereich: 0.001 ≤ `ScaleHeight` ≤ 10.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.1

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Art der Interpolation.
Defaultwert: 'constant'
Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}

Beispiel

```
read_image(Image,'monkey')
dev_display (Image)
zoom_image_factor (Image, ZoomImage, 0.25, 0.25, 'constant')
dev_display (ZoomImage)
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

`zoom_image_size`, `affine_trans_image`, `hom_mat2d_scale`

Siehe auch

`hom_mat2d_scale`, `affine_trans_image`

Modul

Foundation

zoom_image_size (Image : ImageZoom : Width, Height,
Interpolation :)

Skalieren von Bildern auf eine vorgegebene Größe.

`zoom_image_size` vergrößert das Bild `Image` auf die durch `Width` und `Height` vorgegebene Größe. Der Parameter `Interpolation` gibt dabei die Art der verwendeten Interpolationsmethode an (siehe

affine_trans_image). Die Domäne des Eingabebildes wird ignoriert, d.h. es wird die volle Domäne des Bildrechtecks angenommen.

Achtung

Falls der Systemparameter *'int_zooming'* auf *'true'* gesetzt ist, kann die intern verwendete Integer-Arithmetik in den folgenden beiden Fällen zu Fehlern führen: Zum einen kann das Ergebnisbild leicht fehlerhafte Grauwerte enthalten, wenn zoom_image_size auf ein uint2 oder int2 Bild mit hohem Dynamikbereich (das heißt, Bilder mit Werten nahe an den Grenzen der entsprechenden Typen) angewendet wird und gleichzeitig Skalierungsfaktoren (Verhältnis von Ausgabe- zu Eingabebildgröße) kleiner als 0.5 verwendet werden. Zum zweiten können undefinierte Grauwerte am unteren und am rechten Bildrand entstehen, wenn für Interpolation ein anderer Wert als *'nearest_neighbor'* gewählt wird, ein großer Skalierungsfaktor angewendet wird und außerdem ein großes Ergebnisbild entsteht. Die maximale Breite B_{max} dieses Randes mit undefinierten Grauwerten kann mit der Formel $B_{max} = 0.5 \cdot S \cdot I / 2^{15}$ abgeschätzt werden, wobei S den Skalierungsfaktor in einer Dimension und I die Größe des Ausgabebildes in der entsprechenden Dimension darstellt. In beiden Fällen wird empfohlen, den Systemparameter *'int_zooming'* mittels set_system auf *'false'* zu setzen.

Wird *'nearest_neighbor'* als Interpolationsmethode gesetzt, können die Ergebnisse für verschiedene Bildtypen leicht voneinander abweichen. Grund dafür sind Bildtyp-spezifische Optimierungen des Interpolationsverfahrens.

zoom_image_size wird nicht intern parallelisiert, wenn Width und Height den halben Dimensionen von Image entsprechen. Außerdem erfolgt keine interne Parallelisierung für Interpolation=*'nearest_neighbor'*.

zoom_image_size kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet. Aus numerischen Gründen kann es dabei zu geringfügigen Abweichungen des Ergebnisses gegenüber der Ausführung auf der CPU kommen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild
- ▷ **ImageZoom** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Skaliertes Ausgabebild.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Zielbildes.
  Defaultwert: 512
  Wertevorschläge: Width ∈ {128, 256, 512}
  Typischer Wertebereich: $2 \leq$ Width ≤ 512
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Zielbildes.
  Defaultwert: 512
  Wertevorschläge: Height ∈ {128, 256, 512}
  Typischer Wertebereich: $2 \leq$ Height ≤ 512
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
- ▷ **Interpolation** (input_control) ..... string ⇝ *string*
  Art der Interpolation.
  Defaultwert: 'constant'
  Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}

Beispiel

```
read_image(Image,'monkey')
dev_display (Image)
zoom_image_size(Image,ZoomImage,200,200,'constant')
dev_display (ZoomImage)
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

`zoom_image_factor`, `affine_trans_image`, `hom_mat2d_scale`

Siehe auch

`hom_mat2d_scale`, `affine_trans_image`

Modul

Foundation

10.6 Glättung

Dieses Kapitel beschreibt Operatoren für Glättungsfilter. Für weitere Informationen über Filter und deren Anwendung verweisen wir auf die Einleitung zum Kapitel Filter.

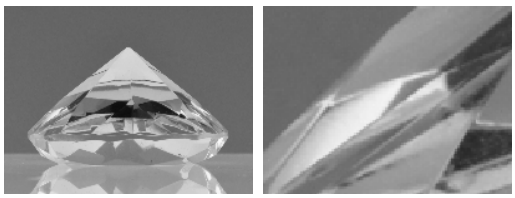
Allgemeine Informationen über Glättungsfilter

Glättungsfilter sind Filter zur Unterdrückung des Rauschens eines Bildes. Zu diesem Zweck wird angenommen, dass im ungestörten oder richtigen Bild der Grauwert eines jeden Punktes sich nicht komplett von seiner Umgebung abhebt, im Idealfall gar nur wenig variiert. Dann kann es hilfreich sein, den gemessenen Wert durch eine Abschätzung anhand benachbarter Datenpunkte zu ersetzen. Eine solche Abschätzung kann auf verschiedene Arten erhalten werden. HALCON stellt daher verschiedene Glättungsfilter zur Verfügung.

Die verschiedenen Operatoren sind für verschiedene Arten von Rauschen geeignet und unterscheiden sich auch bezüglich ihrer Geschwindigkeit. Informationen zur Komplexität (wie die Skalierung) ist, wenn zur Verfügung stehend, in der spezifischen Operator-Referenz zu finden. Während die meisten Operatoren ein einzelnes Bild behandeln, gibt es auch solche, die ein Bild in Abhängigkeit eines anderen Bildes bearbeiten (z.B. Mehrkanalfilter wie `mean_n` und `rank_n` oder kantenerhaltende Filter wie `guided_filter` und `bilateral_filter`, welche ein Führungsbild benutzen können). Es sei erwähnt, dass manche Filter beide Möglichkeiten haben und weitere Information jeweils in der spezifischen Operator-Referenz steht.

Glättungsfilter für ein Einzelbild mit zufälligem Rauschen

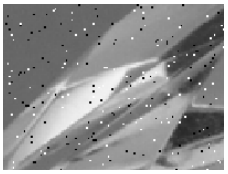
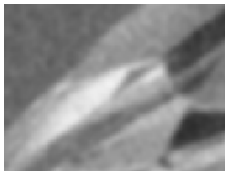
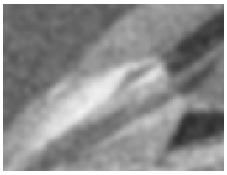
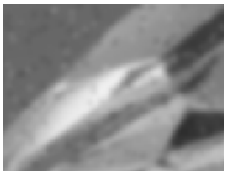
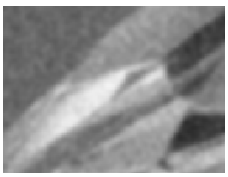
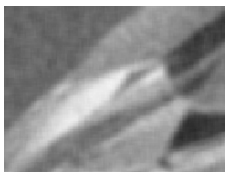
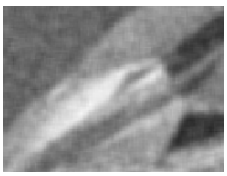
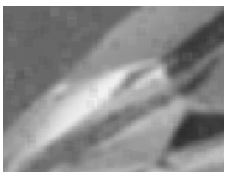
Diese Glättungsfilter wenden ihre Glättungsfunktion auf jedem Kanal des Eingabebildes separat an und geben ein geglättetes Bild mit gleicher Anzahl Kanäle zurück. In der folgenden Tabelle listen wir verschiedene, implementierte Glättungsfilter und wenden diese auf drei verschiedene Arten zufälligen Rauschens an. Die gezeigten Bilder geben bloß einen Hinweis auf die Möglichkeiten des Operators für das gegebene Rauschen. Das geglättete Bild ist stark von den verwendeten Parametern und dem Bild abhängig. Zum Vergleich zeigen wir die verschiedenen verrauschten Bilder ungefiltert in der ersten Zeile der Tabelle. Das rauschfreie Bild ist in der folgenden Abbildung zu sehen ((1) das ganze Bild sowie (2) der Bildausschnitt, der für eine deutlichere Visualisierung möglicher Effekte auf Kanten oder Überbleibsel des Salt & Pepper Rauschens verwendet wird).



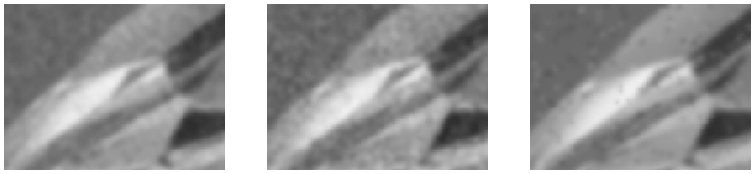
(1) (2)

(1) Rauschfreies Bild, (2) gewählter Bildausschnitt zur besseren Visualisierung der Filtereigenschaften

Zusätzlich markiert sind Filter, die speziell in Bezug auf Geschwindigkeit (G), Erhaltung von Kanten (E) oder einen Kompromiss (K) zwischen den beiden entworfen wurden. Die Zahlen in Klammern verweisen auf weitere Informationen am Ende der Tabelle.

Weißes Rauschen	Gauß'sches Rauschen	Salt & Pepper Rauschen	Zeitdauer[1]	Alternativen
Verrauschtes Bild				
				
binomial_filter(G)				
			1|2	gauss_filter, smooth_image, derivate_gauss, isotropic_diffusion
smooth_image				
			2 | 9	binomial_filter, gauss_filter, mean_image, derivate_gauss, isotropic_diffusion
mean_image(G)				
			1 | 1	binomial_filter, gauss_filter, smooth_image
anisotropic_diffusion(E)[2]				
			805|2568	bilateral_filter, guided_filter

guided_filter(E)[2,3]



13 | 62

bilateral_filter, anisotropic_diffusion, median_image

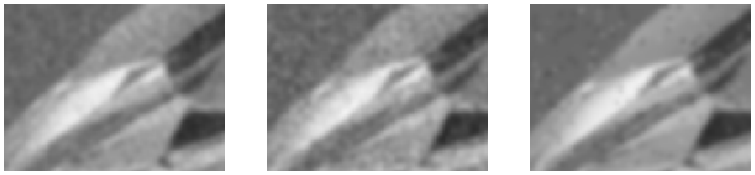
bilateral_filter(E)[3]



16 | 54

guided_filter, anisotropic_diffusion, median_image

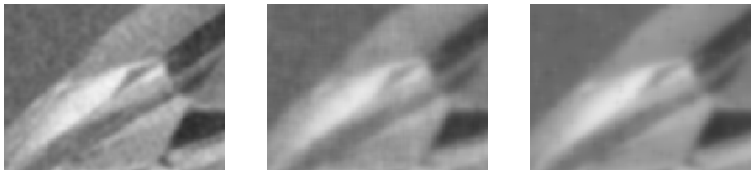
gauss_filter



1 | 4

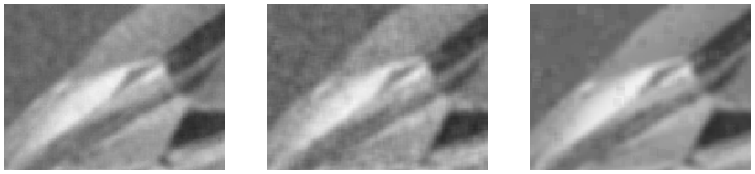
binomial_filter, smooth_image, derivate_gauss, isotropic_diffusion

isotropic_diffusion(E)[2]



11 | 51

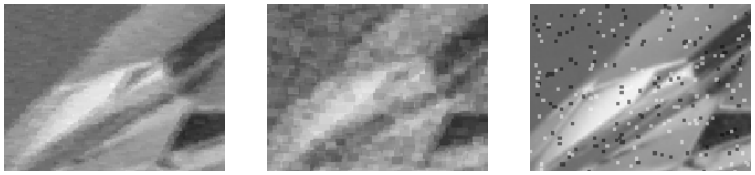
sigma_image



10 | 33

anisotropic_diffusion, rank_image

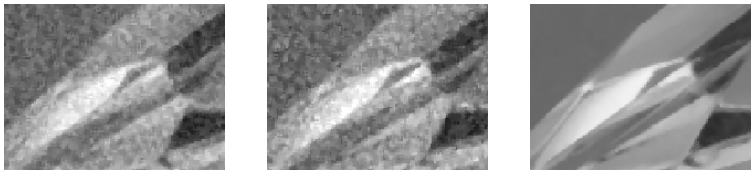
midrange_image



3 | 11

sigma_image

median_image(E)



3 | 4

median_rect, rank_image, rank_rect

median_rect(K)



2 | 3

median_image, rank_rect, rank_image

median_separate(K)



7 | 24

median_image

median_weighted



14 | 47

median_image, trimmed_mean, sigma_image

rank_rect(E)



2 | 8

rank_image, median_rect, median_image

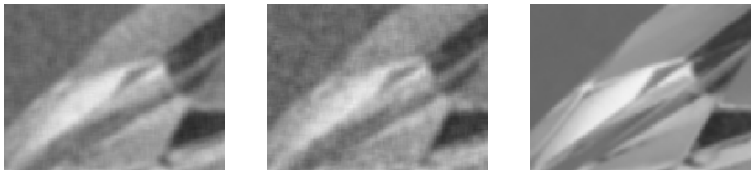
rank_image(E)



3 | 15

rank_rect, median_image, median_rect

mean_sp



4 | 9

mean_image, median_image, median_separate, eliminate_min_max

eliminate_min_max



11 | 42

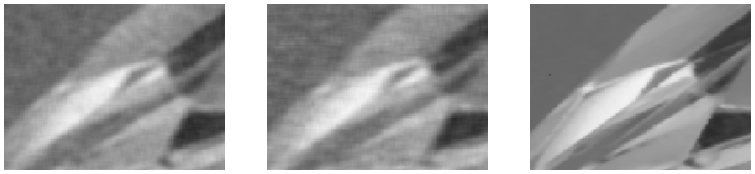
eliminate_sp



1 | 10

mean_sp, mean_image, median_image, eliminate_min_max

trimmed_mean



7 | 29

sigma_image, median_weighted, median_image

Anmerkungen zu den Zahlen in den eckigen Klammern in der obigen Tabelle:

1. Die Zahlen in der Spalte 'Zeitdauer' geben Hinweise zur Laufzeit, welche der Operator für die Bildverarbeitung benötigt. Die Zeitangaben sind in willkürlichen Einheiten. Erhalten wurden sie durch Mittlung über mehrere Glättungen der unterschiedlich verrauschten Bilder. Die Laufzeit des Operators hängt von vielen Faktoren ab, nicht zuletzt von den Parametern oder der Bildgröße. Für jeden Filter steht die erste Zahl für ein Bild der Größe 800x600, die Zweite für ein Bild der Größe 1497x1160. Die gezeigten Bilder sind Ausschnitte des geglätteten kleineren Bildes mit uns vernünftig erscheinenden Parameterwerten (dabei führen die verwendeten Parameter weder notwendigerweise zu den besten Resultaten noch zur schnellsten Laufzeit). Selbst wenn zwei Filter den selben Parameternamen als Eingabegröße verwenden, haben wir nicht notwendigerweise die selben Werte verwendet. Die Zeitangaben dienen also eher zur groben Orientierung.
2. Dieser Operator kann iterativ verwendet werden.
3. Dieser Operator verwendet ein Führungsbild (welches auch das Bild Image selbst sein kann).

Glättungsfilter für ein Einzelbild mit systematischem Rauschen

Videobilder, die aus zwei Halbbildern zusammengesetzt sind, können systematische Fehler enthalten. In solchen Fällen kann der Operator fill_interlace helfen.

Operatoren zur Glättung über mehrere Kanäle

Diese Glättungsfilter nehmen als Eingabe ein Bild mehrerer Kanäle und geben ein Bild mit nur einem Kanal (Graubild) zurück. In HALCON sind folgende Filter dieser Gruppe implementiert:

- mean_n
- rank_n

Weitere Operatoren

Mit den Glättungsfiltern verbunden ist der folgende Operator: info_smooth, welcher Informationen zum Operator smooth_image zurück gibt.

Glossar

Im Folgenden werden die wichtigsten Begriffe, die im Umfeld von Glättungsfiltern verwendet werden, beschrieben:

Glättung Glättung bezeichnet das Anwenden einer Filterfunktion auf ein gegebenes Bild, um die wichtigsten Datenmuster zu erfassen und gleichzeitig das Rauschen zu unterdrücken.

Zufälliges Rauschen Als zufälliges Rauschen verstehen wir stationäre Änderungen der Helligkeits- oder Farbinformation der Pixel um einen zufälligen Wert, wobei ein Mittelwert von 0 über das ganze Bild angenommen wird.

Systematisches Rauschen Systematisches Rauschen ist berechenbares Rauschen, verursacht z.B. durch den spezifischen Aufbau zur Bildgewinnung.

anisotropic_diffusion (Image : ImageAniso : Mode, Contrast, Theta, Iterations :)

Anisotrope Glättung eines Bildes.

Der Operator anisotropic_diffusion führt eine anisotrope Diffusion des Eingabebildes Image nach dem Modell von Perona und Malik durch. Dieser Vorgang wird auch als nichtlinear isotrope Diffusion bezeichnet. Ordnet man dem Bild eine Grauwertfunktion u zu, so ist der Algorithmus eine Diskretisierung der partiellen Differentialgleichung

$$u_t = div(g(|\nabla u|^2, c)\nabla u)$$

mit dem durch Image definierten Anfangswert $u = u_0$ zu einem Zeitpunkt t_0. Die Gleichung wird dabei Iterations mal in Zeitschritten der Länge Theta iteriert, so dass das Ausgabebild ImageAniso die Grauwertfunktion zum Zeitpunkt $t_0 + \texttt{Iterations} \cdot \texttt{Theta}$ zeigt.

Ziel der anisotropen Diffusion ist es, in flächigen Bildbereichen Bildstörungen wie Rauschen zu eliminieren, ohne dabei Bildkanten zu verwischen. Die Unterscheidung zwischen Kanten und Flächen wird durch den Schwellenwert Contrast für die Größe der Grauwertdifferenz zwischen benachbarten Pixeln getroffen. Contrast wird auch als Kontrastparameter bezeichnet und mit c abgekürzt.

Der variable Diffusionskoeffizient g kann dabei verschiedenen monoton fallenden Funktionen mit Werten zwischen *0* und *1* folgen und bestimmt das Antwortverhalten des Diffusionsvorganges auf eine Kante. Über den Parameter Mode stehen die folgenden Funktionen zur Verfügung:

$$g_1(x, c) = \frac{1}{\sqrt{1 + 2\frac{x}{c^2}}}$$

Die Wahl der Funktion g_1, wenn Mode auf *'parabolic'* gesetzt wird, garantiert, dass die zugehörige Differentialgleichung parabolisch ist, so dass eine Wohlgestelltheitstheorie für das Problem existiert und das Verfahren für beliebige Schrittweiten Theta stabil ist. In diesem Falle bleibt es aber bei einer leichten Diffusion auch über Kanten hinweg, deren Stärke c übersteigt.

$$g_2(x, c) = \frac{1}{1 + \frac{x}{c^2}}$$

Die in der Veröffentlichung von Perona und Malik verwendete Funktion bei Wahl von *'perona-malik'* für Mode besitzt nicht die theoretischen Eigenschaften von g_1, hat sich in der Praxis aber dennoch als ausreichend stabil erwiesen. Die theoretische Instabilität zeigt sich in einer leichten Schärfung von starken Kanten.

$$g_3(x,c) = 1 - \exp(-C\frac{c^8}{x^4})$$

Die von Weickert vorgeschlagene Wahl g_3 bei `Mode`=*'weickert'* mit der Konstanten $C = 3.31488$ ist eine Steigerung von g_2 im Bezug auf die Kantenschärfe. Der Übergang zwischen Glättung und Schärfung erfolgt hier sehr abrupt an der Stelle $x = c^2$.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **`ImageAniso`** (output_object) ..... image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Diffusionskoeffizient als Funktion der Kantenstärke.
  Defaultwert: 'weickert'
  Werteliste: `Mode` ∈ {'weickert', 'perona-malik', 'parabolic'}
- ▷ **`Contrast`** (input_control) ..... real ⇝ *real*
  Kontrastparameter.
  Defaultwert: 5.0
  Wertevorschläge: `Contrast` ∈ {2.0, 5.0, 10.0, 20.0, 50.0, 100.0}
  Restriktion: Contrast > 0
- ▷ **`Theta`** (input_control) ..... real ⇝ *real*
  Zeitschritt.
  Defaultwert: 1.0
  Wertevorschläge: `Theta` ∈ {0.5, 1.0, 3.0}
  Restriktion: Theta > 0
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Anzahl Iterationen.
  Defaultwert: 10
  Wertevorschläge: `Iterations` ∈ {1, 3, 10, 100, 500}
  Restriktion: Iterations >= 1

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Alternativen

`bilateral_filter`, `guided_filter`

Literatur

J. Weickert; „'Anisotropic Diffusion in Image Processing'; PhD Thesis; Fachbereich Mathematik, Universität Kaiserslautern; 1996.

P. Perona, J. Malik; „Scale-space and edge detection using anisotropic diffusion"; Transactions on Pattern Analysis and Machine Intelligence 12(7), pp. 629-639; IEEE; 1990.

G. Aubert, P. Kornprobst; „Mathematical Problems in Image Processing"; Applied Mathematical Sciences 147; Springer, New York; 2002.

Modul

Foundation

bilateral_filter (Image, ImageJoint : ImageBilateral : SigmaSpatial, SigmaRange, GenParamName, GenParamValue :)

Bilaterale Filterung eines Bildes.

bilateral_filter führt eine gemeinsame bilaterale Filterung des Eingabebildes Image und des Führungsbildes ImageJoint durch und liefert das Ergebnis in ImageBilateral zurück. Image und ImageJoint müssen dieselbe Größe haben und vom gleichen Typ sein.

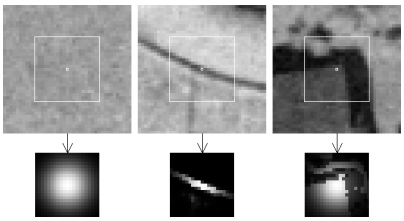
SigmaSpatial definiert die Größe der Filtermaske und korrespondiert mit der Standardabweichung eines gewöhnlichen Gaussfilters. Größere Werte erhöhen den Einflussbereich des Filters und führen dazu, dass weniger Details erhalten bleiben.

SigmaRange wird verwendet, um die Filtermaske abhängig vom Inhalt von ImageJoint um das aktuelle Pixel zu verändern. Nur Pixel in Bereichen mit schwachen Kanten, die einen niedrigeren Kontrast als SigmaRange haben, beeinflussen die Glättung. Es ist zu beachten, dass der Kontrast in uint2- oder real-Bildern deutlich vom Defaultwert von SigmaRange abweichen kann und der Parameter daher gesondert eingestellt werden muss.

GenParamName und GenParamValue können zurzeit verwendet werden, um den Trade-off zwischen Geschwindigkeit und Genauigkeit zu steuern (siehe unten).

Einfluss des Führungsbildes

Jedes Pixel von Image wird mit einer Filtermaske, die von ImageJoint abhängt, gefiltert. Die Filtermaske kombiniert eine Gauss'sche Nähefunktion, die von SigmaSpatial abhängt, und eine Gauss'sche Ähnlichkeitsfunktion, die Grauwertunterschiede abhängig von SigmaRange gewichtet.



Beispiele von lokalen Filtermasken abhängig von ImageJoint: Links: In homogenen Bereichen entspricht die Filtermaske einer Gaussglocke. Mitte: Die Filtermaske passt sich der Linie an. Das heißt, nur dunkle Pixel werden geglättet, die Kanten bleiben erhalten. Rechts: Die Filtermaske ähnelt der Ecke. Man beachte, dass sie über den Schatten in Bereiche mit ähnlichen Grauwerten hineinragt.

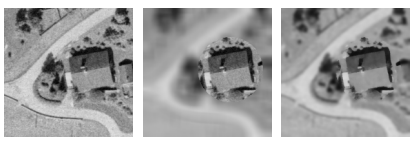
Sind Image und ImageJoint identisch, verhält sich bilateral_filter wie ein kantenerhaltender Glättungsfilter, dessen Maskengröße von SigmaSpatial bestimmt wird. Pixel an Kanten mit einem Kontrast, der deutlich über SigmaRange liegt, werden erhalten, während Pixel in homogenen Bereichen geglättet werden.



(1) (2) (3)

(1) Image und (2) ImageJoint sind identisch. Das führt zu kantenerhaltender Glättung in (3) ImageBilateral.

Sind Image und ImageJoint verschieden, wird jeder Pixel von Image mit einer Filtermaske geglättet, die von ImageJoint beeinflusst wird. Pixel and Positionen, wo ImageJoint starke Kanten aufweist, deren Kontrast deutlich über SigmaRange liegt, werden weniger geglättet als Pixel in homogenen Bereichen.



(1) (2) (3)

(1) Image und (2) ImageJoint sind verschieden. Nur Kanten in Bereichen, wo ImageJoint Kanten hat, bleiben in (3) ImageBilateral erhalten.

Ist ImageJoint konstant, verhält sich bilateral_filter äquivalent zu einer Gauss-Glättung mit SigmaSpatial (siehe gauss_filter oder smooth_image).

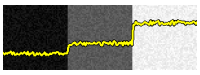


(1) (2) (3)

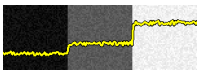
(2) ImageJoint ist konstant. Das entspricht einer Gauss-Glättung von (1) Image. (3) ImageBilateral.

Einfluss der Glättungsparameter

Die folgenden Beispiele zeigen den Einfluss von SigmaRange auf einem künstlichen Bild. In diesem Bild liegt das Rauschlevel bei 10 Grauwerten, die linke Kante hat einen Kontrast von 50 Grauwerten, die rechte Kante hat einen Kontrast von 100 Grauwerten. Die gelbe Linie zeigt das Grauwertprofil eines horizontalen Querschnitts.



Originalbild mit überlagertem Grauwertprofil. Das Bild wird sowohl für Image als auch für ImageJoint verwendet.



Filterergebnis mit SigmaRange = 1: Kein Effekt, da SigmaRange unter dem Rauschlevel liegt. Daher wird das Rauschen als Kante behandelt und erhalten.



Filterergebnis mit SigmaRange = 25: Das Rauschen wird geglättet, die Kanten bleiben erhalten.



Filterergebnis mit SigmaRange = 50: Die schwächere Kante wird geglättet, die stärkere Kante bleibt erhalten.



Filterergebnis mit SigmaRange = 100: Beide Kanten werden geglättet.

Generische Parameter

Die folgenden Werte für GenParamName werden unterstützt:

'sampling_method' Per Default verwendet bilateral_filter einen Näherungsalgorithmus, der nur eine Teilmenge von abgetasteten Punkten für die Berechnung der lokalen Filtermasken verwendet.

Mit *'sampling_method'* kann die verwendete Abtastmethode ausgewählt werden. Mögliche Werte sind:

'grid' **(Default)** Verwendet ein reguläres Gitter zur Abtastung der Filtermasken.

'poisson_disk' Verwendet eine Poisson-Disk-Abtastung. Diese Methode ist langsamer als *'grid'*, möglicherweise liefert sie aber weniger Artefakte.

'exact' Verwendet alle verfügbaren Punkte. Diese Methode ist die genaueste, aber auch die langsamste. Falls *'exact'* verwendet wird, wird *'sampling_ratio'* ignoriert.

'sampling_ratio' Steuert, wie viele Punkte für die Abtastung verwendet werden.

Durch Setzen von *'sampling_ratio'* auf *1.0* wird die exakte Methode verwendet. Eine niedrigere Abtastrate führt zu einer schnelleren Filterung, aber auch zu etwas ungenaueren Ergebnissen.

Empfohlene Werte: 0.25, 0.5, 0.75, 1.0

Default: 0.50

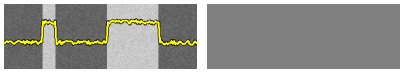
Rolling-Bilateral-Filter

`bilateral_filter` kann iterativ aufgerufen werden. In dem Fall wird das Ergebnis einer Iteration als Führungsbild der nächsten Iteration verwendet. Das kann nützlich sein, z.B. um kleine Strukturen aus dem Originalbild zu entfernen, auch wenn sie einen hohen Kontrast haben.

Das folgende Beispiel zeigt den Effekt des Rolling-Filters anhand eines künstlichen Beispielbildes. In diesem Bild liegt das Rauschlevel bei 10 Grauwerten, der Kontrast zwischen dunklen und hellen Bereichen beträgt 100 Grauwerte. Der linke helle Streifen ist 10 Pixel breit, der rechte Streifen 40 Pixel. Die gelbe Linie zeigt das Grauwertprofil eines horizontalen Querschnitts.

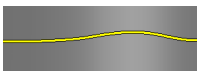
Verwendete Parameter: `ImageJoint` konstant, `SigmaSpatial` = 25, `SigmaRange` = 15.

```
* Verwenden des Rolling-Bilateral-Filter
* (verwende ein konstantes Führungsbild für die erste Iteration).
gen_image_proto (Image, ImageJoint, 128)
for I := 1 to 6 by 1
  bilateral_filter (Image, ImageJoint, ImageJoint, 25, 15, [], [])
endfor
```



(1) (2)

(1) Eingabebild `Image` und (2) `ImageJoint` für die erste Iteration des Rolling-Filters.



Ergebnis nach der ersten Iteration: Der schmalere Streifen ist verschwunden.



Ergebnis nach der dritten Iteration: Die Kanten des rechten Streifens sind teilweise wiederhergestellt.



Ergebnis nach der sechsten Iteration: Die Kanten des rechten Streifens sind komplett wiederhergestellt.

Mathematischer Hintergrund

Die Berechnung der gefilterten Grauwerte I'_i beruht auf folgender Formel:

$$I'_i = \frac{1}{K_i} \sum_{k \in \omega_i} c_{ik} \cdot s_{ik} \cdot I_k$$

Dabei ist c_{ik} die Nähefunktion (closeness function)

$$c_{ik} = exp\left(\frac{-\|i-k\|^2}{2 \cdot SigmaSpatial^2}\right)$$

und s_{ik} die Ähnlichkeitsfunktion (similarity function)

$$s_{ik} = exp\left(\frac{-(J_i - J_k)^2}{2 \cdot SigmaRange^2}\right).$$

K_i ist ein Normalisierungsfaktor

$$K_i = \sum_{k \in \omega_i} c_{ik} \cdot s_{ik}.$$

I'_i, I_i und J_i sind die Grauwerte von `ImageBilateral`, `Image` und `ImageJoint` an der Pixelposition i. ω_i ist die Umgebung um den Pixel i.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Zu filterndes Bild.
- ▷ **`ImageJoint`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Führungsbild.
- ▷ **`ImageBilateral`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Gefiltertes Ausgabebild.
- ▷ **`SigmaSpatial`** (input_control) ..... real ⇝ *real*
  Größe der Gaussglocke der Nähefunktion.
  Defaultwert: 3.0
  Wertevorschläge: `SigmaSpatial` ∈ {1.0, 2.0, 3.0, 10.0}
  Restriktion: SigmaSpatial > 0.6
- ▷ **`SigmaRange`** (input_control) ..... real ⇝ *real*
  Größe der Gaussglocke der Ähnlichkeitsfunktion.
  Defaultwert: 20.0
  Wertevorschläge: `SigmaRange` ∈ {3.0, 10.0, 20.0, 50.0, 100.0}
  Restriktion: SigmaRange > 0.0001
- ▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'sampling_ratio', 'sampling_method'}
- ▷ **`GenParamValue`** (input_control) ..... attribute.value(-array) ⇝ *real* / integer / string
  Werte der generischen Parameter.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {'grid', 'poisson_disk', 'exact', 0.5, 0.25, 0.75, 1.0}

Beispiel

```
read_image (Image, 'mreut')
* Edge-preserving smoothing
bilateral_filter (Image, Image, ImageBilateral, 5, 20, [], [])
* Rolling filter (5 iterations)
gen_image_proto (Image, ImageJoint, 0)
for I := 1 to 5 by 1
  bilateral_filter (Image, ImageJoint, ImageJoint, 5, 20, [], [])
endfor
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`read_image`

Nachfolger

`threshold`, `dyn_threshold`, `var_threshold`, `regiongrowing`

Alternativen

`guided_filter`, `anisotropic_diffusion`, `median_image`

Literatur

C. Tomasi, R. Manduchi: “Bilateral filtering for gray and color images“; Sixth International Conference in Computer Vision; S. 839-846; January 1998.

F. Banterle, M. Corsini, P. Cignoni, R. Scopigno: “A Low-Memory, Straightforward and Fast Bilateral Filter Through Subsampling in Spatial Domain“; Computer Graphics Forum, no. 1, vol 31; S. 19-23; February 2012.

G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, K. Toyama: “Digital Photography with Flash and No-flash Image Pairs“; ACM Trans., no. 3, vol. 23; S. 9; August 2004.

R. Bridson: “Fast Poisson Disk Sampling in Arbitrary Dimensions“; ACM SIGGRAPH 2007 Sketches, no. 22; 2007.

Modul

Foundation

binomial_filter (Image : ImageBinomial : MaskWidth, MaskHeight :)

Glättung eines Bildes mit dem Binomialfilter.

binomial_filter glättet das Bild Image mit einem Binomialfilter mit einer Maskengröße von MaskWidth × MaskHeight Pixeln und liefert das geglättete Bild in ImageBinomial zurück. Der Binomialfilter ist eine sehr gute Annäherung eines Gaußfilters, die extrem effizient mit ganzzahligen Operationen berechnet werden kann. Daher ist binomial_filter sehr schnell. Sei $m =$ MaskHeight und $n =$ MaskWidth. Dann sind die Filterkoeffizienten b_{ij} durch Binomialkoeffizienten

$$\binom{l}{k} = \frac{l!}{k!(l-k)!}$$

wie folgt gegeben:

$$b_{ij} = \frac{1}{2^{n+m-2}} \binom{m-1}{i} \binom{n-1}{j}$$

Hierbei ist $i = 0, \ldots, m-1$ und $j = 0, \ldots, n-1$. Der Binomialfilter führt näherungsweise dieselbe Glättung durch wie ein Gaußfilter mit $\sigma = \sqrt{n-1}/2$, wobei der Einfachheit halber $m = n$ angenommen wird. Im Detail ist der Zusammenhang zwischen n und σ wie folgt:

n	σ
3	0.7523
5	1.0317
7	1.2505
9	1.4365
11	1.6010
13	1.7502
15	1.8876
17	2.0157
19	2.1361
21	2.2501
23	2.3586
25	2.4623
27	2.5618
29	2.6576
31	2.7500
33	2.8395
35	2.9262
37	3.0104

Falls unterschiedliche Werte für MaskHeight und MaskWidth gewählt werden, stimmt der obige Zusammenhang zwischen n und σ immer noch und bezieht sich auf die Stärke der Glättung in Zeilen- und Spaltenrichtung.

binomial_filter kann auf OpenCL-Geräten ausgeführt werden.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **ImageBinomial** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Geglättetes Bild.
- ▷ **MaskWidth** (input_control) ..... integer ⇝ *integer*
  Filterbreite.
  Defaultwert: 5
  Werteliste: MaskWidth ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37}
- ▷ **MaskHeight** (input_control) ..... integer ⇝ *integer*
  Filterhöhe.
  Defaultwert: 5
  Werteliste: MaskHeight ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert binomial_filter den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

read_image, grab_image

Nachfolger

regiongrowing, threshold, sub_image, dyn_threshold, auto_threshold

Alternativen

gauss_filter, smooth_image, derivate_gauss, isotropic_diffusion

Siehe auch

mean_image, anisotropic_diffusion, sigma_image, gen_lowpass

Modul

Foundation

eliminate_min_max (Image : FilteredImage : MaskWidth, MaskHeight, Gap, Mode :)

Ortsraumglättung zur Entfernung von Rauschspitzen.

eliminate_min_max filtert Rauschspitzen aus einem verrauschten Bild. Hierfür wird das Bild punktweise im Ortsraum verarbeitet. Um Kanten und Linien nicht allzu sehr zu glätten, werden nur diejenigen Bildpunkte ersetzt, die ein lokales Minimum bzw. Maximum darstellen. Der Gedanke hierbei ist, dass bei Kanten oder Linien noch

mindestens ein weiterer Punkt innerhalb der lokalen Umgebung existiert, der ungefähr denselben Grauwert besitzt wie der betrachtete. Dadurch bleibt die Bedingung eines lokalen Minimums/Maximums unerfüllt, der Punkt wird nicht ersetzt und die Kante/Linie nicht geglättet. Um die Strenge der Ersetzungsbedingung beeinflussen zu können, kann mit `Gap` angegeben werden, um wie viel kleiner bzw. größer das lokale Minimum/Maximum gegenüber den anderen Grauwerten mindestens sein muss. `eliminate_min_max` arbeitet also nach folgendem Schema: Enthalte $U(x,y)$ alle Grauwerte der $N \times M$ großen, rechteckigen lokalen Umgebung des Punktes (x,y), außer dem Grauwert des Punktes (x,y) selbst;

- Falls $Grauwert(x,y) \geq$ `Gap` $+ maximum(U(x,y)) \rightarrow Ersetzung$;
- Falls $Grauwert(x,y) +$ `Gap` $\leq minimum(U(x,y)) \rightarrow Ersetzung$;
- sonst $Grauwert(x,y)$ gleichbelassen;

Mit welchem neuen Wert ein zu ersetzender Punkt belegt wird, ist durch `Mode` zu steuern:

`Mode` $= 1 \rightarrow$ Ersetzung eines lokalen Maximums durch das nächstkleinere lokale Maximum und Ersetzung eines lokalen Minimums durch das nächstgrößere lokale Minimum
`Mode` $= 2 \rightarrow$ Ersetzung durch das (ungewichtete) arithmetische Mittel aller Punkte innerhalb der lokalen Umgebung (inkl. des betrachteten Punktes)
`Mode` $= 3 \rightarrow$ Ersetzung durch den Median-Wert aller Punkte der lokalen Umgebung (inkl. des betrachteten Punktes) (auch bei allen anderen Werten von `Mode` außer 1,2)

Die Breite und Höhe der (rechteckigen) lokalen Punktumgebung wird durch `MaskWidth` und `MaskHeight` festgelegt. Bemerkung zur Randbehandlung: Punkte jenseits des Bildrands werden nicht in die Betrachtung einbezogen. Sei z.B. eine 3×3-Maske gegeben, so reduziert sich die lokale Umgebung des Punktes $(0,0)$ auf die Werte von $(1,0)$, $(0,1)$ und $(1,1)$.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Der Filter ist nur für Bilder vom Typ 'byte' implementiert. Werden für `MaskWidth` und `MaskHeight` gerade statt ungerader Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt). Die Filtermaskenbreite bzw. -höhe darf die Bildbreite bzw. -höhe nicht überschreiten.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Bild, das gefiltert werden soll.

▷ **`FilteredImage`** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Geglättetes Bild.

▷ **`MaskWidth`** (input_control) . . . . . extent.x ⇝ *integer*
Breite der rechteckigen Filtermaske.
Defaultwert: 3
Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9}
Typischer Wertebereich: 3 ≤ `MaskWidth` ≤ width(Image)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth)

▷ **`MaskHeight`** (input_control) . . . . . extent.y ⇝ *integer*
Höhe der rechteckigen Filtermaske.
Defaultwert: 3
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9}
Typischer Wertebereich: 3 ≤ `MaskHeight` ≤ width(Image)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth)

▷ **`Gap`** (input_control) . . . . . number ⇝ *real*
Mindestabstand vom lokalen Minimum/Maximum zu allen anderen Grauwerten der lokalen Punktumgebung.
Defaultwert: 1.0
Wertevorschläge: `Gap` ∈ {1.0, 2.0, 5.0, 10.0}

▷ **Mode** (input_control) . . . . . integer ⇝ *integer*
Ersetzungsvorschrift für zu ersetzende Grauwerte (1=nächstes Minimum/Maximum, 2=Average, sonst Median).
Defaultwert: 3
Werteliste: `Mode` ∈ {1, 2, 3}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `eliminate_min_max` den Wert TRUE. Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`wiener_filter`, `wiener_filter_ni`

Siehe auch

`mean_sp`, `mean_image`, `median_image`, `median_weighted`, `binomial_filter`, `gauss_filter`, `smooth_image`

Literatur

M. Imme:„A Noise Peak Elimination Filter“; S. 204-211 in CVGIP Graphical Models and Image Processing, Vol. 53, No. 2, March 1991

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Modul

Foundation

eliminate_sp

eliminate_sp (Image : ImageFillSP : MaskWidth, MaskHeight, MinThresh, MaxThresh :)

Ersetzen der Werte außerhalb der Schwellen durch den Mittelwert.

`eliminate_sp` ersetzt alle Grauwerte außerhalb des angegebenen Grauwertintervalls (`MinThresh` bis `MaxThresh`) durch den Mittelwert der Nachbarn. Es werden für die Mittlung aber nur die Nachbarpunkte herangezogen, die ebenfalls innerhalb des Grauwertintervalls liegen. Ist in der Umgebung kein solcher Punkt enthalten, wird der Originalgrauwert verwendet. Die Grauwerte im Eingabebild, die innerhalb des Grauwertintervalls liegen, werden ebenfalls unverändert übernommen.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Werden für `MaskHeight` oder `MaskWidth` gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Eingabebild.

▷ **ImageFillSP** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Geglättetes Bild.

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
Breite der Filtermaske.
Defaultwert: 3
Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9, 11}
Typischer Wertebereich: 3 ≤ `MaskWidth` ≤ 512 (lin)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth) && MaskWidth < width(Image) * 2

▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
Höhe der Filtermaske.
Defaultwert: 3
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11}
Typischer Wertebereich: 3 ≤ `MaskHeight` ≤ 512 (lin)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskHeight) && MaskHeight < height(Image) * 2

▷ **MinThresh** (input_control) . . . . . integer ⇝ *integer*
Mindestgrauwert.
Defaultwert: 1
Wertevorschläge: `MinThresh` ∈ {1, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}

▷ **MaxThresh** (input_control) . . . . . integer ⇝ *integer*
Maximalgrauwert.
Defaultwert: 254
Wertevorschläge: `MaxThresh` ∈ {5, 7, 9, 11, 15, 23, 31, 43, 61, 101, 200, 230, 250, 254}
Restriktion: MinThresh <= MaxThresh

Beispiel

```
read_image(Image,'mreut')
dev_display(Image)
eliminate_sp(Image,ImageFillSP,3,3,101,201)
dev_display(ImageFillSP)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`disp_image`

Alternativen

`mean_sp`, `mean_image`, `median_image`, `eliminate_min_max`

Siehe auch

`binomial_filter`, `gauss_filter`, `smooth_image`, `anisotropic_diffusion`, `sigma_image`, `eliminate_min_max`

Modul

Foundation

`fill_interlace` (ImageCamera : ImageFilled : Mode :)

Interpolation von 2 Video-Halbbildern.

fill_interlace berechnet aus einem Videobild, das aus zwei Halbbildern zusammengesetzt ist, ein interpoliertes Vollbild bzw. entfernt gerade oder ungerade Bildzeilen. Wird ein Bild mit einer Videokamera aufgenommen, so besteht dies aus zwei Halbbildern, die mit einem zeitlichen Abstand aufgenommen wurden, aber in der digitalen Form in einem gemeinsamen Bild abgelegt sind. Hierdurch können starke Fehler bei der weiteren Verarbeitung entstehen. Um diese Fehler etwas zu mildern, wird das Videobild modifiziert. Dabei wird entweder jede zweite Zeile neu berechnet oder entfernt. Der Parameter Mode gibt an, ob dies für gerade ('even' bzw. 'rmeven') oder ungerade ('odd' bzw. 'rmodd') Zeilennummern gemacht werden soll. In den Modi 'even' bzw. 'odd' werden die Grauwerte in den erzeugten Zeilen als Mittelwert aus dem direkten Nachbarn oberhalb bzw. unterhalb des aktuellen Punktes berechnet. In den Modi 'rmeven' bzw. 'rmodd' werden gerade bzw. ungerade Zeilen entfernt (dementsprechend hat dann das Ausgabebild nur noch die halbe Höhe des Eingabebildes). Hat Mode den Wert 'switch', dann werden gerade und ungerade Zeilen vertauscht.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **ImageCamera** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Graubild, bestehend aus zwei Halbbildern.
- ▷ **ImageFilled** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Vollbild mit interpolierten bzw. entfernten Zeilen.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Angabe, ob die geraden oder ungeraden Zeilen ersetzt bzw. entfernt werden sollen.
  Defaultwert: 'odd'
  Werteliste: Mode ∈ {'odd', 'even', 'rmodd', 'rmeven', 'switch'}

Beispiel

```
read_image(Image,'video_image')
fill_interlace(Image,New,'odd')
sobel_amp(New,Sobel,'sum_abs',3)
```

Komplexität

Pro Bildpunkt: $O(2)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert fill_interlace den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

read_image, grab_image

Nachfolger

sobel_amp, edges_image, regiongrowing, diff_of_gauss, threshold, dyn_threshold, auto_threshold, mean_image, binomial_filter, gauss_filter, anisotropic_diffusion, sigma_image, median_image

Siehe auch

median_image, binomial_filter, gauss_filter, crop_part

Modul

Foundation

gauss_filter (Image : ImageGauss : Size :)

Glättung mit diskreten Gaußfunktionen.

gauss_filter glättet Bilder mittels der diskreten Gaußfunktion, welche eine Annäherung der kontinuierlichen Gauß-Funktion ist, die durch die folgende Formel definiert wird,

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-x^2}{2\sigma^2}\right)$$

Die glättende Wirkung erhöht sich dabei mit zunehmender Filtergröße. Es werden folgende Filtergrößen (Size) unterstützt (in Klammer steht der sigma-Wert der Gaußfunktion):

3	(0.600)
5	(1.075)
7	(1.550)
9	(2.025)
11	(2.550)

Zur Randbehandlung werden die Grauwerte der Bilder an den Bildrändern gespiegelt. Im Gegenteil zu dem Operator gauss_image, wächst den Wert des Sigma Parameters gradlinig mit der Filtergröße.

Als Alternative zu gauss_filter bietet sich binomial_filter an. binomial_filter ist deutlich effizienter als gauss_filter. Es ist zu beachten, dass die Maskengröße in binomial_filter nicht dieselbe Glättung wie bei gauss_filter bewirkt. Korrespondierende Maskengrößen lassen sich über die jeweiligen sigma-Werte der Gaußfunktion bestimmen.

gauss_filter kann auf OpenCL-Geräten ausgeführt werden. Allerdings kann das Ergebnis von dem der skalaren Implementierung geringfügig abweichen.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Damit gauss_filter auf einem OpenCL-Gerät ausgeführt werden kann, muss Image eine Breite und Höhe von mindestens 64 Pixeln aufweisen.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Zu glättendes Bild.
- ▷ **ImageGauss** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Gefiltertes Bild.
- ▷ **Size** (input_control) . . . . . integer ⇝ *integer*
  Gewünschte Filtergröße.
  Defaultwert: 5
  Werteliste: Size ∈ {3, 5, 7, 9, 11}

Beispiel

```
gauss_filter(Input,Gauss,7)
regiongrowing(Gauss,Segments,7,7,5,100)
```

Komplexität

Pro Bildpunkt: $O(Size * 2)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gauss_filter den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`read_image`, `grab_image`

Nachfolger

`regiongrowing`, `threshold`, `sub_image`, `dyn_threshold`, `auto_threshold`

Alternativen

`binomial_filter`, `smooth_image`, `derivate_gauss`, `isotropic_diffusion`

Siehe auch

`mean_image`, `anisotropic_diffusion`, `sigma_image`, `gen_lowpass`

Modul

Foundation

guided_filter (Image, ImageGuide : ImageGuided : Radius, Amplitude :)

Geführte Filterung eines Bildes.

`guided_filter` filtert das Eingabebild `Image` mit Hilfe des Führungsbildes `ImageGuide` und liefert das Ergebnis in `ImageGuided` zurück. `Image` und `ImageGuide` müssen die selbe Größe haben und vom selben Typ sein.

Der `Radius` bestimmt die Größe der Filtermaske. Größere Werte erhöhen den Einflussbereich des Filters und führen dazu, dass weniger Details erhalten bleiben. Der Wert von `Radius` hat keinen Einfluss auf die Laufzeit des Operators.

`Amplitude` wird verwendet, um zu entscheiden, was eine Kante und was eine homogene Fläche ist. Größere Werte von `Amplitude` führen dazu, dass stärkere Kanten geglättet werden. Als Daumenregel sollte `Amplitude` kleiner sein als der Kontrast der Kanten, die erhalten werden sollen. Es ist zu beachten, dass der Kontrast in uint2- oder real-Bildern deutlich vom Defaultwert von `Amplitude` abweichen kann und der Parameter daher gesondert eingestellt werden muss.

Einfluss des Führungsbildes

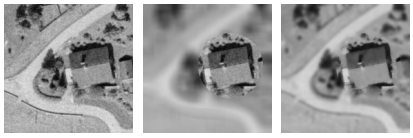
Sind `Image` und `ImageGuide` identisch, verhält sich `guided_filter` wie ein kantenerhaltender Glättungsfilter mit einer Filtermaske mit dem Radius `Radius`. Pixel an Kanten mit einem Kontrast, der deutlich über `Amplitude` liegt, werden erhalten, während Pixel in homogenen Bereichen geglättet werden. Daher ist `guided_filter` eine schnelle Alternative zu `anisotropic_diffusion` oder `bilateral_filter`.



(1) (2) (3)

(1) `Image` und (2) `ImageGuide` sind identisch. Das führt zu kantenerhaltender Glättung in (3) `ImageGuided`.

Sind `Image` und `ImageGuide` verschieden, wird `Image` mit einer Filtermaske mit Radius `Radius` geglättet, außer in Bereichen, wo `ImageGuide` Kanten mit deutlich höherem Kontrast als `Amplitude` aufweist.



(1) (2) (3)

(1) Image und (2) ImageGuide sind verschieden. Nur Kanten in Bereichen, wo ImageGuide Kanten hat, bleiben in (3) ImageGuided erhalten.

Ist ImageGuide konstant, verhält sich guided_filter äquivalent zu 2 aufeinanderfolgenden Aufrufen von mean_image mit Maskengröße 2*Radius+1.

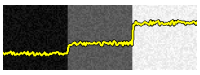


(1) (2) (3)

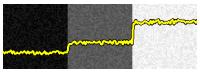
(2) ImageGuide ist konstant. Das entspricht einer doppelten Glättung von (1) Image mit mean_image. (3) ImageGuided.

Einfluss der Glättungsparameter

Die folgenden Beispiele zeigen den Einfluss von Amplitude auf einem künstlichen Bild. In diesem Bild liegt das Rauschlevel bei 10 Grauwerten, die linke Kante hat einen Kontrast von 50 Grauwerten, die rechte Kante hat einen Kontrast von 100 Grauwerten. Die gelbe Linie zeigt das Grauwertprofile eines horizontalen Querschnitts.



Originalbild mit überlagertem Grauwertprofil. Das Bild wird sowohl für Image als auch für ImageGuide verwendet.



Filterergebnis mit Amplitude = 1: Kein Effekt, da Amplitude unter dem Rauschlevel liegt. Daher wird das Rauschen als Kante behandelt und erhalten.



Filterergebnis mit Amplitude = 25: Das Rauschen wird geglättet, die Kanten werden erhalten.



Filterergebnis mit Amplitude = 50: Die schwächere Kante wird geglättet, die stärkere Kante wird erhalten.

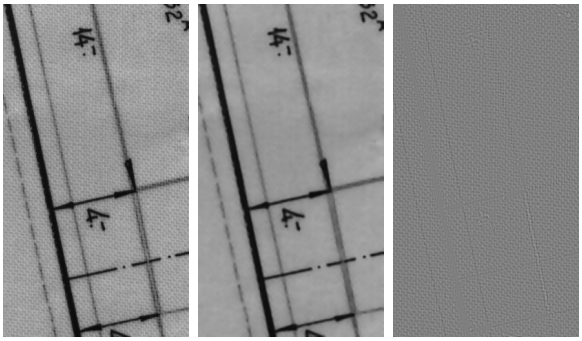


Filterergebnis mit Amplitude = 100: Beide Kanten werden geglättet.

Rolling-Guided-Filter

guided_filter kann iterativ aufgerufen werden. In dem Fall wird das Ergebnis einer Iteration als Führungsbild der nächsten Iteration verwendet. Das kann nützlich sein, z.B. um kleine Strukturen aus dem Originalbild zu entfernen, auch wenn sie einen hohen Kontrast haben.

Im folgenden Beispiel wird der Rolling-Guided-Filter verwendet, um die Textur vom Originalbild zu trennen.



(1) (2) (3)

Texturentfernung mit dem Rolling-Guided-Filter: (1) Original, (2) extrahierte Struktur, (3) extrahierte Textur.

```
* Verwenden des Rolling-Guided-Filter
* (verwende ein konstantes Führungsbild für die erste Iteration).
gen_image_proto (Image, ImageStructure, 0)
for I := 1 to 4 by 1
  guided_filter (Image, ImageStructure, ImageStructure, 1.5, 60)
endfor
* Extrahiere die Textur durch Abziehen der großen Strukturen vom Original.
sub_image (Image, ImageStructure, ImageTexture, 1, 128)
```

Da sich `guided_filter` mit konstantem `ImageGuide` ähnlich zu `mean_image` verhält, kann die erste Iteration durch einen Aufruf von `mean_image` (oder einem ähnlichen Glättungsfilter) ersetzt werden, was schneller ist.

Mathematischer Hintergrund

Die Berechnung der gefilterten Grauwerte I'_i an der Position i beruht auf folgender Formel:

$$I'_i = \overline{a_i} G_i + \overline{b_i}$$

wobei

$$a_i = \frac{\frac{1}{|\omega_i|} \sum_{k \in \omega_i} G_k I_k - \overline{G_i} \overline{I_i}}{\sigma^2_{G_i} + Amplitude^2}$$

und

$$b_i = \overline{I_i} - a_i \overline{G_i}.$$

Dabei sind I_i und G_i die Grauwerte von `Image` und `ImageGuide` an der Pixelposition i. ω_i ist die Umgebung um i mit dem Radius `Radius`. $\overline{a_i}$, $\overline{b_i}$, $\overline{I_i}$ und $\overline{G_i}$ sind die Mittelwerte aller a, b, I oder G in ω_i. σ_{G_i} ist die Standardabweichung aller Grauwerte von G in ω_i. Und $|\omega_i|$ ist die Anzahl der Pixel in ω_i.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **`ImageGuide`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Führungsbild.
- ▷ **`ImageGuided`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **`Radius`** (input_control) ..... integer ⇝ *integer*
  Radius der Filterung.
  Defaultwert: 3
  Wertevorschläge: `Radius` ∈ {1, 2, 3, 5, 10}
  Restriktion: Radius > 0
- ▷ **`Amplitude`** (input_control) ..... real ⇝ *real*
  Steuert den Einfluss der Kanten bei der Filterung.
  Defaultwert: 20.0
  Wertevorschläge: `Amplitude` ∈ {3.0, 10.0, 20.0, 50.0, 100.0}
  Restriktion: Amplitude > 0

Beispiel

```
read_image (Image, 'mreut')
* Edge-preserving smoothing
guided_filter (Image, Image, ImageGuided, 5, 20)
* Rolling filter (5 iterations)
```

```
gen_image_proto (Image, ImageGuide, 0)
for I := 1 to 5 by 1
  guided_filter (Image, ImageGuide, ImageGuide, 5, 20)
endfor
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`read_image`

Nachfolger

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`bilateral_filter`, `anisotropic_diffusion`, `median_image`

Literatur

Kaiming He, Jian Sun, Xiaoou Tang: „Guided Image Filtering“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-35, no. 6; S. 1397-1409; 2013.

Modul

Foundation

info_smooth (: : Filter, Alpha : Size, Coeffs)

Informationen zum Glättungsfilter `smooth_image`.

`info_smooth` liefert eine Abschätzung des Einzugsgebiets der in der Routine `smooth_image` verwendeten Glättungsfilter. Dazu werden die zugrundeliegenden kontinuierlichen Impulsantworten der `Filter` abgetastet, bis ein Filterkoeffizient kleiner als fünf Prozent des Maximalkoeffizienten (im Ursprung) ist. `Alpha` ist der Filterparameter (siehe `smooth_image`). Derzeit werden vier Filter unterstützt (Parameter `Filter`):

’deriche1’, ’deriche2’, ’shen’ und ’gauss’.

Der Gaußfilter wurde dabei konventionell mittels Filtermasken implementiert (bei den drei anderen handelt es sich um rekursive Filter). Im Falle des Gaußfilters werden daher zusätzlich zur Filtergröße auch die Filterkoeffizienten (der eindimensionalen Impulsantwort $f(n)$ mit $n \geq 0$) in `Coeffs` zurückgeliefert.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **`Filter`** (input_control) . . . . . string ⇝ *string*
Name des gewünschten Filters.
Defaultwert: ’deriche2’
Werteliste: `Filter` ∈ {’deriche1’, ’deriche2’, ’shen’, ’gauss’}

▷ **`Alpha`** (input_control) . . . . . real ⇝ *real*
Filterparameter: kleine Werte bewirken starke Glättung (bei ’gauss’ umgekehrt).
Defaultwert: 0.5
Wertevorschläge: `Alpha` ∈ {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0, 10.0}
Typischer Wertebereich: 0.01 ≤ `Alpha` ≤ 50.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Alpha > 0.0

▷ **Size** (output_control) ..... integer ⇝ *integer*
Einzugsgebiet des Filters ist etwa Size x Size Pixel groß.

▷ **Coeffs** (output_control) ..... integer-array ⇝ *integer*
Falls Gaußfilter: Koeffizienten der „positiven“ Hälfte der 1D-Impulsantwort.

Beispiel

```
info_smooth('deriche2',0.5,Size,Coeffs)
smooth_image(Input,Smooth,'deriche2',7)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `info_smooth` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`

Nachfolger

`smooth_image`

Siehe auch

`smooth_image`

Modul

Foundation

isotropic_diffusion (Image : SmoothedImage : Sigma, Iterations :)

Isotrope Glättung eines Bildes.

Der Operator `isotropic_diffusion` führt eine isotrope Diffusion des Eingabebildes `Image` durch. Dies entspricht einer Faltung der Bildmatrix mit einer Gaussmaske mit Standardabweichung `Sigma`. Wird der Parameter `Iterations` auf *0* gesetzt, so wird eine solche Faltung explizit durchgeführt. Für Eingabebilder mit voller ROI entspricht `isotropic_diffusion` dann genau dem Operator `derivate_gauss` bei Wahl von *'none'* für den Parameter `Component`. Ist die Grauwertmatrix größer als die ROI von `Image`, so unterscheiden sich die Operatoren dadurch, dass `derivate_gauss` die Grauwerte außerhalb der ROI miteinbezieht, während `isotropic_diffusion` in jedem Fall die Grauwerte am Rand der ROI spiegelt. Der Rechenaufwand steigt linear mit dem Wert von `Sigma` an.

Besitzt `Iterations` einen positiven Wert, so wird der Glättungsvorgang als Anwendung der Wärmeleitungsgleichung

$$u_t = \Delta u$$

auf die Grauwertfunktion u mit dem durch die Grauwerte von `Image` definierten Anfangswert $u = u_0$ zu einem Zeitpunkt t_0 interpretiert. Diese Gleichung wird dann in `Iterations` Schritten über ein iteratives Verfahren für parabolische partielle Differentialgleichungen bis zum Zeitpunkt $t_0 + \frac{1}{2}\texttt{Sigma}^2$ gelöst, was äquivalent zur obigen Faltung ist. Der Rechenaufwand ist hier proportional zum Wert von `Iterations` und unabhängig von `Sigma`. Für kleine Werte von `Iterations` ist die Rechengenauigkeit aber sehr gering. Die Wahl von $\texttt{Iterations} < 3$ ist daher nicht empfehlenswert.

Für kleinere Werte von `Sigma` ist die Faltungsimplementierung im Normalfall die schnellere Variante. Da die Laufzeit des Differentialgleichungslösers nur von der Anzahl der Iterationen und nicht von `Sigma` abhängt, ist für große Werte von `Sigma` und der Wahl von wenigen Iterationen (z.B. $\texttt{Iterations} = 3$) typischerweise diese Implementierung schneller.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **SmoothedImage** (output_object) . . . . . image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **Sigma** (input_control) . . . . . real ⇝ *real*
  Standardabweichung der Gaussverteilung.
  Defaultwert: 1.0
  Wertevorschläge: `Sigma` ∈ {0.1, 0.5, 1.0, 3.0, 10.0, 20.0, 50.0}
  Restriktion: Sigma > 0
- ▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
  Anzahl Iterationen.
  Defaultwert: 10
  Wertevorschläge: `Iterations` ∈ {0, 3, 10, 100, 500}
  Restriktion: Iterations >= 0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Modul

Foundation

mean_image (Image : ImageMean : MaskWidth, MaskHeight :)

Glättung durch Mittelwertbildung.

`mean_image` führt eine lineare Glättung mit den Grauwerten aller Eingabebilder (`Image`) durch. Die Filtermatrix besteht aus Einsen (gleich gewichtet) und hat die Größe `MaskHeight` × `MaskWidth`. Das Ergebnis der Faltung wird durch `MaskHeight` × `MaskWidth` dividiert. Als Randbehandlung werden die Grauwerte an den Bildrändern gespiegelt.

Für `mean_image` sind spezielle Optimierungen unter Verwendung von SIMD Technologie implementiert. Ob diese tatsächlich zur Anwendung kommen, kann über den Systemparameter *'mmx_enable'* (siehe `set_system`) gesteuert werden. Falls *'mmx_enable'* auf *'true'* gesetzt wird (und die verwendete Hardware SIMD Technologie unterstützt), wird die Berechnung intern unter Verwendung von SIMD Technologie durchgeführt. Es ist zu beachten, dass der Beschleunigungseffekt von SIMD Technologie bei großen, kompakten Eingaberegionen am größten ist. Im Einzelfall kann aber die Ausführung von `mean_image`, abhängig von der Eingaberegion und den Hardwareeigenschaften, mit SIMD Technologie auch länger dauern als ohne.

In jedem Falle wirkt es sich günstig auf das Laufzeitverhalten von `mean_image` aus, wenn man die Eingaberegion von `Image` so wählt, dass es zu keiner Randbehandlung kommt.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Werden für `MaskHeight` oder `MaskWidth` gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

`mean_image` kann für byte-, int2-, uint2-, int4- und real-Bilder auf OpenCL-Geräten ausgeführt werden, wenn `MaskHeight` kleiner als zweimal die Höhe von `Image` ist. Der OpenCL Code berechnet die Filterantwort intern

mit 32 bit Integerarithmetik (für real-Bilder wird einfach genaue Fließkommaarithmetik verwendet). Dies kann zu Überläufen (und damit zu fehlerhaften Ergebnissen) führen, wenn der gesamte Wertebereich von int4- oder real-Bildern verwendet wird. Da zur Verbesserung der Laufzeit für eine `MaskWidth` größer 9 ein vollständiger Scan jeder Zeile von `Image` berechnet wird, kann es in diesem Fall auch bei byte-, int2- oder uint2-Bildern zu Überläufen kommen, wenn `Image` sehr breit ist. In solchen Fällen muss die CPU-Version von `mean_image` verwendet werden.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / int8 / real / vector_field

Zu glättendes Bild.

▷ **`ImageMean`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / int8 / real / vector_field

Geglättetes Bild.

▷ **`MaskWidth`** (input_control) ..... extent.x ⇝ *integer*

Breite der Filtermaske.
Defaultwert: 9
Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
Typischer Wertebereich: 1 ≤ `MaskWidth` ≤ 501
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth) && MaskWidth < width(Image) * 2

▷ **`MaskHeight`** (input_control) ..... extent.y ⇝ *integer*

Höhe der Filtermaske.
Defaultwert: 9
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
Typischer Wertebereich: 1 ≤ `MaskHeight` ≤ 501
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskHeight) && MaskHeight < height(Image) * 2

Beispiel

```
read_image(Image,'fabrik')
mean_image(Image,Mean,3,3)
dev_display(Mean)
```

Komplexität

Pro Bildpunkt: O(15).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `mean_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`reduce_domain`, `rectangle1_domain`

Nachfolger

dyn_threshold, regiongrowing

Alternativen

binomial_filter, gauss_filter, smooth_image

Siehe auch

anisotropic_diffusion, sigma_image, convol_image, gen_lowpass

Modul

Foundation

mean_n (Image : ImageMean : :)

Mittlung der Grauwerte zwischen mehreren Kanälen.

mean_n bildet den pixelweisen Mittelwert aller Kanäle. Es wird für jeden Koordinatenpunkt die Summe alle Grauwerte an diese Koordinate berechnet. Das Ergebnisbild wird mit dem Mittelwert der Grauwerte (Summe geteilt durch Anzahl der Kanäle) belegt. Das Ausgabebild hat einen Kanal.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Mehrkanaliges Graubild.
- ▷ **ImageMean** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Ergebnis der Mittlung.

Beispiel

```
compose3(Channel1,Channel2,Channel3,&MultiChannel);
mean_n(MultiChannel,&Mean);
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

compose2, compose3, compose4, compose5, add_channels

Alternativen

rank_n

Siehe auch

count_channels, mean_image

Modul

Foundation

mean_sp (Image : ImageSPMean : MaskWidth, MaskHeight, MinThresh, MaxThresh :)

Unterdrückung von Salz- und Pfeffer-Rauschen.

mean_sp führt eine Glättung durch Mittelwertbildung durch. Dabei gehen nur die Grauwerte in die Mittlung mit ein, die in dem Intervall von MinThresh bis MaxThresh liegen. Zu helle oder zu dunkle Grauwerte werden bei der Summierung ignoriert. Liegt kein Grauwert bei der Summierung innerhalb des vorgegebenen Intervalls, so wird der Originalgrauwert übernommen. Setzt man die Schwellen auf *0* bzw. *255* so verhält sich mean_sp, abgesehen von der Laufzeit, wie mean_image.

mean_sp wird verwendet um extreme Grauwerte (Salz- und Pfeffer-Rauschen = weiße und schwarze Punkte) zu unterdrücken.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Werden für MaskHeight oder MaskWidth gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **ImageSPMean** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Geglättetes Bild.
- ▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert: 3
  Wertevorschläge: MaskWidth ∈ {3, 5, 7, 9, 11}
  Typischer Wertebereich: 3 ≤ MaskWidth ≤ 512 (lin)
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
  Restriktion: odd(MaskWidth) && MaskWidth < width(Image) * 2
- ▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert: 3
  Wertevorschläge: MaskHeight ∈ {3, 5, 7, 9, 11}
  Typischer Wertebereich: 3 ≤ MaskHeight ≤ 512 (lin)
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
  Restriktion: odd(MaskHeight) && MaskHeight < height(Image) * 2
- ▷ **MinThresh** (input_control) ..... integer ⇝ *integer*
  Mindestgrauwert.
  Defaultwert: 1
  Wertevorschläge: MinThresh ∈ {1, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
- ▷ **MaxThresh** (input_control) ..... integer ⇝ *integer*
  Maximalgrauwert.
  Defaultwert: 254
  Wertevorschläge: MaxThresh ∈ {5, 7, 9, 11, 15, 23, 31, 43, 61, 101, 200, 230, 250, 254}
  Restriktion: MinThresh <= MaxThresh

Beispiel

```
read_image(Image,'mreut')
dev_display(Image)
mean_sp(Image,ImageMeansp,3,3,101,201)
dev_display(ImageMeansp)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`disp_image`

Alternativen

`mean_image`, `median_image`, `median_separate`, `eliminate_min_max`

Siehe auch

`anisotropic_diffusion`, `sigma_image`, `binomial_filter`, `gauss_filter`, `smooth_image`, `eliminate_min_max`

Modul

Foundation

median_image (Image : ImageMedian : MaskType, Radius, Margin :)

Medianfilterung mit verschiedenen Masken.

`median_image` führt eine Medianfilterung des Eingabebildes `Image` mit einer quadratischen oder kreisförmigen Maske durch und liefert das gefilterte Bild in `ImageMedian` zurück. Die Form der Maske kann mit `MaskType` gewählt werden. Der Radius der Maske wird mit `Radius` bestimmt.

Konzeptuell sortiert der Medianfilter alle Grauwerte innerhalb der Maske aufsteigend und wählt dann den Median der Grauwerte aus. Der Median ist der „mittlere“ der sortierten Grauwerte, d.h. der Grauwert mit dem Rang (Position) $(N-1)/2+1$ der sortierten Grauwerte, wobei N die Anzahl der Pixel der Filtermaske ist. Hierbei entspricht der Rang 1 dem kleinsten Grauwert und der Rang N dem größten Grauwert innerhalb der Maske (siehe auch `rank_image`).

Die Filtermaske wird über die Parameter `Radius` und `MaskType` festgelegt. Dabei bestimmt `Radius` die Größe der Maske und `MaskType` die Form mit folgenden Optionen:

- *'circle'*: Die Maske besteht aus den Pixeln innerhalb eines Kreises mit `Radius` um das Pixel des Maskenzentrums.
- *'square'*: Die Maske besteht aus den Pixeln innerhalb eines Quadrats mit einer Kantenlänge von 2·`Radius` +1 Pixel.

`median_image` kann unter anderem zur Glättung von Bildern, zum Unterdrücken unerwünschter Objekte (z.B. Punkte und Linienstrukturen), die kleiner als die Maske sind, und somit zur Schätzung der Hintergrundbeleuchtung für eine Shading-Korrektur oder als Vorverarbeitung für die dynamische Schwellenwertoperation (siehe `dyn_threshold`) verwendet werden.

Bei der Filterung kann über `Margin` zwischen verschiedenen Randbehandlungen gewählt werden:

- Grauwert: Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
- *'continued'*: Fortsetzung der Randpunkte.
- *'cyclic'*: zyklische Fortsetzung der Bildränder.
- *'mirrored'*: Spiegelung der Bildpunkte an den Bildrändern.

Wird eine quadratische Maske mit `Radius` 1 oder 2 (und somit eine 3x3 oder 5x5 Pixel Filtermaske) mit der Randbehandlung *'mirrored'* verwendet, kann `median_image` auf OpenCL-Geräten ausgeführt werden.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

`median_image` verwendet einen Algorithmus mit einer Laufzeit pro Pixel, die von der Maskenhöhe $2 \cdot$ `Radius` $+1$ abhängt. Daher ist `median_image` für quadratische Masken mit einer großen Maskenhöhe langsamer als `median_rect`. Die genaue Maskenhöhe, ab der `median_rect` schneller ist, ist abhängig von der

Rechnerarchitektur (Prozessortyp, Verfügbarkeit von SIMD-Befehlen wie SSE2 oder MMX, Cachegröße und -durchsatz, Speicherdurchsatz). Typischerweise ist das für eine Maskenhöhe > 15 der Fall, kann aber z.B. bei nicht vorhandenen SIMD-Befehlen und langsamem Speicher auch erst für größere Maskenhöhen zutreffen.

Weiterhin ist zu beachten, dass `median_rect` aufgrund der rekursiven Implementierung die Filterantwort intern auf dem kleinsten umschließenden Rechteck der Region des Bildes berechnet. Falls die Region nur einen sehr kleinen Teil des umschließenden Rechtecks abdeckt, kann es daher vorkommen, dass `median_image` auch für größere Maskenhöhen schneller als `median_rect` ist.

Aus Laufzeitgründen wird das Eingabebild `Image` nicht auf NaNs überprüft. Die Verwendung eines Eingabebildes mit NaNs führt zum Absturz von HALCON.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Zu filterndes Bild.
- ▷ **`ImageMedian`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Gefiltertes Bild.
- ▷ **`MaskType`** (input_control) ..... string ⇝ *string*
  Art der Filtermaske.
  Defaultwert: 'circle'
  Werteliste: `MaskType` ∈ {'circle', 'square'}
- ▷ **`Radius`** (input_control) ..... integer ⇝ *integer*
  Radius der Filtermaske.
  Defaultwert: 1
  Wertevorschläge: `Radius` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15, 19, 25, 31, 39, 47, 59}
  Typischer Wertebereich: $1 \leq$ `Radius` ≤ 4095
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 2
- ▷ **`Margin`** (input_control) ..... string ⇝ *string* / integer / real
  Randbehandlung.
  Defaultwert: 'mirrored'
  Wertevorschläge: `Margin` ∈ {'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255}

Beispiel

```
read_image (Image, 'fabrik')
median_image (Image, Median, 'circle', 3, 'continued')
dev_display(Median)
```

Komplexität

Pro Bildpunkt: $O(2 \cdot$ `Radius` $+ 1)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `median_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

read_image

Nachfolger

threshold, dyn_threshold, regiongrowing

Alternativen

median_rect, rank_image, rank_rect

Siehe auch

gray_erosion_rect, gray_dilation_rect, gray_erosion_shape, gray_dilation_shape, gray_erosion, gray_dilation

Literatur

T.S. Huang, G.J. Yang, G.Y. Tang; „A Fast Two-Dimensional Median Filtering Algorithm“; IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 27, no. 1, pp. 13-18, 1979.

R. Haralick, L. Shapiro; „Computer and Robot Vision“; Addison-Wesley, 1992, pp. 318-320.

Modul

Foundation

median_rect (Image : ImageMedian : MaskWidth, MaskHeight :)

Medianfilterung mit rechteckigen Masken.

median_rect führt eine Medianfilterung des Eingabebildes Image mit einer rechteckigen Maske der Größe MaskWidth × MaskHeight durch und liefert das gefilterte Bild in ImageMedian zurück.

Konzeptuell sortiert der Medianfilter alle Grauwerte innerhalb der Maske aufsteigend und wählt dann den Median der Grauwerte aus. Der Median ist der „mittlere“ der sortierten Grauwerte, d.h. der Grauwert mit dem Rang (Position) $(\text{MaskWidth} \cdot \text{MaskHeight} - 1)/2 + 1$ der sortierten Grauwerte, wobei der Rang 1 dem kleinsten Grauwert und der Rang MaskWidth·MaskHeight dem größten Grauwert innerhalb der Maske entspricht (siehe auch rank_rect).

median_rect kann unter anderem zur Glättung von Bildern, zum Unterdrücken unerwünschter Objekte (z.B. Punkte und Linienstrukturen), die kleiner als die Maske sind, und somit zur Schätzung der Hintergrundbeleuchtung für eine Shading-Korrektur oder als Vorverarbeitung für die dynamische Schwellenwertoperation (siehe dyn_threshold) verwendet werden.

median_rect kann bei Verwendung einer quadratischen 3x3 oder 5x5 Filtermaske auf OpenCL-Geräten ausgeführt werden.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Werden für MaskWidth oder MaskHeight gerade statt ungerade Werte übergeben, verwendet median_rect an ihrer Stelle die nächstgrößeren ungeraden Werte.

median_rect verwendet einen Algorithmus mit konstanter Laufzeit pro Pixel, so dass die Laufzeit des Operators nur von der Eingabebildgröße und nicht von der Maskengröße abhängt. Daher ist median_rect für große Masken die schnellste Implementierung des Medianfilters in HALCON. Abhängig von der Rechnerarchitektur (Prozessortyp, Verfügbarkeit von SIMD-Befehlen wie SSE2 oder MMX, Cachegröße und -durchsatz, Speicherdurchsatz) ist für kleine Werte von MaskHeight die in median_image und rank_image verwendete Implementierung schneller als median_rect. Typischerweise ist das für $\text{MaskHeight} \leq 15$ der Fall, kann aber z.B. bei nicht vorhandenen SIMD-Befehlen und langsamem Speicher auch für größere Werte von MaskHeight zutreffen.

Weiterhin ist zu beachten, dass median_rect aufgrund der rekursiven Implementierung die Filterantwort intern auf dem kleinsten umschließenden Rechteck der Region des Bildes berechnet. Falls die Region nur einen sehr kleinen Teil des umschließenden Rechtecks abdeckt, kann es daher vorkommen, dass median_image und rank_image auch für größere Werte von MaskHeight schneller als median_rect sind.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte
Zu filterndes Bild.

▷ **ImageMedian** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte
Gefiltertes Bild.

▷ **MaskWidth** (input_control) . . . . . integer ⇝ *integer*
Breite der Filtermaske.
Defaultwert: 15
Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 31, 49, 51, 61, 71, 81, 91, 101}
Typischer Wertebereich: 3 ≤ `MaskWidth` ≤ 4095
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2

▷ **MaskHeight** (input_control) . . . . . integer ⇝ *integer*
Höhe der Filtermaske.
Defaultwert: 15
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 31, 49, 51, 61, 71, 81, 91, 101}
Typischer Wertebereich: 3 ≤ `MaskHeight` ≤ 4095
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2

Komplexität

Pro Bildpunkt: $O(1)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `median_rect` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`read_image`

Nachfolger

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`median_image`, `rank_rect`, `rank_image`

Siehe auch

`gray_erosion_rect`, `gray_dilation_rect`, `gray_erosion_shape`, `gray_dilation_shape`, `gray_erosion`, `gray_dilation`

Literatur

S. Perreault, P. Hébert; „Median Filtering in Constant Time“; IEEE Transactions on Image Processing vol. 16, no. 9, pp. 2389-2394, 2007.

D. Cline, K.B. White, P.K. Egbert; „Fast 8-Bit Median Filtering Based On Separability“; International Conference on Image Processing, vol. V, pp. 281-284, 2007.

Modul

Foundation

median_separate (Image : ImageSMedian : MaskWidth, MaskHeight, Margin :)

Separierte Medianfilterung mit Rechteckmasken.

median_separate führt eine abgewandelte Variante der Median-Filterung durch: Es werden zunächst zwei Hilfsbilder erzeugt. Das erste entsteht aus einer Medianfilterung mit einer horizontalen, ein Pixel hohen und MaskWidth breiten Maske und anschließender Filterung mit einer vertikalen, MaskHeight hohen und ein Pixel breiten Maske. Das zweite Hilfsbild entsteht durch Filterung mit den gleichen Masken, wobei aber die Reihenfolge der Anwendung umgekehrt wird. Zuerst wird die vertikale, dann die horizontale Maske angewendet. Das Ausgabebild ergibt sich durch pixelweise Mittlung der beiden Hilfsbilder.

median_separate ist deutlich schneller als der normale median_image, da beide Masken ein Pixel breit sind und somit eine sehr effiziente Verarbeitung möglich ist. Die Laufzeit ist praktisch *unabhängig* von der Größe der Maske. median_separate lässt sich beispielsweise gut nach Texturfiltern einsetzen, da dort große Masken benötigt werden.

Zur Verbesserung der Glättung kann der Filter auch mehrfach hintereinander angewandt werden.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Aus Laufzeitgründen wird das Eingabebild Image nicht auf NaNs überprüft. Die Verwendung eines Eingabebildes mit NaNs führt zum Absturz von HALCON.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Zu filterndes Bild.
- ▷ **ImageSMedian** (output_object) . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Mediangefiltertes Bild.
- ▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
  Breite der Rangmaske.
  Defaultwert: 25
  Wertevorschläge: MaskWidth ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 43, 51, 67, 91, 121, 151}
  Typischer Wertebereich: 1 ≤ MaskWidth ≤ 401
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
- ▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
  Höhe der Rangmaske.
  Defaultwert: 25
  Wertevorschläge: MaskHeight ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 43, 51, 67, 91, 121, 151}
  Typischer Wertebereich: 1 ≤ MaskHeight ≤ 401
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
- ▷ **Margin** (input_control) . . . . . string ⇝ *string* / integer / real
  Randbehandlung.
  Defaultwert: 'mirrored'
  Wertevorschläge: Margin ∈ {'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255}

Beispiel

```
read_image(Image,'fabrik')
median_separate(Image,MedianSeparate,5,5,3)
dev_display(MedianSeparate)
```

Komplexität

Pro Bildpunkt: O(40).

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

texture_laws, sobel_amp, deviation_image

Nachfolger

learn_ndim_norm, regiongrowing, auto_threshold

Alternativen

median_image

Siehe auch

rank_image

Literatur

R. Haralick, L. Shapiro; „Computer and Robot Vision“; Addison-Wesley, 1992, Seite 319

Modul

Foundation

median_weighted (Image : ImageWMedian : MaskType, MaskSize :)

Gewichtete Medianfilterung mit verschiedenen Rangmasken.

median_weighted berechnet den Median der Grauwerte innerhalb einer lokalen Umgebung. Im Gegensatz zu median_image, bei dem alle Grauwerte innerhalb der Umgebung genau einmal eingehen, werden bei median_weighted die Grauwerte, abhängig von ihrer Position mehrfach gewichtet. Dabei wird ein Grauwert, entsprechend seiner Gewichtung mehrfach in das zu sortierende Feld aufgenommen. Es stehen folgende Masken zur Verfügung:

'gauss' (MaskSize = 3)

$$\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

'inner' (MaskSize = 3)

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Der median_weighted ist, dass im Gegensatz zu median_image Grauwertecken erhalten bleiben.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
Zu filterndes Bild.

▷ **ImageWMedian** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
Mediangefiltertes Bild.

▷ **MaskType** (input_control) ..... string ⇝ *string*
Art der Median-Maske.
Defaultwert: 'inner'
Werteliste: MaskType ∈ {'inner', 'gauss'}

▷ **MaskSize** (input_control) . . . . . integer ⇝ *integer*
Maskengröße.
Defaultwert: 3
Werteliste: MaskSize ∈ {3}

Beispiel

```
read_image(Image,'fabrik')
median_weighted(Image,MedianWeighted,'gauss',3)
dev_display(MedianWeighted)
```

Komplexität

Pro Bildpunkt: $O(F * log(F))$ mit F = Fläche von MaskType.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

read_image

Nachfolger

threshold, dyn_threshold, regiongrowing

Alternativen

median_image, trimmed_mean, sigma_image

Literatur

R. Haralick, L. Shapiro; „Computer and Robot Vision“; Addison-Wesley, 1992, Seite 319

Modul

Foundation

midrange_image (Image, Mask : ImageMidrange : Margin :)

Mittelwert aus Maximum und Minimum innerhalb einer beliebigen Maske.

midrange_image bildet den Mittelwert aus Maximum und Minimum innerhalb der angegebenen Maske im gesamten Bild. Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (Margin) gewählt werden:

Grauwert	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
'continued'	Fortsetzung der Randpunkte.
'cyclic'	zyklische Fortsetzung der Bildränder.
'mirrored'	Spiegelung der Bildpunkte an den Bildrändern.

Die angegebene Maske (= Region des Maskenobjekts) wird so über die zu filternden Bilder geschoben, dass der Schwerpunkt der Maske alle Bildpunkte einmal berührt.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
Zu filterndes Bild.

▷ **Mask** (input_object) . . . . . region ⇝ *object*
Filtermaske.

▷ **ImageMidrange** (output_object) . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
Gefiltertes Bild.

▷ **Margin** (input_control) . . . . . string ⇝ *string* / integer / real
Randbehandlung.
Defaultwert: 'mirrored'
Wertevorschläge: `Margin` ∈ {'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255}

Beispiel

```
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
midrange_image(Image,Region,Midrange,'mirrored')
dev_display(Midrange)
```

Komplexität

Pro Bildpunkt: $O(sqrt(F) * 5)$ mit F = Fläche von `Mask`.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `midrange_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`read_image`, `draw_region`, `gen_circle`, `gen_rectangle1`

Nachfolger

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`sigma_image`

Siehe auch

`gen_circle`, `gen_rectangle1`, `gray_erosion_rect`, `gray_dilation_rect`, `gray_range_rect`

Literatur

R. Haralick, L. Shapiro; „Computer and Robot Vision“; Addison-Wesley, 1992, Seite 319

Modul

Foundation

rank_image (Image, Mask : ImageRank : Rank, Margin :)

Rangfilterung mit beliebigen Masken.

`rank_image` führt eine Rangfilterung des Eingabebildes `Image` mit einer Maske, die durch die Region `Mask` beschrieben wird, durch und liefert das gefilterte Bild in `ImageRank` zurück. Die Form der Maske kann beliebig

gewählt werden und z.B. mit Operatoren wie gen_circle oder draw_region erzeugt werden. Die Position der Maske spielt dabei keine Rolle, da der Schwerpunkt der Maskenregion als Referenzpunkt der Maske verwendet wird.

Konzeptuell sortiert der Rangfilter alle Grauwerte innerhalb der Maske aufsteigend und wählt dann den Grauwert mit dem Rang Rank aus. Dabei entspricht der Rang 1 dem kleinsten Grauwert und der Rang A dem größten Grauwert innerhalb der Maske. Hierbei ist A die Fläche von Mask (siehe area_center). Für $\texttt{Rank} = (A-1)/2+1$ liefert rank_image den Median zurück (siehe median_image). Für $\texttt{Rank} = 1$ entspricht rank_image einer Grauwerterosion (siehe gray_erosion_rect, gray_erosion_shape und gray_erosion), für $\texttt{Rank} = A$ entspricht rank_rect einer Grauwertdilatation (siehe gray_dilation_rect, gray_dilation_shape und gray_dilation).

rank_image kann unter anderem zur Rauschunterdrückung oder zum Unterdrücken unerwünschter Objekte, die kleiner als die Maske sind, verwendet werden. Außerdem ist rank_image unempfindlicher gegenüber Rauschen als die entsprechenden Operatoren der Grauwertmorpholgie. Um eine robustere Version der Grauwertmorphologie zu erhalten, empfiehlt es sich daher, anstatt 1 bzw. A etwas höhere bzw. niedrigere Werte für Rank zu wählen.

Bei der Filterung kann über Margin zwischen verschiedenen Randbehandlungen gewählt werden:

Grauwert	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
'continued'	Fortsetzung der Randpunkte.
'cyclic'	zyklische Fortsetzung der Bildränder.
'mirrored'	Spiegelung der Bildpunkte an den Bildrändern.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

rank_image verwendet einen Algorithmus mit einer Laufzeit pro Pixel, die von der Anzahl der Sehnen der Maske Mask abhängt. Daher ist rank_image für rechteckige Masken mit einer großen Maskenhöhe langsamer als rank_rect. Die genaue Maskenhöhe, ab der rank_rect schneller ist, ist abhängig von der Rechnerarchitektur (Prozessortyp, Verfügbarkeit von SIMD-Befehlen wie SSE2 oder MMX, Cachegröße und -durchsatz, Speicherdurchsatz). Typischerweise ist das für eine Maskenhöhe > 15 der Fall, kann aber z.B. bei nicht vorhandenen SIMD-Befehlen und langsamem Speicher auch erst für größere Maskenhöhen zutreffen.

Weiterhin ist zu beachten, dass rank_rect aufgrund der rekursiven Implementierung die Filterantwort intern auf dem kleinsten umschließenden Rechteck der Region des Bildes berechnet. Falls die Region nur einen sehr kleinen Teil des umschließenden Rechtecks abdeckt, kann es daher vorkommen, dass rank_image auch für größere Maskenhöhen schneller als rank_rect ist.

rank_image sollte weder mit $\texttt{Rank} = 1$ zur Berechnung einer Grauwerterosion noch mit $\texttt{Rank} = A$ zur Berechnung einer Grauwertdilatation verwendet werden. In diesen Fällen sind die Operatoren gray_erosion_rect, gray_erosion_shape oder gray_erosion bzw. gray_dilation_rect, gray_dilation_shape oder gray_dilation typischerweise schneller als rank_image.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Zu filterndes Bild.
- ▷ **Mask** (input_object) . . . . . region ⇝ *object*
  Filtermaske.
- ▷ **ImageRank** (output_object) . . . . . multichannel-image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Gefiltertes Bild.
- ▷ **Rank** (input_control) . . . . . integer ⇝ *integer*
  Rang des Ausgabegrauwerts.
  Defaultwert: 5
  Wertevorschläge: Rank ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 31, 49, 51, 61, 71, 81, 91, 101}
  Typischer Wertebereich: $1 \leq \texttt{Rank} \leq 4095$
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 2

▷ **Margin** (input_control) . . . . . string ⇝ *string* / integer / real
Randbehandlung.
Defaultwert: 'mirrored'
Wertevorschläge: Margin ∈ {'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255}

Beispiel

```
read_image (Image, 'fabrik')
draw_region (Region, WindowHandle)
rank_image (Image, Region, ImageRank, 5, 'mirrored')
dev_display(ImageRank)
```

Komplexität

Pro Bildpunkt: $O(N)$, wobei N die Anzahl der Sehnen von Mask ist (siehe runlength_features).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert rank_image den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

read_image, draw_region, gen_circle, gen_rectangle1, gen_rectangle2

Nachfolger

threshold, dyn_threshold, regiongrowing

Alternativen

rank_rect, median_image, median_rect

Siehe auch

gray_erosion_rect, gray_dilation_rect, gray_erosion_shape, gray_dilation_shape, gray_erosion, gray_dilation

Literatur

T.S. Huang, G.J. Yang, G.Y. Tang; „A Fast Two-Dimensional Median Filtering Algorithm“; IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 27, no. 1, pp. 13-18, 1979.

R. Haralick, L. Shapiro; „Computer and Robot Vision“; Addison-Wesley, 1992, pp. 318-320.

Modul

Foundation

rank_n (Image : RankImage : RankIndex :)

Rückgabe der Grauwerte mit angegebenem Rang aus mehreren Kanälen.

rank_n gibt pixelweise das Ergebnis der Rang-Funktion über alle Kanäle des Eingabebildes Image zurück.

Für jedes Pixel des Eingabebildes gilt: Die Grauwerte aller Kanäle an dieser Position werden aufsteigend sortiert. Dann wird das Pixel mit Index RankIndex ausgewählt und im Ausgabebild an die gleiche Position geschrieben. Das Ausgabebild hat einen Kanal.

Die SpezialfälleRankIndex = *1* und RankIndex = *'(Anzahl Eingabekanäle)'* liefern jeweils das Minimum bzw. Maximum. RankIndex = *'(Anzahl Eingabekanäle + 1) / 2'* liefert den Median (hierbei bezeichnet / die ganzzahlige Division). Für ein fünfkanaliges Bild liefert also *3* den Median.

Der Operator rank_n ist nicht mit dem Operator rank_image zu verwechseln der die Rang-Funktion innerhalb einer bestimmten Maske durchführt.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / int8 / real
  Mehrkanaliges Graubild.
- ▷ **RankImage** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / int4 / int8 / real
  Ergebnis der Rang-Funktion.
- ▷ **RankIndex** (input_control) . . . . . integer ⇝ *integer*
  Rang der zurückzugebenden Grauwertbilder.
  Defaultwert: 2
  Wertevorschläge: RankIndex ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20}

Beispiel

```
compose5 (Image1, Image2, Image3, Image4, Image5, MultiChannelImage)
rank_n (MultiChannelImage, ImageMin, 1)
rank_n (MultiChannelImage, ImageMax, 5)
rank_n (MultiChannelImage, ImageMedian, 3)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

compose2, compose3, compose4, compose5, add_channels

Alternativen

mean_n

Siehe auch

count_channels, rank_image

Modul

Foundation

rank_rect (Image : ImageRank : MaskWidth, MaskHeight, Rank :)

Rangfilterung mit rechteckigen Rangmasken.

rank_rect führt eine Rangfilterung des Eingabebildes Image mit einer rechteckigen Maske der Größe MaskWidth × MaskHeight durch und liefert das gefilterte Bild in ImageRank zurück.

Konzeptuell sortiert der Rangfilter alle Grauwerte innerhalb der Maske aufsteigend und wählt dann den Grauwert mit dem Rang Rank aus. Dabei entspricht der Rang 1 dem kleinsten Grauwert und der Rang MaskWidth · MaskHeight dem größten Grauwert innerhalb der Maske. Für Rank = (MaskWidth · MaskHeight − 1)/2 + 1 liefert rank_rect den Median zurück (siehe median_rect). Für Rank = 1 entspricht rank_rect einer Grauwerterosion (siehe gray_erosion_rect, gray_erosion_shape und gray_erosion), für Rank = MaskWidth · MaskHeight entspricht rank_rect einer Grauwertdilatation (siehe gray_dilation_rect, gray_dilation_shape und gray_dilation).

rank_rect kann unter anderem zur Rauschunterdrückung oder zum Unterdrücken unerwünschter Objekte, die kleiner als die Maske sind, verwendet werden. Außerdem ist rank_rect unempfindlicher gegenüber Rauschen

als die entsprechenden Operatoren der Grauwertmorpholgie. Um eine robustere Version der Grauwertmorphologie zu erhalten, empfiehlt es sich daher, anstatt 1 bzw. MaskWidth · MaskHeight etwas höhere bzw. niedrigere Werte für Rank zu wählen.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Werden für MaskHeight oder MaskWidth gerade statt ungerade Werte übergeben, verwendet rank_rect an ihrer Stelle die nächstgrößeren ungeraden Werte.

rank_rect verwendet einen Algorithmus mit konstanter Laufzeit pro Pixel, so dass die Laufzeit des Operators nur von der Eingabebildgröße und nicht von der Maskengröße abhängt. Daher ist rank_rect für große Masken die schnellste Implementierung des Rangfilters in HALCON. Abhängig von der Rechnerarchitektur (Prozessortyp, Verfügbarkeit von SIMD-Befehlen wie SSE2 oder MMX, Cachegröße und -durchsatz, Speicherdurchsatz) ist für kleine Werte von MaskHeight die in rank_image verwendete Implementierung schneller als rank_rect. Typischerweise ist das für MaskHeight $\leq$ 15 der Fall, kann aber z.B. bei nicht vorhandenen SIMD-Befehlen und langsamem Speicher auch für größere Werte von MaskHeight zutreffen.

Weiterhin ist zu beachten, dass rank_rect aufgrund der rekursiven Implementierung die Filterantwort intern auf dem kleinsten umschließenden Rechteck der Region des Bildes berechnet. Falls die Region nur einen sehr kleinen Teil des umschließenden Rechtecks abdeckt, kann es daher vorkommen, dass rank_image auch für größere Werte von MaskHeight schneller als rank_rect ist.

rank_rect sollte weder mit Rank = 1 zur Berechnung einer Grauwerterosion noch mit Rank = MaskWidth · MaskHeight zur Berechnung einer Grauwertdilatation verwendet werden. In diesen Fällen sind die Operatoren gray_erosion_rect oder gray_erosion_shape bzw. gray_dilation_rect oder gray_dilation_shape für fast alle Maskengrößen schneller als rank_rect.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
Zu filterndes Bild.

▷ **ImageRank** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
Gefiltertes Bild.

▷ **MaskWidth** (input_control) ..... integer ⇝ *integer*
Breite der Filtermaske.
Defaultwert: 15
Wertevorschläge: MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 31, 49, 51, 61, 71, 81, 91, 101}
Typischer Wertebereich: 3 ≤ MaskWidth ≤ 4095
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2

▷ **MaskHeight** (input_control) ..... integer ⇝ *integer*
Höhe der Filtermaske.
Defaultwert: 15
Wertevorschläge: MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 31, 49, 51, 61, 71, 81, 91, 101}
Typischer Wertebereich: 3 ≤ MaskHeight ≤ 4095
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2

▷ **Rank** (input_control) ..... integer ⇝ *integer*
Rang des Ausgabegrauwerts.
Defaultwert: 5
Wertevorschläge: Rank ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 31, 49, 51, 61, 71, 81, 91, 101}
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2
Restriktion: 1 <= Rank && Rank <= MaskWidth * MaskHeight

Komplexität

Pro Bildpunkt: $O(1)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert rank_rect den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`read_image`

Nachfolger

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`rank_image`, `median_rect`, `median_image`

Siehe auch

`gray_erosion_rect`, `gray_dilation_rect`, `gray_erosion_shape`, `gray_dilation_shape`, `gray_erosion`, `gray_dilation`

Literatur

S. Perreault, P. Hébert; „Median Filtering in Constant Time“; IEEE Transactions on Image Processing, vol. 16, no. 9, pp. 2389-2394, 2007.

D. Cline, K.B. White, P.K. Egbert; „Fast 8-Bit Median Filtering Based On Separability“; International Conference on Image Processing, vol. V, pp. 281-284, 2007.

Modul

Foundation

sigma_image (Image : ImageSigma : MaskHeight, MaskWidth, Sigma :)

Nichtlineare Glättung mit dem Sigmafilter.

`sigma_image` führt eine nichtlineare Glättung der Grauwerte aller Eingabebilder (`Image`) durch. Alle Bildpunkte in einem rechteckigen Fenster (`MaskHeight` × `MaskWidth`) werden verwendet, um den neuen Grauwert des zentralen Bildpunktes in diesem Fenster zu bestimmen. Zunächst wird die Standardabweichung der Grauwerte aller Bildpunkte in diesem Fenster berechnet. Dann werden alle Bildpunkte des Fensters, deren Grauwert von dem des zentralen Bildpunktes um weniger als `Sigma` abweicht, verwendet, um den neuen Wert des zentralen Bildpunktes zu berechnen. Der Grauwert des zentralen Bildpunktes ergibt sich aus dem Mittelwert dieser ausgewählten Bildpunkte. Wenn kein Bildpunkt für die Mittelwertbestimmung der Grauwerte ausgewählt werden konnte, wird der Grauwert des zentralen Bildpunktes unverändert übernommen.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Werden für `MaskHeight` oder `MaskWidth` gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / cyclic / int1 / int2 / uint2 / int4 / real

Zu glättendes Bild.

▷ **ImageSigma** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / cyclic / int1 / int2 / uint2 / int4 / real

Geglättetes Bild.

▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
Fenstergröße für Mittlung (Anzahl Zeilen).
Defaultwert: 5
Wertevorschläge: MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskHeight ≤ 101
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskHeight)

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
Fenstergröße für Mittlung (Anzahl Spalten).
Defaultwert: 5
Wertevorschläge: MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskWidth ≤ 101
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth)

▷ **Sigma** (input_control) . . . . . integer ⇝ *integer*
Maximale Differenz zum Mittelwert.
Defaultwert: 3
Wertevorschläge: Sigma ∈ {3, 5, 7, 9, 11, 20, 30, 50}
Typischer Wertebereich: 0 ≤ Sigma ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2

Beispiel

```
read_image(Image,'fabrik')
sigma_image(Image,ImageSigma,5,5,3)
dev_display(ImageSigma)
```

Komplexität

Pro Bildpunkt: O(MaskHeight × MaskWidth).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert sigma_image den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

read_image

Nachfolger

threshold, dyn_threshold, regiongrowing

Alternativen

anisotropic_diffusion, rank_image

Siehe auch

smooth_image, binomial_filter, gauss_filter, mean_image

Literatur

R. Haralick, L. Shapiro; „Computer and Robot Vision“; Addison-Wesley, 1992, Seite 325

Modul

Foundation

smooth_image (Image : ImageSmooth : Filter, Alpha :)

Glättung mit verschiedenen Filtern.

smooth_image dient zur Glättung von Graustufenbildern mittels rekursiver Filter (nach Deriche und Shen) bzw. mittels des konventionell implementierten Gaußfilters. Konkret lassen sich über den Parameter Filter folgende vier Filter auswählen:

'deriche1', 'deriche2', 'shen' und 'gauss'.

Die „Filterbreite“ (d.h. das Einzugsgebiet und damit die glättende Wirkung der Filter) ist frei wählbar. Sie nimmt bei den Deriche- und dem Shen-Filter mit wachsendem Filterparameter Alpha ab, beim Gaußfilter hingegen zu (Alpha ist in diesem Fall die Standardabweichung der Gaußfunktion). Eine Abschätzung des Einzugsgebiets der Filter für konkrete Werte von Alpha liefert die Routine info_smooth.

Nicht-rekursive Filter, wie hier der Gaußfilter, werden häufig mittels Filtermasken realisiert. In diesem Fall erhöht sich die Laufzeit natürlich mit wachsender Filterbreite. Die Laufzeit der rekursiven Filter ist hingegen praktisch konstant. Lediglich die Randbehandlung (Spiegelung der Randpunkte) ist bei „breiteren“ Filter etwas aufwändiger. Das Einzugsgebiet der Deriche- bzw. des Shen-Filters ist also fast ohne Mehraufwand beliebig vergrößerbar. Der daraus resultierende Laufzeitvorteil gegenüber dem Gaußfilter nimmt naturgemäß mit wachsender „Filterbreite“ zu. Allerdings ist nur der Gaußfilter isotrop, alle anderen Filter sind anisotrop (wobei der 'deriche2'-Filter nur schwach richtungs-sensitiv ist). Eine vergleichbare glättende Wirkung der Filter erhält man bei folgender Wahl von Alpha:

$$\begin{aligned} \texttt{Alpha}('deriche2') &= \frac{\texttt{Alpha}('deriche1')}{2} \\ \texttt{Alpha}('shen') &= \frac{\texttt{Alpha}('deriche1')}{2} \\ \texttt{Alpha}('gauss') &= \frac{1.77}{\texttt{Alpha}('deriche1')} \end{aligned}$$

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Zu glättende Bilder.
- **ImageSmooth** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Geglättete Bilder.
- **Filter** (input_control) ..... string ⇝ *string*
  Gewünschter Filter.
  Defaultwert: 'deriche2'
  Werteliste: Filter ∈ {'deriche1', 'deriche2', 'shen', 'gauss'}
- **Alpha** (input_control) ..... real ⇝ *real*
  Filterparameter: kleine Werte bewirken starke Glättung (bei *'gauss'* umgekehrt).
  Defaultwert: 0.5
  Wertevorschläge: Alpha ∈ {0.1, 0.2, 0.3, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0, 10.0}
  Typischer Wertebereich: $0.01 \leq$ Alpha ≤ 50.0
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Alpha > 0

Beispiel

```
info_smooth('deriche2',0.5,Size,Coeffs)
smooth_image(Input,Smooth,'deriche2',7)
```

Ergebnis

Sind die Parameterwerte korrekt liefert `smooth_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_image`

Nachfolger

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`binomial_filter`, `gauss_filter`, `mean_image`, `derivate_gauss`, `isotropic_diffusion`

Siehe auch

`info_smooth`, `median_image`, `sigma_image`, `anisotropic_diffusion`

Literatur

R.Deriche: „Fast Algorithms for Low-Level Vision“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-12, no. 1; S. 78-87; 1990.

Modul

Foundation

trimmed_mean (Image, Mask : ImageTMean : Number, Margin :)

Glättung mit einer beliebigen Rangmaske.

`trimmed_mean` führt eine nichtlineare Glättung der Grauwerte aller Eingabebilder (`Image`) durch. Die Verschiebungsmaske (`Mask`) wird in Form einer Region übergeben. Berechnet wird der Mittelwert von `Number` Grauwert, die in der Nähe des Medians liegen. Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (`Margin`) gewählt werden:

Grauwert	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
'continued'	Fortsetzung der Randpunkte.
'cyclic'	zyklische Fortsetzung der Bildränder.
'mirrored'	Spiegelung der Bildpunkte an den Bildrändern.

Die angegebene Maske (= Region des Maskenbildes) wird so über die zu filternden Bilder geschoben, dass der Schwerpunkt der Maske alle Bildpunkte einmal berührt. Für jeden solchen Bildpunkt werden alle Nachbarpunkte, die von der Maske überdeckt werden, bzgl. ihrer Grauwerte aufsteigend sortiert. Jede solche sortierte Sequenz von Grauwerten enthält somit genau so viele Grauwerte, wie die Maske Bildpunkte hat. Sei F die Fläche der Maske, dann wird aus diesen Sequenzen der Mittelwert folgendermaßen berechnet: Die ersten (F - Number)/2 Grauwerte werden ignoriert. Dann werden die folgenden `Number` Grauwerte aufsummiert und durch `Number` dividiert. Die verbleibenden (F - `Number`)/2 Grauwerte werden wiederum ignoriert.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
Zu filterndes Bild.

▷ **Mask** (input_object) . . . . . region ⇝ *object*
Bild, dessen Region als Filtermaske dient.

▷ **ImageTMean** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
Gefiltertes Ergebnisbild.

▷ **Number** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Punkte über die gemittelt wird. Typischer Wert: Fläche(Mask) / 2.
Defaultwert: 5
Wertevorschläge: `Number` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31}
Typischer Wertebereich: 1 ≤ `Number` ≤ 401
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2

▷ **Margin** (input_control) . . . . . string ⇝ *string* / integer / real
Randbehandlung.
Defaultwert: 'mirrored'
Wertevorschläge: `Margin` ∈ {'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255}

Beispiel

```
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
trimmed_mean(Image,Region,TrimmedMean,5,'mirrored')
dev_display(TrimmedMean)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `trimmed_mean` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`read_image`, `draw_region`, `gen_circle`, `gen_rectangle1`

Nachfolger

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`sigma_image`, `median_weighted`, `median_image`

Siehe auch

`gen_circle`, `gen_rectangle1`, `gray_erosion_rect`, `gray_dilation_rect`

Literatur

R. Haralick, L. Shapiro; „Computer and Robot Vision“; Addison-Wesley, 1992, Seite 320

Modul

Foundation

10.7 Inpainting

harmonic_interpolation (Image,
Region : InpaintedImage : Precision :)

Harmonische Interpolation auf einer Bildregion.

Der Operator harmonic_interpolation rekonstruiert die in Image zerstörten Bilddaten innerhalb der Region Region, indem er dort die diskrete Laplacegleichung $u_{xx} + u_{yy} = 0$ auf der Grauwertfunktion u löst. Die mit den Dirichtlet-Randwerten aus Image außerhalb von Region eindeutige Lösung wird in InpaintedImage zurückgeliefert.

Diese Methode nennt man harmonische Interpolation, da in der Funktionentheorie die Lösungen der Laplacegleichung als harmonische Funktionen bezeichnet werden.

Falls Region den Rand der Grauwertmatrix von Image berührt und damit einige der Dirichlet-Randwerte nicht existieren, werden stattdessen von Neumann-Randwerte verwendet. Dies bedeutet, dass die Grauwerte am Rand von Image gespiegelt werden. Existieren überhaupt keine Dirichlet-Randwerte, so wird ein konstantes Bild mit Grauwert *0* zurückgeliefert.

Die Ortsableitungen werden diskretisiert durch $u_{xx}(x,y) = u(x-1,y) - 2u(x,y) + u(x+1,y)$ und $u_{yy}(x,y) = u(x,y-1) - 2u(x,y) + u(x,y+1)$. Die Gleichung wird mit Hilfe eines iterativen Konjugierte Gradienten Lösers behandelt, der die Genauigkeit schrittweise verbessert, bis die Maximum-Norm des Update-Schrittes unter den Anteil Precision der Norm der Eingabedaten fällt oder die maximale Anzahl von *1000* Schritten erreicht ist. Precision $= 0.01$ entspricht so einer relativen Rechengenauigkeit von 1%.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **Region** (input_object) ..... region ⇝ *object*
  Inpaintingregion.
- ▷ **InpaintedImage** (output_object) ..... image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **Precision** (input_control) ..... real ⇝ *real*
  Rechengenauigkeit.
  Defaultwert: 0.001
  Wertevorschläge: Precision ∈ {0.0, 0.0001, 0.001, 0.01}
  Restriktion: Precision >= 0.0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

inpainting_ct, inpainting_aniso, inpainting_mcf, inpainting_texture, inpainting_ced

Literatur

L.C. Evans; „Partial Differential Equations“; AMS, Providence; 1998.

W. Hackbusch; „Iterative Lösung großer schwachbesetzter Gleichungssysteme“; Teubner, Stuttgart;1991.

Modul

Foundation

inpainting_aniso (Image, Region : InpaintedImage : Mode, Contrast, Theta, Iterations, Rho :)

Inpainting eines Bildbereiches durch anisotrope Diffusion.

Der Operator inpainting_aniso nutzt die anisotrope Diffusion von Grauwerten nach dem Modell von Perona und Malik, um an die Bildregion Region des Eingabebildes Image angrenzende Bildkanten in Region hinein fortzusetzen und zu verbinden.

Dadurch kann die Struktur der Bildkanten in Region konsistent zur umgebenden Bildmatrix gemacht werden, so dass eine Überdeckung von Fehlstellen oder Störobjekten im Eingabebild, ein sogenanntes Inpainting, für einen menschlichen Beobachter schwächer in Erscheinung tritt, da keine auffälligen Artefakte oder Schmiereffekte zurückbleiben.

Ordnet man dem Bild eine Grauwertfunktion u zu, so ist der Algorithmus eine Diskretisierung der partiellen Differentialgleichung

$$u_t = div(g(|\nabla u|^2, c)\nabla u)$$

mit dem durch Image definierten Anfangswert $u = u_0$ zu einem Zeitpunkt $t_0 = 0$. Die Gleichung wird dabei Iterations mal in Zeitschritten der Länge Theta iteriert, so dass das Ausgabebild InpaintedImage die Grauwertfunktion zum Zeitpunkt Iterations $\cdot$ Theta enthält.

Ziel der anisotropen Diffusion, die auch als nichtlinear isotrope Diffusion bezeichnet wird, ist es ursprünglich, in flächigen Bildbereichen Bildstörungen wie Rauschen zu eliminieren, ohne dabei Bildkanten zu verwischen. Die Unterscheidung zwischen Kanten und Flächen wird durch den Schwellenwert Contrast für die Größe der Grauwertdifferenz zwischen benachbarten Pixeln getroffen. Contrast wird auch als Kontrastparameter bezeichnet und mit c abgekürzt. Wird die Kanteninformation durch Glättung der Kantenamplitude in der Umgebung der bereits vorhandenen Kanten ausgebreitet, so ist es darüber hinaus möglich, die Kanten in das Rechengebiet Region hinein fortzusetzen. Die Standardabweichung dieses Glättungsvorganges wird durch den Parameter Rho bestimmt.

Der verwendete Algorithmus entspricht im Wesentlichen dem im anisotropen Diffusionsfilter anisotropic_diffusion, allerdings geschieht hier keine Randbehandlung durch Spiegelung der Grauwerte am Rand von Region. Stattdessen ist dieses Verfahren nur auf Regionen anwendbar, die einen Abstand von mindestens 3 Pixeln zum Rand der Bildmatrix von Image einhalten, da die Grauwerte auf diesem Streifen um Region genutzt werden, um Randwerte für die verwendete Differentialgleichung zu definieren und so die Konsistenz zur Umgebung von Region herzustellen. Dabei ist zu beachten, dass der Inpaintingvorgang sich auf diejenigen Pixel beschränkt, die innerhalb der ROI des Eingabebildes Image liegen. Umfasst diese nicht die gesamte Region Region, so wird ein Streifen um die Schnittmenge aus Region und der ROI verwendet, um die Randwerte zu definieren.

Das Ergebnis des Diffusionsvorganges ist auch hier abhängig von den Grauwerten im Rechengebiet des Eingabebildes Image. Es ist zu beachten, dass bereits vorhandene Bildkanten dort erhalten werden. Dies gilt insbesondere auch für Grauwertsprünge am Rand von Region, die etwa durch Inpainting mit einem konstanten Grauwert entstehen. Soll das Verfahren zum Inpainting verwendet werden, empfiehlt es sich, zuvor den Operator harmonic_interpolation anzuwenden, um sämtliche störenden Kanten aus dem Rechengebiet zu entfernen und die maximale Grauwertdifferenz zwischen Nachbarpixeln zu minimieren, es sei denn das ursprüngliche Bild enthält in Region bereits Informationen, die erhalten werden sollen.

Der variable Diffusionskoeffizient g der Differentialgleichung kann verschiedenen monoton fallenden Funktionen mit Werten zwischen *0* und *1* folgen und bestimmt das Antwortverhalten des Diffusionsvorganges auf eine Kante. Über den Parameter Mode stehen die folgenden Funktionen zur Verfügung:

$$g_1(x, c) = \frac{1}{\sqrt{1 + 2\frac{x}{c^2}}}$$

Die Wahl der Funktion g_1, wenn Mode auf *'parabolic'* gesetzt wird, garantiert, dass die zugehörige Differentialgleichung parabolisch ist, so dass eine Wohlgestelltheitstheorie für das Problem existiert und das Verfahren für beliebige Schrittweiten Theta stabil ist. In diesem Falle bleibt es aber bei einer leichten Diffusion auch über Kanten hinweg, deren Stärke c übersteigt.

$$g_2(x, c) = \frac{1}{1 + \frac{x}{c^2}}$$

Die in der Veröffentlichung von Perona und Malik verwendete Funktion bei Wahl von *'perona-malik'* für `Mode` besitzt nicht die theoretischen Eigenschaften von g_1, hat sich in der Praxis aber dennoch als ausreichend stabil erwiesen. Die theoretische Instabilität zeigt sich in einer leichten Schärfung von starken Kanten.

$$g_3(x,c) = 1 - \exp(-C\frac{c^8}{x^4})$$

Die von Weickert vorgeschlagene Wahl g_3 bei `Mode`=*'weickert'* mit der Konstanten $C = 3.31488$ ist eine Steigerung von g_2 im Bezug auf die Kantenschärfe. Der Übergang zwischen Glättung und Schärfung erfolgt hier sehr abrupt an der Stelle $x = c^2$.

Darüber hinaus ist für `Mode` noch der Wert *'shock'* möglich, durch den eine kontrastinvariante Abwandlung der anisotropen Diffusion gewählt wird. In dieser Variante wird die Erzeugung von Kanten nicht durch die Variation des Diffusionskoeffizienten g erreicht, sondern ein konstanter Koeffizient $g = 1$ und damit isotrope Diffusion verwendet. Zusätzlich wird ein Schockfilter vom Typ

$$u_t = -sgn(\nabla|\nabla u|)|\nabla u|$$

eingesetzt, der ebenso wie ein negativer Diffusionskoeffizient der Kantenschärfung dient, diese Schärfung aber unabhängig vom Absolutwert von $|\nabla u|$ bewirkt. In diesem Modus hat `Contrast` nicht die Bedeutung eines Kontrastparameters, sondern bestimmt das Verhältnis zwischen Diffusions- und Schockfilteranteil in jedem Iterationsschritt. Der Wert *0* entspräche demnach reiner isotroper Diffusion, wie sie im Operator `isotropic_diffusion` verwendet wird. Der Parameter ist so skaliert, dass sich Diffusion und Schärfung für `Contrast` = *1* gerade aufheben. Ein Wert `Contrast` > *1* sollte nicht verwendet werden, da das Verfahren dadurch instabil wird.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **`Region`** (input_object) ..... region ⇝ *object*
  Inpaintingregion.
- ▷ **`InpaintedImage`** (output_object) ..... image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Art des Kantenschärfungsalgorithmus.
  Defaultwert: 'weickert'
  Werteliste: `Mode` ∈ {'weickert', 'perona-malik', 'parabolic', 'shock'}
- ▷ **`Contrast`** (input_control) ..... real ⇝ *real*
  Kontrastparameter.
  Defaultwert: 5.0
  Wertevorschläge: `Contrast` ∈ {0.5, 2.0, 5.0, 10.0, 20.0, 50.0, 100.0}
  Restriktion: Contrast > 0
- ▷ **`Theta`** (input_control) ..... real ⇝ *real*
  Schrittweite.
  Defaultwert: 0.5
  Wertevorschläge: `Theta` ∈ {0.5, 1.0, 5.0, 10.0, 30.0, 100.0}
  Restriktion: Theta > 0
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Anzahl Iterationen.
  Defaultwert: 10
  Wertevorschläge: `Iterations` ∈ {1, 3, 10, 100, 500}
  Restriktion: Iterations >= 1
- ▷ **`Rho`** (input_control) ..... real ⇝ *real*
  Glättungskoeffizient für Kanteninformation.
  Defaultwert: 3.0
  Wertevorschläge: `Rho` ∈ {0.0, 0.1, 0.5, 1.0, 3.0, 10.0}
  Restriktion: Rho >= 0

Beispiel

```
read_image (Image, 'fabrik')
gen_rectangle1 (Rectangle, 270, 180, 320, 230)
harmonic_interpolation (Image, Rectangle, InpaintedImage, 0.01)
inpainting_aniso (InpaintedImage, Rectangle, InpaintedImage2, \
                  'perona-malik', 5.0, 100, 50, 0.5)
dev_display(InpaintedImage2)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Alternativen

harmonic_interpolation, inpainting_ct, inpainting_mcf, inpainting_texture, inpainting_ced

Literatur

J. Weickert; „Anisotropic Diffusion in Image Processing“; PhD Thesis; Fachbereich Mathematik, Universität Kaiserslautern; 1996.

P. Perona, J. Malik; „Scale-space and edge detection using anisotropic diffusion“; Transactions on Pattern Analysis and Machine Intelligence 12(7), pp. 629-639; IEEE; 1990.

G. Aubert, P. Kornprobst; „Mathematical Problems in Image Processing“; Applied Mathematical Sciences 147; Springer, New York; 2002.

Modul

Foundation

inpainting_ced (Image, Region : InpaintedImage : Sigma, Rho, Theta, Iterations :)

Inpainting eines Bildbereiches durch kohärenzverstärkende Diffusion.

Der Operator inpainting_ced führt einen anisotropen Diffusionsvorgang in der Region Region des Eingabebildes Image durch mit dem Ziel, durch Verstärkung der Kohärenz der in Image enthaltenen Bildstrukturen unterbroche Bildkanten diffusiv miteinander zu verbinden ohne sie dabei senkrecht zur Kantenrichtung zu verschmieren. Dieser Mechanismus entspricht dem des Operators coherence_enhancing_diff, der auf einer Diskretisierung der anisotropen Diffusionsgleichung

$$u_t = div(G(u)\nabla u)$$

nach Weickert beruht. Diese Gleichung, mit einer von den Grauwerten in Image abhängigen 2×2 Koeffizientenmatrix G, ist eine Verallgemeinerung der Krümmungsfluss- oder intrinsischen Wärmeleitungsgleichung

$$u_t = div(\frac{\nabla u}{|\nabla u|})|\nabla u| = curv(u)|\nabla u|$$

mit der durch das Eingabebild Image zu einem Zeitpunkt $t_0 = 0$ definierten Grauwertfunktion u. Der Operator mean_curvature_flow ist eine direkte Anwendung der Krümmungsflussgleichung, mit Hilfe des Operators inpainting_mcf, kann sie auch für das Image Inpainting genutzt werden. Die diskrete Gleichung wird in Iterations Zeitschritten der Länge Theta gelöst, so dass das Ausgabebild InpaintedImage die Grauwertfunktion zur Zeit Iterations · Theta enthält.

Um die Kantenrichtungen insbesondere bei verrauschten Eingabedaten stabiler zu bestimmen, kann der Berechnung der Grauwertgradienten ein zusätzlicher isotroper Glättungsschritt vorgeschaltet werden. Der Parameter

Sigma bestimmt die Stärke dieser Glättung als Standardabweichung des zugehörigen Gauß-Faltungskernes, wie er auch im Operator isotropic_diffusion zur isotropen Glättung eines Bildes verwendet wird.

Ebenso wie im Operator inpainting_mcf wird hier die Struktur der Bilddaten in Region durch Glättung der Niveaulinien vereinfacht. So lassen sich störende Objekte aus dem Bild entfernen, wobei die Kanten der Umgebung stetig fortgesetzt werden. Diesen Vorgang bezeichnet man als Image Inpainting. Ziel ist es, dass keine auffälligen Artefakte oder Schmiereffekte zurückbleiben, damit die Bildmanipulation für einen menschlichen Beobachter möglichst schwach sichtbar wird.

Während die Matrix G im Fall des Krümmungsflusses inpainting_mcf gegeben wird durch

$$G_{MCF}(u) = I - \frac{1}{|\nabla u|^2} \nabla u (\nabla u)^T,$$

wobei I die Einheitsmatrix bezeichnet, wird diese Matrix in inpainting_ced komponentenweise in Ortsrichtung durch einen Gaußfilter mit Standardabweichung Rho geglättet und anschließend aus den Eigenwerten λ_1, λ_2 und Eigenvektoren w_1, w_2 der so entstandenen Matrix die endgültige Koeffizientenmatrix

$$G_{CED} = g_1\left((\lambda_1 - \lambda_2)^2\right) w_1 (w_1)^T + g_2\left((\lambda_1 - \lambda_2)^2\right) w_2 (w_2)^T$$

konstruiert, wobei die Funktionen

$$\begin{aligned} g_1(p) &= 0.001 \\ g_2(p) &= 0.001 + 0.999 \exp\left(\frac{-1}{p}\right) \end{aligned}$$

empirisch bestimmt und aus der Veröffentlichung von Weickert entnommen sind.

Die Diffusionsrichtung wird bei inpainting_mcf also alleine durch die lokale Richtung des Grauwertgradienten bestimmt, wohingegen G_{CED} die makroskopische Struktur der Bildobjekte auf der Skala Rho wiedergibt und die Stärke der Diffusion in inpainting_ced davon abhängt, wie gut diese Struktur definiert ist.

Um bestmögliche Übereinstimmung der erzeugten Kanten mit den Bilddaten der Umgebung zu erreichen, werden nicht wie bei Filtern üblich die Grauwerte am Rand des Rechengebietes Region gespiegelt, um die Glättung mit der Filtermaske zum Wert Rho auf den randnahen Pixeln durchzuführen. Stattdessen wird die Existenz von Grauwerten auf einem $ceil(3.1 * Rho) + 2$ Pixel breiten Streifen um Region vorausgesetzt und diese Werte bei der Anwendung der Faltungsmaske mitverwendet. Das bedeutet, Region muss mindestens diesen Abstand zum Rand der Bildmatrix Image einhalten. Durch die Einbeziehung der Grauwerte und Richtungsinformationen in diesem erweiterten Gebiet wird erreicht, dass die Fortsetzung der Kanten nicht nur stetig, sondern auch glatt, das heißt ohne Knicke, erfolgt. Es ist zu beachten, dass der Inpaintingvorgang sich auf diejenigen Pixel beschränkt, die innerhalb der ROI des Eingabebildes Image liegen. Umfasst diese nicht die gesamte Region Region, so wird ein Streifen um die Schnittmenge aus Region und der ROI verwendet, um die Randwerte zu definieren.

Um die Anzahl der notwendigen Iterationen bis zum Erreichen eines zufriedenstellenden Zustandes zu reduzieren, kann es sinnvoll sein, die Grauwertmatrix in Region vor Aufruf von inpainting_ced durch den Operator harmonic_interpolation mit der harmonischen Interpolante, einer stetigen Funktion mit minimaler Krümmung, vorzuinitialisieren.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **Region** (input_object) ..... region ⇝ *object*
  Inpaintingregion.
- ▷ **InpaintedImage** (output_object) ..... image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **Sigma** (input_control) ..... real ⇝ *real*
  Glättungsparameter für Ableitungsoperator.
  Defaultwert: 0.5
  Wertevorschläge: Sigma ∈ {0.0, 0.1, 0.5, 1.0}
  Restriktion: Sigma >= 0

▷ **Rho** (input_control) ..... real ⇝ *real*
Glättungsparameter für Diffusionskoeffizienten.
Defaultwert: 3.0
Wertevorschläge: Rho ∈ {0.0, 1.0, 3.0, 5.0, 10.0, 30.0}
Restriktion: Rho >= 0

▷ **Theta** (input_control) ..... real ⇝ *real*
Zeitschritt.
Defaultwert: 0.5
Wertevorschläge: Theta ∈ {0.1, 0.2, 0.3, 0.4, 0.5}
Restriktion: 0 < Theta <= 0.5

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Anzahl Iterationen.
Defaultwert: 10
Wertevorschläge: Iterations ∈ {1, 5, 10, 20, 50, 100, 500}
Restriktion: Iterations >= 1

Beispiel

```
read_image (Image, 'fabrik')
gen_rectangle1 (Rectangle, 270, 180, 320, 230)
harmonic_interpolation (Image, Rectangle, InpaintedImage, 0.01)
inpainting_ced (InpaintedImage, Rectangle, InpaintedImage2, \
                0.5, 3.0, 0.5, 1000)
dev_display(InpaintedImage2)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Alternativen

harmonic_interpolation, inpainting_ct, inpainting_aniso, inpainting_mcf, inpainting_texture

Literatur

J. Weickert, V. Hlavac, R. Sara; „Multiscale texture enhancement“; Computer analysis of images and patterns, Lecture Notes in Computer Science, Vol. 970, pp. 230-237; Springer, Berlin; 1995.

J. Weickert, B. ter Haar Romeny, L. Florack, J. Koenderink, M. Viergever; „A review of nonlinear diffusion filtering“; Scale-Space Theory in Computer Vision, Lecture Notes in Comp. Science, Vol. 1252, pp. 3-28; Springer, Berlin; 1997.

Modul

Foundation

inpainting_ct (Image, Region : InpaintedImage : Epsilon, Kappa, Sigma, Rho, ChannelCoefficients :)

Inpainting eines Bildbereiches mittels Kohärenztransport.

Der Operator inpainting_ct führt ein Inpainting auf der Region Region im Bild Image durch, indem Bildinformation vom Rand dieser Region entlang der Kohärenzrichtung in das Innere transportiert wird.

Das grundlegende Konzept dieses Operators ist Inpainting durch Fortsetzten unterbrochener Höhenlinien. Somit müssen der Bildinhalt und die Inpaintingregion so beschaffen sein, dass diese Idee anwendbar ist. Das heißt, wenn eine Höhenlinie die Inpaintingregion beim Pixel p trifft, dann sollte es einen gegenüberliegenden Punkt q geben, an dem diese Höhenlinie weiterläuft so, dass die Fortsetzung Höhenlinie von zwei gegenüberliegenden Seiten funktioniert. In Fällen, in denen das Bild wenig Geometrie besitzt, kann ein diffusionsbasierter Inpainter, z.B., harmonic_interpolation bessere Ergebnisse liefern. Alternativ kann Kappa auf *0* gesetzt werden. Ein

Extremfall mit geringen globalen Geometrien sind reine Texturen. Dann wird die Idee hinter diesem Operator keine guten Ergenisse produzieren (man denke z.B. an ein Schachbrettmuster mit einer großen Inpaintingregion relativ zu den Schachfeldern). Für diesen Typ Bild ist ein texturbasiertes Inpaiting, wie z.B. `inpainting_texture`, zu empfehlen.

Der Operator verwendet ein sogenanntes Upwind-Schema, um den fehlenden Pixeln Grauwerte zuzuweisen, d.h.,:

- Die Reihenfolge der zu bearbeitenden Pixel ist durch deren euklidischen Abstand zum Rand der Inpaintingregion gegeben.
- Ein neuer Wert u_i wird als ein gewichtetes Mittel von bereits bekannten Werten u_j, die innerhalb einer Kreisscheibe vom Radius `Epsilon` um das aktuelle Pixel liegen, berechnet. Diese Kreisscheibe ist beschränkt auf bereits bekannte Pixel.
- Die Maskengröße dieses Schemas hängt von `Epsilon` ab.

Die anfänglich verwendeten Bilddaten stammen aus einem Streifen der Dicke `Epsilon` um die Inpaintingregion. Damit das Schema funktioniert, muss `Epsilon` mindestens 1 sein, sollte aber größer gewählt werden. Der maximale Wert für `Epsilon` hängt von den Grauwerten, die die in die Region transportiert werden sollen, ab. Eine Wahl von `Epsilon` $= 5$ ist in vielen Fällen anwendbar.

Da das Ziel ist, unterbrochene Höhenlinien zu schließen, muss deren Richtung geschätzt und als Gewicht verwendet werden. Die geschätzte Richtung heißt Kohärenzrichtung, und wird mit Hilfe des Strukturtensors S berechnet.

$$S = G_\rho * DvDv^T$$

und

$$v = G_\sigma * u$$

wobei $*$ die Faltung, u das Grauwertbild , D die Ableitung und G Gaußkerne mit Standardabweichungen σ bzw. ρ bezeichnen. Diese Standardabweichungen werden in den Parametern `Sigma` und `Rho` des Operators übergeben. `Sigma` sollte von der Größenordnung des Rauschens oder unwichtiger kleiner Objekte sein, welche dann durch durch das Vorglätten im Schätzschritt nicht mehr berücksichtigt werden. `Rho` legt die Größe des Fensters fest, welches um einen Pixel zur Richtungsschätzung gelegt wird. Die Kohärenzrichtung c ist die Eigenrichtung von S zum minimalen Eigenwert λ , d.h.

$$Sc = \lambda c, |c| = 1$$

Zur Bearbeitung von Mehrkanal- oder Farbbildern wird das obige Schema auf jeden Kanal separat angewendet. Die Gewichte aber müssen für alle Kanäle gleich sein, um die Information in allen Kanälen in die gleiche Richtung zu propagieren. Wegen der Abhängigkeit der Gewichte von der Kohärenzrichtung wird hier die gemeinsame Richtung als Eigenrichtung eines zusammengesetzten Strukturtensors berechnet. Wenn nun $u_1, ..., u_n$ die n Kanäle des Bildes bezeichnen, so werden die Kanalstrukturtensoren $S_1, ..., S_n$ berechnet und wie folgt zum zusammengesetzten Strukturtensor S kombiniert.

$$S = \sum_{i=1}^{n} a_i S_i$$

Die Koeffizienten a_i werden im Parameter `ChannelCoefficients` übergeben. `ChannelCoefficients` ist dabei ein Tupel der Länge n oder der Länge 1. Falls die Tupellänge 1 ist, wird das arithmetische Mittel verwendet, d.h. $a_i = 1/n$. Ist die Tupellänge genau die Anzahl der Kanäle, so werden die Koeffizienten a_i auf

$$a_i = \frac{\texttt{ChannelCoefficients}_i}{\sum_{i=1}^{n} \texttt{ChannelCoefficients}_i}$$

gesetzt, um eine gültige Konvexkombination zu erhalten. Deswegen müssen die `ChannelCoefficients` selbst größer gleich Null sein und deren Summe größer als Null. Sollte die Tupellänge weder 1 noch gleich der Anzahl der Kanäle sein, oder die Voraussetzung an die Koeffizienten verletzt, so gibt der Operator eine Fehlermeldung aus.

Der Zweck, andere ChannelCoefficients als das arithmetische Mittel zu benutzen, besteht darin, verschiedene Farbcodes zu adaptieren. Die Kohärenzrichtung ist eine geometrische Information des zusammengesetzten Bildes, welche durch starke Kontraste, wie z.B. Kanten, gegeben ist. Je kontrastreicher ein Kanal ist, desto mehr geometrische Information enthält er. Folglich sollte dessen Koeffizient, relativ zu den anderen, größer gewählt werden. Für RGB-Bilder ist *[0.299, 0.587, 0.114]* eine gute Wahl.

Das Gewicht in obigem Schema ist das Produkt einer Richtungskomponente und einer Distanzkomponente. Bezeichne p den 2D-Koordinatenvektor des aktuell zu bearbeitenden Pixels und q den 2D-Koordinatenvektor eines Pixels in der Nachbarschaft (die Kreisscheibe beschränkt auf bereits bekannte Pixel). Dann misst die Richtungskomponente die Abweichung des Differenzvektors $p - q$ von der Kohärenzrichtung. Falls die Abweichung, exponentiell skaliert mit β, groß ist, wird eine kleine Richtungskomponente zugewiesen. Ansonsten wird eine große Richtungskomponente zugewiesen. β wird gesteuert durch Kappa (in Prozent):

$$\beta = 20 * \texttt{Epsilon} * \texttt{Kappa}/100$$

Kappa definiert, wie wichtig es ist, Information entlang der Kohärenzrichtung zu propagieren, d.h. ein großer Wert für Kappa bewirkt scharfe Kanten, während ein kleines Kappa mehr Diffusion erlaubt.

Ein Sonderfall ist Kappa gleich Null: In dieser Situation ist die Richtungskomponente konstant (1). Die Richtungsschätzung wird dabei übersprungen, um Laufzeit zu sparen, und die Parameter Sigma, Rho, ChannelCoefficients werden in diesem Fall bedeutungslos, d.h. die Propagation von Information basiert nicht mehr auf den im Bild sichtbaren Strukturen.

Die Distanzkomponente ist $1/|p - q|$. Folglich wird, wenn q weit entfernt von p liegt, eine kleine Distanzkomponente zugewiesen, andernfalls, wenn q nahe bei p liegt, eine große Distanzkomponente.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **Region** (input_object) . . . . . region ⇝ *object*
  Inpaintingregion.
- ▷ **InpaintedImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **Epsilon** (input_control) . . . . . number ⇝ *real*
  Radius der Pixelnachbarschaft.
  Defaultwert: 5.0
  Typischer Wertebereich: $1.0 \leq$ Epsilon ≤ 20.0
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0
- ▷ **Kappa** (input_control) . . . . . number ⇝ *real*
  Schärfeparameter in Prozent.
  Defaultwert: 25.0
  Typischer Wertebereich: $0.0 \leq$ Kappa ≤ 100.0
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0
- ▷ **Sigma** (input_control) . . . . . number ⇝ *real*
  Vorglättungsparmeter.
  Defaultwert: 1.41
  Typischer Wertebereich: $0.0 \leq$ Sigma ≤ 20.0
  Minimale Schrittweite: 0.001
  Empfohlene Schrittweite: 0.01
- ▷ **Rho** (input_control) . . . . . number ⇝ *real*
  Glättungsparameter für die Richtungsschätzung.
  Defaultwert: 4.0
  Typischer Wertebereich: $0.001 \leq$ Rho ≤ 20.0
  Minimale Schrittweite: 0.001
  Empfohlene Schrittweite: 0.01

▷ **ChannelCoefficients** (input_control) . . . . . number(-array) ⇝ *real*
Kanalgewichte.
Defaultwert: 1

Beispiel

```
read_image (Image, 'claudia')
gen_circle (Circle, 333, 164, 35)
inpainting_ct (Image, Circle, InpaintedImage, 15, 25, 1.5, 3,1.0)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Alternativen

harmonic_interpolation, inpainting_aniso, inpainting_mcf, inpainting_ced, inpainting_texture

Literatur

Folkmar Bornemann, Tom März: „Fast Image Inpainting Based On Coherence Transport“; Journal of Mathematical Imaging and Vision; vol. 28, no. 3; pp. 259-278; 2007.

Modul

Foundation

inpainting_mcf (Image, Region : InpaintedImage : Sigma, Theta, Iterations :)

Inpainting eines Bildbereiches durch Glättung der Höhenlinien.

Der Operator inpainting_mcf setzt die an die Bildregion Region des Eingabebildes Image angrenzenden Bildkanten in Region hinein fort und verbindet sie durch Glättung der Niveaulinien der Grauwertfunktion von Image miteinander.

Dies geschieht durch Anwendung der Krümmungsfluss- oder intrinsischen Wärmeleitungsgleichung

$$u_t = div(\frac{\nabla u}{|\nabla u|})|\nabla u| = curv(u)|\nabla u|$$

auf die durch das Eingabebild Image zu einem Zeitpunkt $t_0 = 0$ definierte Grauwertfunktion u in der Region Region. Die diskretisierte Gleichung wird in Iterations Zeitschritten der Länge Theta gelöst, so dass das Ausgabebild InpaintedImage die Grauwertfunktion zur Zeit Iterations · Theta enthält.

Ein stationärer Zustand der Krümmungsflussgleichung, die auch die Grundlage des Operators mean_curvature_flow bildet, zeichnet sich dadurch aus, dass alle Niveaulinien von u die Krümmung *0* besitzen. Dies bedeutet, dass nach hinreichend vielen Iterationsschritten im Rechengebiet des Ergebnisbildes InpaintedImage nur noch gerade Kanten enthalten sind. Dadurch kann die Struktur der Objekte innerhalb von Region vereinfacht werden, wobei die verbleibenden Kanten stetig an die Kanten der umgebenden Bildmatrix anschließen. So wird eine Entfernung von Fehlstellen oder Störobjekten im Eingabebild, ein sogenanntes Inpainting, erreicht, ohne dass dies für einen menschlichen Beobachter deutlich in Erscheinung tritt, da keine auffälligen Artefakte oder Schmiereffekte zurückbleiben.

Um die Kantenrichtungen insbesondere bei verrauschten Eingabedaten stabiler zu bestimmen, kann der Berechnung der Grauwertgradienten ein zusätzlicher isotroper Glättungsschritt vorgeschaltet werden. Der Parameter Sigma bestimmt die Stärke dieser Glättung als Standardabweichung des zugehörigen Gauß-Faltungskernes, wie er auch im Operator isotropic_diffusion zur isotropen Glättung eines Bildes verwendet wird.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **Region** (input_object) . . . . . region ⇝ *object*
  Inpaintingregion.
- ▷ **InpaintedImage** (output_object) . . . . . image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **Sigma** (input_control) . . . . . real ⇝ *real*
  Glättungsparameter für Ableitungsoperator.
  Defaultwert: 0.5
  Wertevorschläge: `Sigma` ∈ {0.0, 0.1, 0.5, 1.0}
  Restriktion: Sigma >= 0
- ▷ **Theta** (input_control) . . . . . real ⇝ *real*
  Zeitschritt.
  Defaultwert: 0.5
  Wertevorschläge: `Theta` ∈ {0.1, 0.2, 0.3, 0.4, 0.5}
  Restriktion: 0 < Theta <= 0.5
- ▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
  Anzahl Iterationen.
  Defaultwert: 10
  Wertevorschläge: `Iterations` ∈ {1, 5, 10, 20, 50, 100, 500}
  Restriktion: Iterations >= 1

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Alternativen

`harmonic_interpolation`, `inpainting_ct`, `inpainting_aniso`, `inpainting_ced`, `inpainting_texture`

Literatur

M. G. Crandall, P. Lions; „Convergent Difference Schemes for Nonlinear Parabolic Equations and Mean Curvature Motion“; Numer. Math. 75 pp. 17-41; 1996.

G. Aubert, P. Kornprobst; „Mathematical Problems in Image Processing“; Applied Mathematical Sciences 147; Springer, New York; 2002.

Modul

Foundation

inpainting_texture (Image, Region : InpaintedImage : MaskSize, SearchSize, Anisotropy, PostIteration, Smoothness :)

Inpainting eines Bildbereiches durch Texturfortsetzung.

Der Operator `inpainting_texture` dient zur Entfernung größerer Objekte oder Bildstörungen aus der Region `Region` des Eingabebildes `Image`. Dazu werden Bildblöcke der Kantenlänge `MaskSize` aus dem intakten Teil des Bildes an den Rand des Rechengebietes angefügt, bis dieses vollständig mit neuen Grauwerten gefüllt ist. Dieser Vorgang wird als Image Inpainting bezeichnet. Das Rechengebiet heißt daher auch Inpaintinggebiet, und wird ausgehend von `Region` mit jedem eingefügten Rechteck verkleinert. Der Mittelpunkt des aktuell zu bearbeitenden Blockes befinde sich an einer Stelle x. Da x immer auf dem Rand des Inpaintinggebietes gewählt wird, überlappt dieser aktuelle Block mit den bekannten oder bereits eingefügten Grauwerten. Welcher andere Bildblock an die Stelle x passt, wird über seine Grauwertkorrelation mit den Grauwerten im Überlappungsbereich des aktuellen Blocks bestimmt. Als Korrelation wird die Summe der Quadrate der Grauwertdifferenzen verwendet.

Die Bildblöcke, die für die Korrelation herangezogen werden und als Kandidaten für die Datenquelle des nächsten Inpaintingschrittes dienen, bezeichnen wir als Vergleichsblöcke. Der Suchbereich für geeignete Grauwertmuster ist auf ein Quadrat der Kantenlänge 2· `SearchSize` um den Punkt x beschränkt, in dem sich die Mittelpunkte der Vergleichsblöcke befinden müssen.

Die Reihenfolge, in der die Pixel von `Region` aufgefüllt werden, hängt zum einen von der Größe des Überlappungsbereiches und damit von der Anzahl der für die Grauwertkorrelation zur Verfügung stehenden Pixel und zum anderen vom Betrag der Ableitung der Grauwertfunktion tangential zum Rand des Rechengebietes ab. Je größer der Wert des Parameters `Anisotropy` ist, desto stärker werden Punkte, an denen die Ableitung groß ist, bevorzugt. Dadurch wird z.B. erreicht, dass gerade Kanten, die durch den hohen Gradienten repräsentiert werden, durch das gesamte Rechengebiet hindurch fortgesetzt werden können, ohne bei kleiner werdendem Inpaintinggebiet durch das Inpainting von Bildstrukturen von anderen Teilen des Randes unterbrochen zu werden. Auf der anderen Seite führt ein größer Wert von `Anisotropy` aber auch dazu, dass eventuelle Scheinkanten, also während des Inpaintings zufällig entstandene unerwünschte Strukturen, ebenfalls propagiert und Störungen so vergrößert werden können.

Um die Bildung solcher Artefakte einzudämmen, kann der ursprüngliche Algorithmus um eine Nachiteration erweitert werden, bei der glatte oder unscheinbare Bildbereiche als Datenquellen für das Inpainting bevorzugt werden. Setzt man den Parameter `PostIteration` auf den Wert *'min_grad'*, so wird die Summe der Quadrate der Grauwertgradienten über die Vergleichsblöcke minimiert, setzt man ihn auf *'min_range_extension'*, so minimiert man das Anwachsen des Grauwertintervalls der Vergleichsblöcke gegenüber dem Referenzrechteck um den Punkt x. Hat `PostIteration` den Wert *'none'*, wird keine Nachiteration durchgeführt. Welche Blöcke für die nachträgliche Minimierung zulässig sind, wird durch den Parameter `Smoothness` festgelegt. Dieser bestimmt, wie viel größer die mittlere absolute Abweichung der Grauwerte in den zulässigen Vergleichsblöcken gegenüber der vom Ursprungsalgorithmus bestimmten minimalen Abweichung ausfallen darf. Mit zunehmendem Wert von `Smoothness` wird das Inpaintingergebnis daher glatter und Strukturen verschwinden. Die Passgenauigkeit der gewählten Vergleichsblöcke nimmt dabei ab. Wird `Smoothness` auf *0* gesetzt, so berücksichtigt die Nachiteration nur Vergleichsblöcke mit gleich starker Korrelation zum Referenzblock.

Kann das Inpainting nicht vollständig durchgeführt werden, weil Punkte x existieren, für die kein vollständiger Block mit intakten Grauwertinformationen im Suchbereich der Größe `SearchSize` vorhanden ist, so behalten die verbleibenden Pixel ihren ursprünglichen Grauwert und die ROI des Ausgabebildes `InpaintedImage` wird um den unbearbeiteten Bereich reduziert. Ist die Strukturgröße der ROI von `Image` oder des Rechengebietes `Region` kleiner als `MaskSize`, so kann die Laufzeit des Algorithmus extrem anwachsen. Es sollten daher nur klar strukturierte Eingaberegionen verwendet werden.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **`Region`** (input_object) . . . . . region ⇝ *object*
  Inpaintingregion.
- ▷ **`InpaintedImage`** (output_object) . . . . . image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **`MaskSize`** (input_control) . . . . . integer ⇝ *integer*
  Größe der Inpaintingblöcke.
  Defaultwert: 9
  Wertevorschläge: `MaskSize` ∈ {7, 9, 11, 15, 21}
  Restriktion: MaskSize >= 3 && odd(MaskSize)
- ▷ **`SearchSize`** (input_control) . . . . . integer ⇝ *integer*
  Größe des Suchfensters.
  Defaultwert: 30
  Wertevorschläge: `SearchSize` ∈ {15, 30, 50, 100, 1000}
  Restriktion: 2 * SearchSize > MaskSize
- ▷ **`Anisotropy`** (input_control) . . . . . real ⇝ *real*
  Einfluss der Kantenstärke auf die Inpaintingreihenfolge.
  Defaultwert: 1.0
  Wertevorschläge: `Anisotropy` ∈ {0.0, 0.01, 0.1, 0.5, 1.0, 10.0}
  Restriktion: Anisotropy >= 0

▷ **PostIteration** (input_control) ..... string ⇝ *string*
Nachiteration zur Artefaktreduzierung.
Defaultwert: 'none'
Werteliste: `PostIteration` ∈ {'none', 'min_grad', 'min_range_extension'}

▷ **Smoothness** (input_control) ..... real ⇝ *real*
Grauwerttoleranz für Nachiteration.
Defaultwert: 1.0
Wertevorschläge: `Smoothness` ∈ {0.0, 0.1, 0.2, 0.5, 1.0}
Restriktion: Smoothness >= 0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Modul

Foundation

10.8 Kanten

close_edges (Edges, EdgeImage : RegionResult : MinAmplitude :)

Schließen von Kanten gemäß Amplitudenbild.

`close_edges` schließt die von einem Kantendetektor gelieferten Kanten zu Konturen. Dabei wird iterativ von jedem Kantenpunkt aus nach den beiden Nachbarpunkten mit stärkster Amplitude (i.e. größtem Gradienten) gesucht und diese beiden Punkte zur Kantenregion hinzugenommen, falls ihre Amplituden größer oder gleich der übergebenen Mindestamplitude sind. Als Eingabe erwartet die Routine also neben den Kanten (`Edges`) auch ein Amplitudenbild (`EdgeImage`), wie es beispielsweise die Routinen `edges_image` oder `sobel_amp` als Ergebnis liefern. Nicht berücksichtigt wird hingegen die Richtung der Konturen. Sie können daher in Gebieten mit Plateaus im Amplitudenbild zu „schwingen“ beginnen.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Edges** (input_object) ..... region(-array) ⇝ *object*
Region mit ein Pixel breiten Kanten.

▷ **EdgeImage** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / int4
Kanten-Amplitudenbild.

▷ **RegionResult** (output_object) ..... region(-array) ⇝ *object*
Ausgaberegionen (geschlossene Kanten).

▷ **MinAmplitude** (input_control) ..... integer ⇝ *integer*
Mindestamplitude für Kanten.
Defaultwert: 16
Wertevorschläge: `MinAmplitude` ∈ {5, 8, 10, 12, 16, 20, 25, 30, 40, 50}
Typischer Wertebereich: 1 ≤ `MinAmplitude` ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: MinAmplitude >= 0

Beispiel

```
sobel_amp(Image,&EdgeAmp,"sum_abs",5);
threshold(EdgeAmp,&EdgeRegion,40.0,255.0);
skeleton(EdgeRegion,&ThinEdge);
close_edges(ThinEdge,EdgeAmp,&CloseEdges,15);
skeleton(CloseEdges,&ThinCloseEdges);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert close_edges den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

edges_image, sobel_amp, threshold, skeleton

Nachfolger

skeleton

Alternativen

close_edges_length, dilation1, closing

Siehe auch

gray_skeleton

Modul

Foundation

close_edges_length (Edges, Gradient : ClosedEdges : MinAmplitude, MaxGapLength :)

Schließen von Kanten gemäß Amplitudenbild.

close_edges_length schließt die von einem Kantendetektor gelieferten Kanten zu Konturen. Als Eingabe erwartet die Routine also neben den Kanten (Edges) auch ein Amplitudenbild (Gradient), wie es beispielsweise die Routinen edges_image oder sobel_amp als Ergebnis liefern.

Der Konturschluss erfolgt in zwei Schritten: Zunächst werden ein Pixel große Lücken in den Eingabekonturstücken geschlossen und isolierte Kantenpunkte eliminiert. Im zweiten Schritt werden dann offene Konturen um maximal MaxGapLength verlängert, bis entweder die Kontur geschlossen ist oder kein Fortsetzungspunkt mit ausreichend signifikantem Gradienten mehr gefunden werden kann. Ein Gradientenwert gilt dabei als signifikant, wenn er größer oder gleich MinAmplitude ist. Kandidaten für den nächsten Fortsetzungspunkt sind der Nachbarpunkt in Richtung der Kontur (gemäß Gradient im aktuellen Endpunkt) sowie die beiden angrenzenden Nachbarpunkte (8er Nachbarschaft). Für jeden dieser drei potentiellen Fortsetzungspunkte wird die Summe aus dem eigenen Gradienten und dem maximalen Gradienten der nächsten drei möglichen Fortsetzungspunkte gebildet (*look ahead* der Länge 1). Der Punkt mit maximaler Summe wird dann als Fortsetzungspunkt ausgewählt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Edges** (input_object) ..... region(-array) ⇝ *object*
Region mit ein Pixel breiten Kanten.

▷ **Gradient** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
Kanten-Amplitudenbild.

▷ **ClosedEdges** (output_object) ..... region(-array) ⇝ *object*
Ausgaberegion (geschlossene Kanten).

▷ **MinAmplitude** (input_control) ..... integer ⇝ *integer*
Mindestamplitude für Kanten.
Defaultwert: 16
Wertevorschläge: MinAmplitude ∈ {5, 8, 10, 12, 16, 20, 25, 30, 40, 50}
Typischer Wertebereich: 1 ≤ MinAmplitude ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: MinAmplitude >= 0

▷ **MaxGapLength** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl von Punkten, um die die Kanten verlängert werden.
Defaultwert: 3
Wertevorschläge: MaxGapLength ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50, 70, 100}
Typischer Wertebereich: 1 ≤ MaxGapLength ≤ 127
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: MaxGapLength > 0 && MaxGapLength <= 127

Beispiel

```
sobel_amp(Image,&EdgeAmp,"sum_abs",5);
threshold(EdgeAmp,&EdgeRegion,40.0,255.0);
skeleton(EdgeRegion,&ThinEdge);
close_edges_length(ThinEdge,EdgeAmp,&CloseEdges,15,3);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert close_edges_length den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

edges_image, sobel_amp, threshold, skeleton

Alternativen

close_edges, dilation1, closing

Literatur

M. Üsbeck: „Untersuchungen zur echtzeitfähigen Segmentierung“; Studienarbeit, Bayerisches Forschungszentrum für Wissensbasierte Systeme (FORWISS), Erlangen, 1993.

Modul

Foundation

derivate_gauss (Image : DerivGauss : Sigma, Component :)

Ableitungen der Gaußfunktion.

derivate_gauss berechnet verschiedene Ableitungen der Gaußfunktion und daraus abgeleiteter Größen. Dabei ist Sigma der Parameter der Gaußfunktion (d.h. die Glättung). Falls in Sigma ein Wert angegeben wird, ist die Glättung in Spalten- und Zeilenrichtung gleich. Falls zwei Werte in Sigma übergeben werden, wird mit dem ersten Wert die Glättung in Spaltenrichtung und mit dem zweiten Wert die Glättung in Zeilenrichtung festgelegt. Mögliche Werte für Component sind:

'none' Nur Glättung.

'x' 1. Ableitung nach x.

$$g'(x,y) = \frac{\partial g(x,y)}{\partial x}$$

'y' 1. Ableitung nach y.

$$g'(x,y) = \frac{\partial g(x,y)}{\partial y}$$

'gradient' Betrag des Gradienten.

$$g'(x,y) = \sqrt{\frac{\partial g(x,y)^2}{\partial x}\frac{\partial g(x,y)^2}{\partial y}}$$

'gradient_dir' Richtung des Gradienten im Bogenmaß.

$$\phi = \text{atan2}(\frac{\partial g(x,y)}{\partial y}, \frac{\partial g(x,y)}{\partial x})$$

'xx' 2. Ableitung nach x.

$$g'(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2}$$

'yy' 2. Ableitung nach y.

$$g'(x,y) = \frac{\partial^2 g(x,y)}{\partial y^2}$$

'xy' Ableitung nach x und y.

$$g'(x,y) = \frac{\partial^2 g(x,y)}{\partial x \partial y}$$

'xxx' 3. Ableitung nach x.

$$g'(x,y) = \frac{\partial^3 g(x,y)}{\partial x^3}$$

'yyy' 3. Ableitung nach y.

$$g'(x,y) = \frac{\partial^3 g(x,y)}{\partial y^3}$$

'xxy' Ableitung nach x, x und y.

$$g'(x,y) = \frac{\partial^3 g(x,y)}{\partial x^2 \partial y}$$

'xyy' Ableitung nach x, y und y.

$$g'(x,y) = \frac{\partial^3 g(x,y)}{\partial x \partial y^2}$$

'det' Determinante der Hessematrix:

$$DET = \frac{\partial^2 g(x,y)}{\partial x^2}\frac{\partial^2 g(x,y)}{\partial y^2} - \left(\frac{\partial^2 g(x,y)}{\partial y \partial x}\right)^2$$

'laplace' Laplace-Operator (Spur der Hessematrix)

$$TR = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}$$

'mean_curvature' Mittlere Krümmung H

$$\begin{aligned}
a &= (1 + \frac{\partial g(x,y)^2}{\partial x})\frac{\partial^2 g(x,y)}{\partial y^2} \\
b &= 2\frac{\partial g(x,y)}{\partial x}\frac{\partial g(x,y)}{\partial y}\frac{\partial^2 g(x,y)}{\partial y \partial x} \\
c &= (1 + \frac{\partial g(x,y)^2}{\partial y})\frac{\partial^2 g(x,y)}{\partial x^2} \\
d &= (1 + \frac{\partial g(x,y)^2}{\partial x} + \frac{\partial g(x,y)^2}{\partial y})^{\frac{3}{2}} \\
H &= \frac{a - b + c}{d} \\
H &= \frac{1}{2}(\kappa_{min} + \kappa_{max})
\end{aligned}$$

'gauss_curvature' Gaußkrümmung K

$$K = \frac{DET}{(1 + \frac{\partial g(x,y)^2}{\partial x} + \frac{\partial g(x,y)^2}{\partial y})^2}$$

'area' Differentielle Fläche A

$$\begin{aligned} A &= EG - F^2 \\ E &= 1 + \frac{\partial g(x,y)}{\partial x}^2 \\ F &= \frac{\partial g(x,y)}{\partial x}\frac{\partial g(x,y)}{\partial y} \\ G &= 1 + \frac{\partial g(x,y)}{\partial y}^2 \end{aligned}$$

'eigenvalue1' Erster Eigenwert

$$\begin{aligned} a &= \frac{\frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}}{2} \\ \lambda_1 &= a + \sqrt{a^2 - (\frac{\partial^2 g(x,y)}{\partial x^2}\frac{\partial^2 g(x,y)}{\partial y^2} - \frac{\partial^2 g(x,y)^2}{\partial y \partial x})} \end{aligned}$$

'eigenvalue2' Zweiter Eigenwert

$$\begin{aligned} a &= \frac{\frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}}{2} \\ \lambda_2 &= a - \sqrt{a^2 - (\frac{\partial^2 g(x,y)}{\partial x^2}\frac{\partial^2 g(x,y)}{\partial y^2} - \frac{\partial^2 g(x,y)^2}{\partial y \partial x})} \end{aligned}$$

'eigenvec_dir' Richtung des zum ersten Eigenwert gehörigen Eigenvektors im Bogenmaß

'main1_curvature' Erste Hauptkrümmung

$$\kappa_{max} = H + \sqrt{H^2 - K}$$

'main2_curvature' Zweite Hauptkrümmung

$$\kappa_{min} = H - \sqrt{H^2 - K}$$

'kitchen_rosenfeld' 2. Ableitung in Richtung senkrecht zum Gradienten

$$k = \frac{\frac{\partial^2 g(x,y)}{\partial x^2}\frac{\partial g(x,y)^2}{\partial y} + \frac{\partial^2 g(x,y)}{\partial y^2}\frac{\partial g(x,y)^2}{\partial x} - 2\frac{\partial^2 g(x,y)}{\partial y \partial x}\frac{\partial g(x,y)^2}{\partial x}\frac{\partial g(x,y)^2}{\partial y}}{\frac{\partial g(x,y)^2}{\partial x} + \frac{\partial g(x,y)^2}{\partial y}}$$

'zuniga_haralick' Normierte 2. Ableitung in Richtung senkrecht zum Gradienten

$$k = \frac{\frac{\partial^2 g(x,y)}{\partial x^2}\frac{\partial g(x,y)^2}{\partial y} + \frac{\partial^2 g(x,y)}{\partial y^2}\frac{\partial g(x,y)^2}{\partial x} - 2\frac{\partial^2 g(x,y)}{\partial y \partial x}\frac{\partial g(x,y)^2}{\partial x}\frac{\partial g(x,y)^2}{\partial y}}{\left(\frac{\partial g(x,y)^2}{\partial x} + \frac{\partial g(x,y)^2}{\partial y}\right)^{\frac{3}{2}}}$$

'2nd_ddg' 2. Ableitung in Richtung des Gradienten

$$k = \frac{\frac{\partial^2 g(x,y)}{\partial x^2}\frac{\partial g(x,y)^2}{\partial x} + 2\frac{\partial g(x,y)}{\partial x}\frac{\partial g(x,y)}{\partial y}\frac{\partial^2 g(x,y)}{\partial y \partial x} + \frac{\partial^2 g(x,y)}{\partial y^2}\frac{\partial g(x,y)^2}{\partial y}}{\frac{\partial g(x,y)^2}{\partial x} + \frac{\partial g(x,y)^2}{\partial y}}$$

'de_saint_venant' 2. Ableitung in Richtung des Gradienten und senkrecht dazu

$$k = \frac{\frac{\partial g(x,y)}{\partial x}\frac{\partial g(x,y)}{\partial y}(\frac{\partial^2 g(x,y)}{\partial x^2} - \frac{\partial^2 g(x,y)}{\partial y^2}) - (\frac{\partial g(x,y)^2}{\partial x} - \frac{\partial g(x,y)^2}{\partial y})\frac{\partial^2 g(x,y)}{\partial x \partial y}}{\frac{\partial g(x,y)^2}{\partial x} + \frac{\partial g(x,y)^2}{\partial y}}$$

Achtung

Nebst der reinen C Version gibt es spezifische Implementierungen von `derivate_gauss` zur Beschleunigung. Eine solche Optimierung wird verwendet, wenn diese vom System unterstützt wird und der entsprechende Systemparameter `*_enable` auf *'true'* gesetzt ist, siehe `set_system`. Folgende Optimierungen werden unterstützt (der Priorität nach gelistet):

- *AVX512f* (*'avx512f_enable'*)
- *AVX* (Instruktionen (*'avx_enable'*)
- *SSE2* Instruktionen (*'sse2_enable'*)

Diese Implementierungen sind aufgrund numerischer Probleme im Vergleich zur reinen C Version etwas ungenauer. Mit den SSE2 Instruktionen bewegt sich der Unterschied beispielsweise in einer Größenordnung von $1.0e-5$ für 'byte' Bilder und *'none'*, *'x'* oder *'y'* als `Component`.

Wird Genauigkeit der Geschwindigkeit gegenüber bevorzugt, müssen vor dem Aufruf von `derivate_gauss` alle entsprechenden Systemparameter mit `set_system` auf *'false'* gesetzt werden. Dadurch verwendet `derivate_gauss` keine der Optimierungen. Vergessen Sie nicht den Systemparameter danach wieder auf *'true'* zu setzen.

`derivate_gauss` wird nur dann auf einem OpenCL-Gerät ausgeführt wenn `Sigma` eine Filterbreite bzw. Filterhöhe bis zu 129 impliziert. Das entspricht einem `Sigma` von weniger als 20.7 für `Component` = *'none'*. Außerdem ist die OpenCL Implementierung wie die SSE2 Implementierung aufgrund numerischer Probleme etwas ungenauer als die reine C Version.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Eingabebilder.

▷ **`DerivGauss`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* real
Gefilterte Bilder.

▷ **`Sigma`** (input_control) ..... real(-array) ⇝ *real*
Sigma der Gaußfunktion.
Defaultwert: 1.0
Wertevorschläge: `Sigma` ∈ {0.7, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0}
Typischer Wertebereich: $0.2 \leq$ `Sigma` ≤ 50.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: 0.01 <= Sigma <= 50.0

▷ **`Component`** (input_control) ..... string ⇝ *string*
Zu berechnende Ableitung oder Merkmal.
Defaultwert: 'x'
Werteliste: `Component` ∈ {'none', 'x', 'y', 'gradient', 'xx', 'yy', 'xy', 'xxx', 'yyy', 'xxy', 'xyy', 'det', 'mean_curvature', 'gauss_curvature', 'eigenvalue1', 'eigenvalue2', 'main1_curvature', 'main2_curvature', 'kitchen_rosenfeld', 'zuniga_haralick', '2nd_ddg', 'de_saint_venant', 'area', 'laplace', 'gradient_dir', 'eigenvec_dir'}

Beispiel

```
read_image(&Image,"mreut");
derivate_gauss(Image,&Gauss,3.0,"x");
zero_crossing(Gauss,&ZeroCrossings);
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

zero_crossing, dual_threshold

Alternativen

laplace, laplace_of_gauss, binomial_filter, gauss_filter, smooth_image, isotropic_diffusion

Siehe auch

zero_crossing, dual_threshold

Modul

Foundation

diff_of_gauss (Image : DiffOfGauss : Sigma, SigFactor :)

Näherung für den LoG-Operator (Laplace of Gaussian).

diff_of_gauss ist eine Näherung für den Laplace-Operator. Dieser wird hier als Differenz zweier Gaußfunktionen approximiert. Die Standardabweichungen dieser Gaußfunktionen ergeben sich nach Marr aus dem Parameter Sigma des Laplace-Operators und dem Verhältnis der beiden Standardabweichungen zueinander (SigFactor):

$$sigma1 = \frac{\texttt{Sigma}}{\sqrt{-2\frac{\log(\frac{1}{\texttt{SigFactor}})}{\texttt{SigFactor}^2-1}}}$$

$$sigma2 = \frac{sigma1}{\texttt{SigFactor}}$$

$$\texttt{DiffOfGauss} = (\texttt{Image} * gauss(sigma1)) - (\texttt{Image} * gauss(sigma2))$$

Für SigFactor $= 1.6$ ergibt sich nach Marr eine Näherung an den Mexican-Hat-Operator (LoG, Laplace of Gaussian). Das Ergebnisbild wird in DiffOfGauss abgelegt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Eingabebilder

▷ **DiffOfGauss** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* int2
Gefilterte Ausgabebilder.

▷ **Sigma** (input_control) . . . . . real ⇝ *real*
Glättungsparameter des (zu approximierenden) Laplace-Operators.
Defaultwert: 3.0
Wertevorschläge: Sigma ∈ {2.0, 3.0, 4.0, 5.0}
Typischer Wertebereich: 0.2 ≤ Sigma ≤ 50.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Sigma > 0.0

▷ **SigFactor** (input_control) . . . . . real ⇝ *real*
Verhältnis der Standardabweichungen der eingesetzten Gaußfunktionen (Marr empfiehlt 1.6).
Defaultwert: 1.6
Typischer Wertebereich: 0.1 ≤ SigFactor ≤ 10.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: SigFactor > 0.0

Beispiel

```
read_image(Image,'fabrik')
diff_of_gauss(Image,Laplace,2.0,1.6)
zero_crossing(Laplace,ZeroCrossings)
```

Komplexität

Die Komplexität hängt linear von der Anzahl der Bildpunkte und der Größe von Sigma ab.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `diff_of_gauss` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`zero_crossing`, `dual_threshold`

Alternativen

`laplace`, `derivate_gauss`

Literatur

D. Marr: „Vision (A computational investigation into human representation and processing of visual information)“; New York, W.H. Freeman and Company; 1982.

Modul

Foundation

edges_color (Image : ImaAmp, ImaDir : Filter, Alpha, NMS, Low, High :)

Farbkantendetektion mit Canny-, Deriche- oder Shen-Filtern.

`edges_color` extrahiert Farbkanten aus dem Eingabebild `Image`. Zur Definition von Farbkanten wird das mehrkanalige Bild `Image` als Abbildung $f : R^2 \mapsto R^n$ betrachtet, wobei n der Anzahl der Kanäle in `Image` entspricht. Für solche Funktionen existiert eine natürliche Erweiterung des Gradienten: der metrische Tensor G, welcher es erlaubt, die Größe der Veränderung von f für jede beliebige Richtung, die durch den Richtungsvektor v gegeben ist, zu berechnen. Zur Vereinfachung der Notation wird G als zweidimensionale Matrix betrachtet. Damit kann die Größe der Veränderung von f in Richtung v durch $v^T G v$ berechnet werden. Dabei ist

$$G = \begin{pmatrix} f_x^T f_x & f_x^T f_y \\ f_x^T f_y & f_y^T f_y \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} \frac{\partial f_i}{\partial x}\frac{\partial f_i}{\partial x} & \sum_{i=1}^{n} \frac{\partial f_i}{\partial x}\frac{\partial f_i}{\partial y} \\ \sum_{i=1}^{n} \frac{\partial f_i}{\partial x}\frac{\partial f_i}{\partial y} & \sum_{i=1}^{n} \frac{\partial f_i}{\partial y}\frac{\partial f_i}{\partial y} \end{pmatrix} .$$

Zur Kantenextraktion werden, wie auch bei `edges_image`, die partiellen Ableitungen der Bilder, die im metrischen Tensor berechnet werden müssen, mit den jeweiligen Kantenfiltern berechnet. Für `Filter` = *'canny'* werden die partiellen Ableitungen der Gaußmasken (siehe `derivate_gauss`) verwendet, für `Filter` = *'deriche1'* und `Filter` = *'deriche2'* die entsprechenden Deriche-Filter, für `Filter` = *'shen'* die entsprechenden Shen-Filter und für `Filter` = *'sobel_fast'* der Sobel-Filter. Wie bei einkanaligen Bildern wird die Gradientenrichtung dadurch definiert, dass die Größe der Veränderung von f in der Richtung v maximal ist. Diese Richtung

ist gegeben durch den Eigenvektor von G, der zum größten Eigenwert von G gehört. Die Wurzel aus dem größten Eigenwert ist das Äquivalent des Gradientenbetrags (der Amplitude) bei einkanaligen Bildern. Sie wird in ImaAmp zurückgeliefert. Für einkanalige Bilder sind beide Definitionen äquivalent. Da der Gradientenbetrag sich potentiell nicht im Eingabedatentyp (byte oder uint2) darstellen lässt, wird in ImaAmp der nächstgrößere Datentyp (uint2 oder int4) verwendet. Der Eigenvektor liefert gleichzeitig auch die Kantenrichtung. Diese ist, im Gegensatz zu einkanaligen Bildern, nur modulo 180 Grad definiert. Die Kantenrichtungen werden wie bei edges_image in 2-Grad-Schritten kodiert und in ImaDir zurückgeliefert. In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert. Aus Geschwindigkeitsgründen wird für Filter = *'sobel_fast'* das Richtungsbild nicht explizit berechnet, und kann somit nicht zurückgegeben werden. In diesem Fall ist ImaDir ein leeres Bildobjekt.

Die „Filterbreite“ (d.h. das Einzugsgebiet der Filter) ist für alle Filter außer *'sobel_fast'* (wo die Filtergröße 3×3 ist und Alpha ignoriert wird) frei wählbar und kann mittels info_edges für konkrete Werte des Parameters Alpha abgeschätzt werden. Sie nimmt bei den Deriche- und Shen-Filtern mit Alpha ab bzw. beim Canny-Filter zu (Alpha ist in diesem Fall die Standardabweichung der zugrundeliegenden Gaußfunktion). „Breite“ Filter weisen eine höhere Rauschinvarianz, aber auch ein verringertes Auflösungsvermögen für Bilddetails auf. Nichtrekursive Filter, wie hier die Canny-Filter, werden mittels Filtermasken realisiert. In diesem Fall erhöht sich die Laufzeit mit wachsender Filterbreite. Die Laufzeit der rekursiver Filter ist hingegen konstant. Das Einzugsgebiet der Deriche- und Shen-Filter ist also ohne Mehraufwand beliebig vergrößerbar. Der daraus resultierende Laufzeitvorteil gegenüber dem Canny-Filter nimmt naturgemäß mit wachsender „Filterbreite“ zu. Zur Randbehandlung wird bei den rekursiven Filtern angenommen, das Bildsignal falle außerhalb des Bildes auf Null ab. Beim Canny-Operator wird hingegen das Bild durch Spiegelung der Randpunkte fortgesetzt. Das Signal-Rausch-Verhältnis der Filter ist vergleichbar, wenn Alpha wie folgt gewählt wird:

$$\begin{aligned} \texttt{Alpha}('deriche2') &= \texttt{Alpha}('deriche1')/2 \\ \texttt{Alpha}('shen') &= \texttt{Alpha}('deriche1')/2 \\ \texttt{Alpha}('canny') &= 1.77/\texttt{Alpha}('deriche1') \end{aligned}$$

edges_color bietet (optional) die Weiterverarbeitung der Filterergebnisse mittels Non-Maximum-Suppression (NMS = *'nms'*/*'inms'*/*'hvnms'*, sonst Wert *'none'*) und Hysterese-Schwellenwertoperation (Low,High, falls nicht erwünscht: einer der Werte negativ). Dies entspricht im wesentlichen nachfolgenden Aufrufen der Routinen

```
nonmax_suppression_dir(...,NMS,...)
hysteresis_threshold(...,Low,High,1000,...)
```

Die Hysterese-Schwellenwertoperation wird nicht angewandt, wenn NMS auf *'none'* gesetzt ist.

Für *'sobel_fast'* wird für alle Werte von NMS außer *'none'* dieselbe Non-Maximum-Suppression ausgeführt. Zusätzlich wird für *'sobel_fast'* auch noch eine Verdünnung der Kanten auf ein Pixel Breite durchgeführt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **ImaAmp** (output_object) ..... singlechannelimage(-array) ⇝ *object :* uint2 / int4
  Amplitudenbild.
- ▷ **ImaDir** (output_object) ..... singlechannelimage(-array) ⇝ *object :* direction
  Richtungsbild.
- ▷ **Filter** (input_control) ..... string ⇝ *string*
  Gewünschter Kanten-Operator.
  Defaultwert: 'canny'
  Werteliste: Filter ∈ {'canny', 'deriche1', 'deriche2', 'shen', 'sobel_fast'}
- ▷ **Alpha** (input_control) ..... real ⇝ *real*
  Filterparameter: kleine Werte bewirken starke Glättung, also auch weniger Bilddetails (bei 'canny'

umgekehrt).
Defaultwert: 1.0
Wertevorschläge: `Alpha` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.0, 1.1, 1.2, 1.5, 2.0, 2.5, 3.0}
Typischer Wertebereich: 0.2 ≤ `Alpha` ≤ 50.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Alpha > 0.0

▷ **NMS** (input_control) . . . . . string ⇝ *string*
Non-Maximum-Suppression ('none', falls nicht erwünscht).
Defaultwert: 'nms'
Werteliste: `NMS` ∈ {'nms', 'inms', 'hvnms', 'none'}

▷ **Low** (input_control) . . . . . integer ⇝ *integer*
Untere Schwelle für Hysterese-Schwellenwertoperation (negativ, falls keine Schwellenwertbildung erwünscht).
Defaultwert: 20
Wertevorschläge: `Low` ∈ {5, 10, 15, 20, 25, 30, 40}
Typischer Wertebereich: 1 ≤ Low
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: Low >= 1 || Low < 0

▷ **High** (input_control) . . . . . integer ⇝ *integer*
Obere Schwelle für Hysterese-Schwellenwertoperation (negativ, falls keine Schwellenwertbildung erwünscht).
Defaultwert: 40
Wertevorschläge: `High` ∈ {10, 15, 20, 25, 30, 40, 50, 60, 70}
Typischer Wertebereich: 1 ≤ High
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: High >= 1 || High < 0 && High >= Low

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `edges_color` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`threshold`

Alternativen

`edges_color_sub_pix`

Siehe auch

`edges_image`, `edges_sub_pix`, `info_edges`, `nonmax_suppression_amp`, `hysteresis_threshold`

Literatur

C. Steger: „Subpixel-Precise Extraction of Lines and Edges"; International Archives of Photogrammetry and Remote Sensing, vol. XXXIII, part B3; pp. 141-156; 2000.

C. Steger: „Unbiased Extraction of Curvilinear Structures from 2D and 3D Images"; Herbert Utz Verlag, München; 1998.

S. Di Zenzo: „A Note on the Gradient of a Multi-Image"; Computer Vision, Graphics, and Image Processing, vol. 33; pp. 116-125; 1986.

Aldo Cumani: „Edge Detection in Multispectral Images"; Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing, vol. 53, no. 1; pp. 40-51; 1991.

J.Canny: „Finding Edges and Lines in Images“; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge; 1983.

J.Canny: „A Computational Approach to Edge Detection“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-8, vol. 6; pp. 679-698; 1986.

R.Deriche: „Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector“; International Journal of Computer Vision; vol. 1, no. 2; pp. 167-187; 1987.

R.Deriche: „Fast Algorithms for Low-Level Vision“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-12, no. 1; pp. 78-87; 1990.

J. Shen, S. Castan: „An Optimal Linear Operator for Step Edge Detection“; Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing, vol. 54, no. 2; pp. 112-133; 1992.

Modul

Foundation

edges_color_sub_pix (Image : Edges : Filter, Alpha, Low, High :)

Subpixel-genaue Farbkantendetektion mit Deriche-, Shen- oder Canny-Filtern.

edges_color_sub_pix extrahiert subpixel-genaue Farbkanten aus dem Eingabebild Image. Die Definition von Farbkanten ist in der Beschreibung von edges_color erläutert. Es stehen dieselben Kantenfilter wie bei edges_color zur Verfügung: *'canny'*, *'deriche1'*, *'deriche2'* und *'shen'*. Zusätzlich steht noch ein schneller Sobel-Filter mit *'sobel_fast'* zur Verfügung. Die Filter werden durch den Parameter Filter spezifiziert.

Die „Filterbreite“ (Stärke der Glättung) Alpha ist frei wählbar. Für eine ausführliche Beschreibung dieses Parameters siehe edges_color. Bei Filter = *'sobel_fast'* wird dieser Parameter ignoriert.

Die extrahierten Kanten werden als subpixel-genaue XLD-Konturen in Edges zurückgeliefert. Für alle Kantenoperatoren außer *'sobel_fast'* werden für jeden Kantenpunkt die folgenden Attribute definiert (Für weitere Informationen siehe get_contour_attrib_xld):

'edge_direction': Gibt die Richtung der Kante (nicht der Kontur), berechnet aus den Grauwertgradienten in Spalten- und Zeilenrichtung, im Bogenmaß an. Die Winkel sind bezüglich der Zeilenachse gegen den Uhrzeigersinn gegeben.

'angle': Richtung der Normalenvektoren der Kontur (so orientiert, dass die Normalenvektoren auf die rechte Seite der Kontur in Durchlaufrichtung der Kontur zeigen; die Winkel sind bzgl. der Zeilenachse des Bildes im Bogenmaß angegeben).

'response': Die Amplitude der Kante.

edges_color_sub_pix verknüpft die einzelnen Kantenpunkte mit einem Hysterese-Schwellenwert-artigen Verfahren zu Kanten, wie es auch in edges_sub_pix und lines_gauss verwendet wird. Dabei werden Punkte, deren Amplitude größer als High ist, sofort als sichere Kantenpunkte akzeptiert. Punkte, deren Amplitude kleiner als Low ist, werden sofort verworfen. Alle Punkte, die eine zweite Amplitude zwischen diesen zwei Werten besitzen, werden akzeptiert, wenn sie durch einen Pfad mit sicheren Punkten verbunden sind (siehe lines_gauss und hysteresis_threshold).

Da Kantenextraktoren oftmals bestimmte Kreuzungspunkte nicht extrahieren können, kann durch Anhängen von *'_junctions'* an die oben beschriebenen Werte von Filter versucht werden, diese mit anderen Mitteln zu extrahieren. Dieser Modus ist analog zu dem bei edges_sub_pix und lines_gauss verfügbaren Modus zur Vervollständigung von Kreuzungspunkten.

Der Kantenoperator *'sobel_fast'* hat dieselbe Semantik wie alle anderen Kantenoperatoren, basiert intern aber auf einer stark vereinfachten Variante der einzelnen Schritte (Hysterese-Schwellenwert-Operation, Verknüpfung der Kantenpunkte, Extraktion der Subpixel-Position). Daher können die Ergebnisse von *'sobel_fast'* in manchen Fällen eine geringere Genauigkeit als die Ergebnisse der anderen Filter aufweisen und andere Kantenteile selektieren.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild.

▷ **Edges** (output_object) . . . . . xld_cont-array ⇝ *object*
Extrahierte Kanten.

▷ **Filter** (input_control) . . . . . string ⇝ *string*
Gewünschter Kanten-Operator.
Defaultwert: 'canny'
Werteliste: `Filter` ∈ {'canny', 'deriche1', 'deriche2', 'shen', 'sobel_fast', 'canny_junctions', 'deriche1_junctions', 'deriche2_junctions', 'shen_junctions'}

▷ **Alpha** (input_control) . . . . . real ⇝ *real*
Filterparameter: kleine Werte bewirken starke Glättung, also auch weniger Bilddetails (bei 'canny' umgekehrt).
Defaultwert: 1.0
Wertevorschläge: `Alpha` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.0, 1.1, 1.2, 1.5, 2.0, 2.5, 3.0}
Typischer Wertebereich: 0.7 ≤ `Alpha` ≤ 50.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Alpha > 0.0

▷ **Low** (input_control) . . . . . number ⇝ *real* / integer
Untere Schwelle für Hysterese-Schwellenwertoperation.
Defaultwert: 20
Wertevorschläge: `Low` ∈ {5, 10, 15, 20, 25, 30, 40}
Typischer Wertebereich: 1 ≤ Low
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: Low > 0

▷ **High** (input_control) . . . . . number ⇝ *real* / integer
Obere Schwelle für Hysterese-Schwellenwertoperation.
Defaultwert: 40
Wertevorschläge: `High` ∈ {10, 15, 20, 25, 30, 40, 50, 60, 70}
Typischer Wertebereich: 1 ≤ High
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: High > 0 && High >= Low

Komplexität

Der Bedarf an temporären Speicher ist abhängig von der Höhe H der Region von `Image`.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `edges_color_sub_pix` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

`edges_color`

Siehe auch

`edges_image`, `edges_sub_pix`, `info_edges`, `hysteresis_threshold`, `lines_gauss`, `lines_facet`

Literatur

C. Steger: „Subpixel-Precise Extraction of Lines and Edges“; International Archives of Photogrammetry and Remote Sensing, vol. XXXIII, part B3; pp. 141-156; 2000.

C. Steger: „Unbiased Extraction of Curvilinear Structures from 2D and 3D Images“; Herbert Utz Verlag, München; 1998.

S. Di Zenzo: „A Note on the Gradient of a Multi-Image“; Computer Vision, Graphics, and Image Processing, vol. 33; pp. 116-125; 1986.

Aldo Cumani: „Edge Detection in Multispectral Images“; Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing, vol. 53, no. 1; pp. 40-51; 1991.

J.Canny: „Finding Edges and Lines in Images“; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge; 1983.

J.Canny: „A Computational Approach to Edge Detection“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-8, vol. 6; pp. 679-698; 1986.

R.Deriche: „Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector“; International Journal of Computer Vision; vol. 1, no. 2; pp. 167-187; 1987.

R.Deriche: „Fast Algorithms for Low-Level Vision“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-12, no. 1; pp. 78-87; 1990.

J. Shen, S. Castan: „An Optimal Linear Operator for Step Edge Detection“; Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing, vol. 54, no. 2; pp. 112-133; 1992.

Modul

2D Metrology

edges_image (Image : ImaAmp, ImaDir : Filter, Alpha, NMS, Low, High :)

Kantendetektion mit Deriche-, Lanser-, Shen- oder Canny-Filtern.

edges_image dient der Detektion von Stufenkanten mittels rekursiver Filter (nach Deriche, Lanser und Shen) bzw. mittels des konventionell über Filtermasken realisierten „Derivative of Gaussian“ Filter, den auch Canny verwendet. Zudem existiert eine sehr schnelle Variante des Sobel-Filters. Konkret stehen somit die Kantenoperatoren

'deriche1', *'lanser1'*, *'deriche1_int4'*, *'deriche2'*, *'lanser2'*, *'deriche2_int4'*, *'shen'*, *'mshen'*, *'canny'* und *'sobel_fast'*

zur Verfügung (Parameter Filter).

Die Kantenamplituden werden in ImaAmp zurückgeliefert.

Für alle Filter außer *'sobel_fast'* werden die Kantenrichtungen in ImaDir zurückgeliefert. Für *'sobel_fast'* wird die Kantenrichtung aus Geschwindigkeitsgründen nicht berechnet. Deshalb ist in diesem Fall ImaDir ein leeres Bildobjekt. Die Kantenoperatoren *'deriche1'* bzw. *'deriche2'* stehen auch für int4-Bilder zur Verfügung. In dem Fall liefert die Routine statt der Amplituden, also dem Betrag der Filterantwort, die eigentliche, vorzeichenbehaftete Filterantwort als int4-Bild. Dieses Verhalten lässt sich durch Verwendung von *'deriche1_int4'* bzw. *'deriche2_int4'* auch für Byte-Bilder erzwingen. Auf die vorzeichenbehafteten Filterantworten lässt sich edges_image (mit Parameterwert *'lanser2'*) neuerlich anwenden, um die zweite Ableitung im Bild zu bestimmen. Die Kantenrichtungen werden in 2-Grad-Schritten kodiert, d.h. eine Kantenrichtung von x Grad (im mathematisch positiven Sinn und bezogen auf die Horizontale) wird zu $x/2$ im entsprechenden Ergebnisbild. Außerdem wird auch die Richtung der Helligkeitsänderung berücksichtigt. Bezeichnet $[E_x, E_y]$ den Bildgradienten, ergeben sich folgende Kantenrichtungen r (zurückgeliefert als $r/2$) zwischen 0 und 359 Grad:

Helligkeitszunahme	E_x/E_y	
Kantenrichtung r		
von unten nach oben	$0/+$	0
von rechts unten nach links oben	$-/+$	$]0, 90[$
von rechts nach links	$-/0$	90
von rechts oben nach links unten	$-/-$	$]90, 180[$
von oben nach unten	$0/-$	180
von links oben nach rechts unten	$+/-$	$]180, 270[$
von links nach rechts	$+/0$	270
von links unten nach rechts oben	$+/+$	$]270, 360[$

In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert.

Die „Filterbreite“ (i.e. das Einzugsgebiet der Filter) ist für alle Filter außer *'sobel_fast'* (wo die Filtergröße 3×3 ist und `Alpha` ignoriert wird) frei wählbar und mittels `info_edges` für konkrete Werte des Parameters `Alpha` abzuschätzen. Sie nimmt bei den Deriche-, Lanser- und Shen-Filtern mit `Alpha` ab bzw. beim Canny-Filter zu (`Alpha` ist in diesem Fall die Standardabweichung der zugrundeliegenden Gaußfunktion). „Breite“ Filter weisen eine höhere Rauschinvarianz, aber auch ein verringertes Auflösungsvermögen für Bilddetails auf. Nicht-rekursive Filter, wie hier die Canny-Filter, werden häufig mittels Filtermasken realisiert. In diesem Fall erhöht sich die Laufzeit natürlich mit wachsender Filterbreite. Die Laufzeit der rekursiver Filter ist hingegen konstant. Das Einzugsgebiet der Deriche-, Lanser bzw. Shen-Filter ist also ohne Mehraufwand beliebig vergrößerbar. Der daraus resultierende Laufzeitvorteil gegenüber dem Canny-Operator nimmt naturgemäß mit wachsender „Filterbreite“ zu. Zur Randbehandlung wird bei den rekursiven Filtern angenommen, das Bildsignal falle außerhalb des Bildes auf Null ab. Beim Canny-Operator wird hingegen das Bild durch „Vervielfachung“ der Randpunkte fortgesetzt. Das Signal-Rausch-Verhältnis der `Filter` ist vergleichbar, wenn `Alpha` wie folgt gewählt wird:

$$\begin{aligned}
\texttt{Alpha}('lanser1') &= \texttt{Alpha}('deriche1') \\
\texttt{Alpha}('deriche2') &= \texttt{Alpha}('deriche1')/2 \\
\texttt{Alpha}('lanser2') &= \texttt{Alpha}('deriche2') \\
\texttt{Alpha}('shen') &= \texttt{Alpha}('deriche1')/2 \\
\texttt{Alpha}('mshen') &= \texttt{Alpha}('shen') \\
\texttt{Alpha}('canny') &= 1.77/\texttt{Alpha}('deriche1')
\end{aligned}$$

Die hier eingesetzten rekursiven Filter verzerren in ihrer ursprünglichen Form (*'deriche1'*, *'deriche2'*, *'shen'*) die Amplituden diagonaler Kanten. Diese Fehler sind in den zugehörigen modifizierten Operatoren *'lanser1'*, *'lanser2'* und *'mshen'* (bei unveränderter Laufzeit) beseitigt.

Bei relativ kleinem Einzugsgebiet der Filter (11×11) Bildpunkte, etwa für `Alpha`('lanser2') = *0.5* liefern alle (modifizierten) Operatoren praktisch identische Ergebnisse. Erst bei „breiteren“ Filtern zeigen sich Unterschiede: Insbesondere fallen dann die Shen-basierten Operatoren qualitativ ab. Allerdings sind sie die schnellsten der implementierten Operatoren — dicht gefolgt von den Deriche-Operatoren.

`edges_image` bietet (optional) die Weiterverarbeitung der Filterergebnisse mittels Non-Maximum-Suppression (`NMS` = *'nms'*/*'inms'*/*'hvnms'*, sonst Wert *'none'*) und Hysterese-Schwellenwertoperation (`Low`,`High`, falls nicht erwünscht: einer der Werte negativ). Dies entspricht im wesentlichen nachfolgenden Aufrufen der Routinen

```
nonmax_suppression_dir(...,NMS,...)
hysteresis_threshold(...,Low,High,999,...)
```

Die Hysterese-Schwellenwertoperation wird nicht angewandt, wenn `NMS` auf *'none'* gesetzt ist.

Für *'sobel_fast'* wird für alle Werte von `NMS` außer *'none'* dieselbe Non-Maximum-Suppression ausgeführt. Zusätzlich wird für *'sobel_fast'* auch noch eine Verdünnung der Kanten auf ein Pixel Breite durchgeführt.

`edges_image` kann für die Filtertypen *'canny'* und *'sobel_fast'* auf OpenCL-Geräten ausgeführt werden.

Achtung

Die OpenCL-Implementierung von `edges_image` wird generell von der CPU-Implementierung abweichende Ergebnisse berechnen.

Da `edges_image` für den *'canny'*-Filter intern Gauss-Faltung verwendet, gelten dieselben Einschränkungen wie für `derivate_gauss`: `Alpha` muss so gewählt werden, dass die resultierende Filtermaske kleiner als 129 Pixel ist.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Eingabebilder.
- ▷ **`ImaAmp`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / int4 / real
  Amplitudenbilder.
- ▷ **`ImaDir`** (output_object) ..... image(-array) ⇝ *object :* direction
  Richtungsbilder.

▷ **Filter** (input_control) . . . . . string ⇝ *string*
Gewünschter Kanten-Operator.
Defaultwert: 'canny'
Werteliste: `Filter` ∈ {'deriche1', 'deriche1_int4', 'deriche2', 'deriche2_int4', 'lanser1', 'lanser2', 'shen', 'mshen', 'canny', 'sobel_fast'}

▷ **Alpha** (input_control) . . . . . real ⇝ *real*
Filterparameter: kleine Werte bewirken starke Glättung, also auch weniger Bilddetails (bei 'canny' umgekehrt).
Defaultwert: 1.0
Wertevorschläge: `Alpha` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.1}
Typischer Wertebereich: 0.2 ≤ `Alpha` ≤ 50.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Alpha > 0.0

▷ **NMS** (input_control) . . . . . string ⇝ *string*
Non-Maximum Suppression ('none', falls nicht erwünscht).
Defaultwert: 'nms'
Werteliste: `NMS` ∈ {'nms', 'inms', 'hvnms', 'none'}

▷ **Low** (input_control) . . . . . integer ⇝ *integer* / real
Untere Schwelle für Hysterese-Schwellenwertoperation (negativ, falls keine Schwellenwertbildung erwünscht).
Defaultwert: 20
Wertevorschläge: `Low` ∈ {5, 10, 15, 20, 25, 30, 40}
Typischer Wertebereich: 1 ≤ `Low` ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: Low > 1 || Low < 0

▷ **High** (input_control) . . . . . integer ⇝ *integer* / real
Obere Schwelle für Hysterese-Schwellenwertoperation (negativ, falls keine Schwellenwertbildung erwünscht).
Defaultwert: 40
Wertevorschläge: `High` ∈ {10, 15, 20, 25, 30, 40, 50, 60, 70}
Typischer Wertebereich: 1 ≤ `High` ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: High > 1 || High < 0 && High >= Low

Beispiel

```
read_image(Image,'fabrik')
edges_image(Image,Amp,Dir,'lanser2',0.5,'none',-1,-1)
hysteresis_threshold(Amp,Margin,20,30,30)
```

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `edges_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`info_edges`

Nachfolger

threshold, hysteresis_threshold, close_edges_length

Alternativen

sobel_dir, frei_dir, kirsch_dir, prewitt_dir, robinson_dir

Siehe auch

info_edges, nonmax_suppression_amp, hysteresis_threshold, bandpass_image

Literatur

S.Lanser, W.Eckstein: „Eine Modifikation des Deriche-Verfahrens zur Kantendetektion“; 13. DAGM-Symposium, München; Informatik Fachberichte 290; Seite 151 - 158; Springer-Verlag; 1991.

S.Lanser: „Detektion von Stufenkanten mittels rekursiver Filter nach Deriche“; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

J.Canny: „Finding Edges and Lines in Images“; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge; 1983.

J.Canny: „A Computational Approach to Edge Detection“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-8, vol. 6; S. 679-698; 1986.

R.Deriche: „Using Canny’s Criteria to Derive a Recursively Implemented Optimal Edge Detector“; International Journal of Computer Vision; vol. 1, no. 2; S. 167-187; 1987.

R.Deriche: „Optimal Edge Detection Using Recursive Filtering“; Proc. of the First International Conference on Computer Vision, London; S. 501-505; 1987.

R.Deriche: „Fast Algorithms for Low-Level Vision“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-12, no. 1; S. 78-87; 1990.

S.Castan, J.Zhao und J.Shen: „Optimal Filter for Edge Detection Methods and Results“; Proc. of the First European Conference on Computer Vision, Antibes; Lecture Notes on computer Science; no. 427; S. 12-17; Springer-Verlag; 1990.

Modul

Foundation

edges_sub_pix (Image : Edges : Filter, Alpha, Low, High :)

Sub-pixel-genaue Kantendetektion mit Deriche-, Lanser-, Shen- oder Canny-Filtern.

edges_sub_pix dient der Detektion von Stufenkanten mittels rekursiver Filter (nach Deriche, Lanser und Shen) bzw. mittels des konventionell über Filtermasken realisierten „Derivative of Gaussian“ Filter, den auch Canny verwendet. Konkret stehen damit für Filter folgende Kantenoperatoren zur Verfügung:

'deriche1', *'lanser1'*, *'deriche2'*, *'lanser2'*, *'shen'*, *'mshen'*, *'canny'*, *'sobel'* und *'sobel_fast'*.

Die extrahierten Kanten werden als sub-pixel-genaue XLD-Konturen in Edges zurückgeliefert. Für alle Kantenoperatoren außer *'sobel_fast'* werden für jeden Kantenpunkt die folgenden Attribute definiert (Für weitere Informationen siehe get_contour_attrib_xld):

'edge_direction': Gibt die Richtung der Kante (nicht der Kontur), berechnet aus den Grauwertgradienten in Spalten- und Zeilenrichtung, an. Die Winkel sind bezüglich der Zeilenachse gegen den Uhrzeigersinn gegeben.

'angle': Richtung der Normalenvektoren der Kontur in Radiant. (so orientiert, dass die Normalenvektoren auf die rechte Seite der Kontur in Durchlaufrichtung der Kontur zeigen; die Winkel sind bzgl. der Zeilenachse des Bildes angegeben).

'response': Die Amplitude der Kante.

Die „Filterbreite“ (bzw. das Einzugsgebiet der Filter) ist für alle Kantenoperatoren außer *'sobel'* und *'sobel_fast'* frei wählbar und lässt sich mittels info_edges für konkrete Werte des Parameters Alpha abschätzen. Bei allen Filtern (Deriche-, Lanser- und Shen-Filtern) bewirkt eine Vergrößerung von Alpha eine Verkleinerung der „Filterbreite“. Die einzige Ausnahme bildet der Canny-Filter, bei dem sich für größer werdende Werte für Alpha auch die „Filterbreite“ vergrößert. „Breite“ Filter weisen eine höhere Rauschinvarianz, aber auch ein verringertes Auflösungsvermögen für Bilddetails auf. Nicht-rekursive Filter, wie hier die Canny-Filter, werden häufig mittels

Filtermasken realisiert. In diesem Fall erhöht sich die Laufzeit natürlich mit wachsender Filterbreite. Die Laufzeit der rekursiven Filter ist hingegen konstant. Das Einzugsgebiet der Deriche-, Lanser bzw. Shen-Filter ist also ohne Mehraufwand beliebig vergrößerbar. Der daraus resultierennde Laufzeitvorteil gegenüber dem Canny-Operator nimmt naturgemäß mit wachsender „Filterbreite“ zu. Zur Randbehandlung wird bei den rekursiven Filtern angenommen, das Bildsignal falle außerhalb des Bildes auf Null ab. Beim Canny-Operator wird hingegen das Bild durch „Vervielfachung“ der Randpunkte fortgesetzt. Das Signal-Rausch-Verhältnis der `Filter` ist vergleichbar, wenn `Alpha` wie folgt gewählt wird:

$$\begin{aligned}
\texttt{Alpha}('lanser1') &= \texttt{Alpha}('deriche1')\\
\texttt{Alpha}('deriche2') &= \texttt{Alpha}('deriche1')/2\\
\texttt{Alpha}('lanser2') &= \texttt{Alpha}('deriche2')\\
\texttt{Alpha}('shen') &= \texttt{Alpha}('deriche1')/2\\
\texttt{Alpha}('mshen') &= \texttt{Alpha}('shen')\\
\texttt{Alpha}('canny') &= 1.77/\texttt{Alpha}('deriche1')
\end{aligned}$$

Die hier eingesetzten rekursiven Filter verzerren in ihrer ursprünglichen Form (*'deriche1'*, *'deriche2'*, *'shen'*) die Amplituden diagonaler Kanten. Diese Fehler sind in den zugehörigen modifizierten Operatoren *'lanser1'*, *'lanser2'* und *'mshen'* (bei unveränderter Laufzeit) beseitigt.

Bei relativ kleinem Einzugsgebiet der Filter (11×11) Bildpunkte, etwa für `Alpha` (*'lanser2' = '0.5)'* liefern alle (modifizierten) Operatoren praktisch identische Ergebnisse. Erst bei „breiteren“ Filtern zeigen sich Unterschiede: Insbesondere fallen dann die Shen-basierten Operatoren qualitativ ab. Allerdings sind sie die schnellsten der implementierten Operatoren, die eine freie Wahl der Filterbreite erlauben, dicht gefolgt von den Deriche-Operatoren. Die beiden Sobel-Filter, die eine feste Maskengröße von (3×3) verwenden, sind schneller als die anderen Filter, wobei der Filter *'sobel_fast'* erheblich schneller ist als der Filter *'sobel'*.

`edges_sub_pix` verknüpft die einzelnen Kantenpunkte mit einem Hysterese-Schwellenwert-artigem Verfahren zu Kanten, wie es auch in `lines_gauss` verwendet wird. Dabei werden Punkte, deren Amplitude größer als `High` ist, sofort als sichere Kantenpunkte akzeptiert. Punkte, deren Amplitude kleiner als `Low` ist, werden sofort verworfen. Alle Punkte, die eine zweite Ableitung zwischen diesen zwei Werten besitzen, werden akzeptiert, wenn sie durch einen Pfad mit sicheren Punkten verbunden sind (siehe `lines_gauss` und `hysteresis_threshold`).

Da Kantenextraktoren oftmals bestimmte Kreuzungspunkte nicht extrahieren können, kann durch Anhängen von *'_junctions'* an die oben beschriebenen Werte von `Filter` versucht werden, diese mit anderen Mitteln zu extrahieren. Dieser Modus ist analog zu dem bei `lines_gauss` verfügbaren Modus zur Vervollständigung von Kreuzungspunkten.

Der Kantenoperator *'sobel_fast'* hat dieselbe Semantik wie alle anderen Kantenoperatoren, basiert intern aber auf einer stark vereinfachten Variante der einzelnen Schritte (Hysterese-Schwellenwert-Operation, Verknüpfung der Kantenpunkte, Extraktion der Subpixel-Position). Daher können die Ergebnisse von *'sobel_fast'* in manchen Fällen eine geringere Genauigkeit als die Ergebnisse der anderen Filter aufweisen und andere Kantenteile selektieren.

`edges_sub_pix` kann für die Filtertypen *'canny'* und *'sobel_fast'* auf OpenCL-Geräten ausgeführt werden. Dazu werden bis zu Breite*Höhe*29 Bytes pinned Speicher benötigt. Daher sollte der pinned Speichercache auf mindestens diese Größe gesetzt werden (mit `set_compute_device_param` und dem Parameter *'pinned_mem_cache_capacity'*, oder die Verwendung von pinned Speicher komplett abzuschalten (mit `set_compute_device_param` und dem Parameter *'alloc_pinned'*) - dann wird der normale Speichercache verwendet. Die Ergebnisse können dabei von dem der CPU-Implementierung abweichen.

Achtung

Da `edges_sub_pix` für den *'canny'*-Filter intern Gauss-Faltung verwendet, gelten dieselben Einschränkungen wie für `derivate_gauss`: `Alpha` muss so gewählt werden, dass die resultierende Filtermaske kleiner als 129 Pixel ist.

In HALCON XL kann `edges_sub_pix` gar nicht auf OpenCL-Geräten ausgeführt werden, da dafür doppelt genaue Fließkommaarithmetik notwendig wäre, OpenCL-Geräte aber nur für einfach genaue Fließkommaarithmetik optimiert sind.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **Edges** (output_object) . . . . . xld_cont-array ⇝ *object*
  Extrahierte Kanten.
- ▷ **Filter** (input_control) . . . . . string ⇝ *string*
  Gewünschter Kanten-Operator.
  Defaultwert: 'canny'
  Werteliste: `Filter` ∈ {'deriche1', 'lanser1', 'deriche2', 'lanser2', 'shen', 'mshen', 'canny', 'sobel', 'sobel_fast', 'deriche1_junctions', 'lanser1_junctions', 'deriche2_junctions', 'lanser2_junctions', 'shen_junctions', 'mshen_junctions', 'canny_junctions', 'sobel_junctions'}
- ▷ **Alpha** (input_control) . . . . . real ⇝ *real*
  Filterparameter: kleine Werte bewirken starke Glättung, also auch weniger Bilddetails (bei 'canny' umgekehrt).
  Defaultwert: 1.0
  Wertevorschläge: `Alpha` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.1}
  Typischer Wertebereich: 0.2 ≤ `Alpha` ≤ 50.0
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: Alpha > 0.0
- ▷ **Low** (input_control) . . . . . integer ⇝ *integer* / real
  Untere Schwelle für Hysterese-Schwellenwertoperation.
  Defaultwert: 20
  Wertevorschläge: `Low` ∈ {5, 10, 15, 20, 25, 30, 40}
  Typischer Wertebereich: 1 ≤ `Low` ≤ 255
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 5
  Restriktion: Low > 0
- ▷ **High** (input_control) . . . . . integer ⇝ *integer* / real
  Obere Schwelle für Hysterese-Schwellenwertoperation.
  Defaultwert: 40
  Wertevorschläge: `High` ∈ {10, 15, 20, 25, 30, 40, 50, 60, 70}
  Typischer Wertebereich: 1 ≤ `High` ≤ 255
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 5
  Restriktion: High > 0 && High >= Low

Beispiel

```
read_image(Image,'fabrik')
edges_sub_pix(Image,Edges,'lanser2',0.5,20,40)
```

Komplexität

Sei A die Anzahl von Pixeln in der Region von `Image`. Dann ist die Laufzeitkomplexität $O(A * Alpha)$ für den Canny-Filter und $O(A)$ für die rekursiven Lanser-, Deriche- und Shen-Filter.

Der Bedarf an temporären Speicher ist abhängig von der Höhe H der Region von `Image` und der Breite W von `Image`. Sei $S = W * H$, dann benötigt `edges_sub_pix` mindestens $60 * S$ Bytes an temporärem Speicher bei der Ausführung für alle Kantenoperatoren außer *'sobel_fast'*. Für *'sobel_fast'* werden mindestens $9 * S$ Bytes an temporärem Speicher benötigt.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `edges_sub_pix` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`segment_contours_xld`, `gen_polygons_xld`, `select_shape_xld`

Alternativen

`sobel_dir`, `frei_dir`, `kirsch_dir`, `prewitt_dir`, `robinson_dir`, `edges_image`

Siehe auch

`info_edges`, `hysteresis_threshold`, `bandpass_image`, `lines_gauss`, `lines_facet`

Literatur

S.Lanser, W.Eckstein: „Eine Modifikation des Deriche-Verfahrens zur Kantendetektion“; 13. DAGM-Symposium, München; Informatik Fachberichte 290; Seite 151 - 158; Springer-Verlag; 1991.

S.Lanser: „Detektion von Stufenkanten mittels rekursiver Filter nach Deriche“; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

J.Canny: „Finding Edges and Lines in Images“; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge; 1983.

J.Canny: „A Computational Approach to Edge Detection“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-8, vol. 6; S. 679-698; 1986.

R.Deriche: „Using Canny’s Criteria to Derive a Recursively Implemented Optimal Edge Detector“; International Journal of Computer Vision; vol. 1, no. 2; S. 167-187; 1987.

R.Deriche: „Optimal Edge Detection Using Recursive Filtering“; Proc. of the First International Conference on Computer Vision, London; S. 501-505; 1987.

R.Deriche: „Fast Algorithms for Low-Level Vision“; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-12, no. 1; S. 78-87; 1990.

S.Castan, J.Zhao und J.Shen: „Optimal Filter for Edge Detection Methods and Results“; Proc. of the First European Conference on Computer Vision, Antibes; Lecture Notes on computer Science; no. 427; S. 12-17; Springer-Verlag; 1990.

Modul

2D Metrology

frei_amp (Image : ImageEdgeAmp : :)

Kantendetektion (Amplitude) mit dem Frei-Chen-Operator.

`frei_amp` berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort der Maske A und B eingetragen.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
Eingabebild.

▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
Kantenamplitude.

Beispiel

```
read_image(Image,'fabrik')
frei_amp(Image,Frei_amp)
threshold(Frei_amp,Edges,128,255)
```

Ergebnis

`frei_amp` liefert immer den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`binomial_filter`, `gauss_filter`, `sigma_image`, `median_image`, `smooth_image`

Alternativen

`sobel_amp`, `kirsch_amp`, `prewitt_amp`, `robinson_amp`, `roberts`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Foundation

frei_dir (Image : ImageEdgeAmp, ImageEdgeDir : :)

Kantendetektion (Amplitude und Richtung) mit dem Frei-Chen-Operator.

`frei_dir` berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort der Maske A und B eingetragen. In dem Parameter `ImageEdgeDir` wird die Kantenrichtungen zurückgeliefert. Diese werden in 2 Grad Schritten kodiert, d.h. eine Kantenrichtung von x Grad (im mathematisch positiven Sinn und bezogen auf die Horizontale) wird zu $x/2$ im entsprechenden Ergebnisbild. Außerdem wird auch die Richtung der Helligkeitsänderung berücksichtigt. Bezeichnet $[Ex, Ey]$ den Bildgradienten, ergeben sich folgende Kantenrichtungen r (zurückgeliefert als $r/2$) zwischen 0 und 359 Grad:

Helligkeitszunahme E_x/E_y		$Kantenrichtung$r
von unten nach oben	$0/+$	0
von rechts unten nach links oben	$-/+$	$]0, 90[$
von rechts nach links	$-/0$	90
von rechts oben nach links unten	$-/-$	$]90, 180[$
von oben nach unten	$0/-$	180
von links oben nach rechts unten	$+/-$	$]180, 270[$
von links nach rechts	$+/0$	270
von links unten nach rechts oben	$+/+$	$]270, 360[$

In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Kantenamplitude.
- ▷ **ImageEdgeDir** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* direction
  Kantenrichtung.

Beispiel

```
read_image(Image,'fabrik')
frei_dir(Image,Frei_dirA,Frei_dirD)
threshold(Frei_dirA,Res,128,255)
```

Ergebnis

`frei_dir` liefert immer den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`binomial_filter`, `gauss_filter`, `sigma_image`, `median_image`, `smooth_image`

Nachfolger

`hysteresis_threshold`, `threshold`, `gray_skeleton`, `nonmax_suppression_dir`, `close_edges`, `close_edges_length`

Alternativen

`edges_image`, `sobel_dir`, `robinson_dir`, `prewitt_dir`, `kirsch_dir`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Foundation

highpass_image (Image : Highpass : Width, Height :)

Verstärkung hochfrequenter Bildanteile.

highpass_image verstärkt hochfrequente Bildanteile durch eine lineare Filterung mit folgender Filtermatrix (am Beispiel 7×5):

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -35 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{matrix}$$

Die Anwendung der Prozedur entspricht der Anwendung des Mittelwertfilters (mean_image), gefolgt von der Subtraktion des Originalgrauwerts vom Ergebnis. Zu der Differenz wird 128 addiert, d.h. der Nulldurchgang hat den Wert 128.

Der Filter hebt hochfrequente Anteile im Bild hervor (Kanten und Ecken). Die Trennfrequenz wird über die Größe (Height $\times$ Width) der Filtermatrix festgelegt: Je größer die Matrix ist, desto niedriger wird die Trennfrequenz.

Als Randbehandlungen werden die Grauwerte an den Objekträndern gespiegelt. Über- bzw. Unterlauf von Grauwerten wird beschnitten (255 bzw. 0).

Achtung

Werden für Height oder Width gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

highpass_image kann auf OpenCL-Geräten ausgeführt werden. Dabei gelten dieselben Einschränkungen wie für mean_image und sub_image.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Eingabebilder.
- ▷ **Highpass** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Hochpass-gefilterte Ausgabebilder.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert: 9
  Wertevorschläge: Width $\in$ {3, 5, 7, 9, 11, 13, 17, 21, 29, 41, 51, 73, 101}
  Typischer Wertebereich: $3 \leq$ Width ≤ 501
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
  Restriktion: Width >= 3 && odd(Width)
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert: 9
  Wertevorschläge: Height $\in$ {3, 5, 7, 9, 11, 13, 17, 21, 29, 41, 51, 73, 101}
  Typischer Wertebereich: $3 \leq$ Height ≤ 501
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
  Restriktion: Height >= 3 && odd(Height)

Beispiel

```
highpass_image(Image,&Highpass,7,5);
threshold(Highpass,&Region,60.0,255.0);
skeleton(Region,&Skeleton);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `highpass_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`threshold`, `skeleton`

Alternativen

`mean_image`, `sub_image`, `convol_image`, `bandpass_image`

Siehe auch

`dyn_threshold`

Modul

Foundation

info_edges (: : Filter, Mode, Alpha : Size, Coeffs)

Abfragen der Filterkoeffizienten der Filter aus `edges_image`.

`info_edges` liefert die Filterkoeffizienten `Coeffs` und eine Abschätzung des Einzugsgebiets `Size` der in der Routine `edges_image` verwendeten Filter. Dazu werden die zugrundeliegenden kontinuierlichen Impulsantworten der Filter abgetastet, bis ein Filterkoeffizient kleiner als fünf Prozent des Maximalkoeffizienten ist. `Alpha` ist der Filterparameter (siehe `edges_image`). Es werden sieben Kantenoperatoren unterstützt (Parameter `Filter`):

'deriche1', 'lanser1', 'deriche2', 'lanser2', 'shen', 'mshen' und 'canny'.

Über den Parameter `Mode` ('edge'/'smooth') wird festgelegt, ob der entsprechende Kanten- oder Glättungsfilter gemeint ist.

Der Canny-Operator (der sich auf die Gaußfunktion abstützt) wurde dabei konventionell mittels Filtermasken implementiert. Im Falle der Canny-Filter werden daher die Filterkoeffizienten der eindimensionalen Impulsantworten $f(n)$ mit $n \geq 0$ in `Coeffs` zurückgeliefert. Alle anderen `Filter` sind rekursiv implementiert. Hierbei werden die Koeffizienten `Coeffs` einer entsprechenden nicht-rekursiven Implementierung des Filters zurückgegeben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Filter** (input_control) ..... string ⇝ *string*
Name des Kanten-Operators.
Defaultwert: 'lanser2'
Werteliste: `Filter` ∈ {'deriche1', 'lanser1', 'deriche2', 'lanser2', 'shen', 'mshen', 'canny'}

▷ **Mode** (input_control) ..... string ⇝ *string*
1D-Kantenfilter ('edge') oder 1D-Glättungsfilter ('smooth').
Defaultwert: 'edge'
Werteliste: `Mode` ∈ {'edge', 'smooth'}

▷ **Alpha** (input_control) ..... real ⇝ *real*
Filterparameter: kleine Werte bewirken starke Glättung, also auch weniger Bilddetails (bei canny umgekehrt).
Defaultwert: 0.5
Typischer Wertebereich: $0.2 \leq$ `Alpha` ≤ 50.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Alpha > 0.0

▷ **Size** (output_control) ..... integer ⇝ *integer*
Größe des Einzugsgebietes des Filters in Pixeln.

▷ **Coeffs** (output_control) ..... integer-array ⇝ *integer*
Falls Cannyfilter: Koeffizienten der „positiven“ Hälfte der 1D-Impulsantwort. Alle anderen: Koeffizienten einer entsprechenden nicht-rekursiven Implementierung des Filters.

Beispiel

```
read_image(Image,'fabrik')
info_edges('lanser2','edge',0.5,Size,Coeffs)
edges_image(Image,Amp,Dir,'lanser2',0.5,'none',-1,-1)
hysteresis_threshold(Amp,Margin,20,30,30)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `info_edges` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`edges_image`, `threshold`, `skeleton`

Siehe auch

`edges_image`

Modul

Foundation

kirsch_amp (Image : ImageEdgeAmp : :)

Kantendetektion mit dem Kirsch-Operator.

`kirsch_amp` berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende acht Filtermasken zugrunde:

$$\begin{matrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{matrix}$$

$$\begin{matrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort aller Masken eingetragen.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... image(-array) ⇝ *object :* byte / int2 / uint2
  Kantenamplitude.

Beispiel

```
read_image(Image,'fabrik')
kirsch_amp(Image,Kirsch_amp)
threshold(Kirsch_amp,Edges,128,255)
```

Ergebnis

`kirsch_amp` liefert immer den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`binomial_filter`, `gauss_filter`, `sigma_image`, `median_image`, `smooth_image`

Alternativen

`sobel_amp`, `frei_amp`, `prewitt_amp`, `robinson_amp`, `roberts`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Foundation

kirsch_dir (Image : ImageEdgeAmp, ImageEdgeDir : :)

Kantendetektion (Amplitude und Richtung) mit dem Kirsch-Operator.

kirsch_dir berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende acht Filtermasken zugrunde:

$$\begin{matrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{matrix}$$

$$\begin{matrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort aller Masken eingetragen. In dem Parameter ImageEdgeDir wird die Kantenrichtungen zurückgeliefert als $x/2$. Die Kantenrichtungen entsprechen der Richtung der Maske mit der maximalen Filterantwort.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Kantenamplitude.
- ▷ **ImageEdgeDir** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* direction
  Kantenrichtung.

Beispiel

```
read_image(Image,'fabrik')
kirsch_dir(Image,Kirsch_dirA,Kirsch_dirD)
threshold(Kirsch_dirA,Res,128,255)
```

Ergebnis

`kirsch_dir` liefert immer den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`binomial_filter`, `gauss_filter`, `sigma_image`, `median_image`, `smooth_image`

Nachfolger

`hysteresis_threshold`, `threshold`, `gray_skeleton`, `nonmax_suppression_dir`, `close_edges`, `close_edges_length`

Alternativen

`edges_image`, `sobel_dir`, `robinson_dir`, `prewitt_dir`, `frei_dir`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Foundation

laplace (Image : ImageLaplace : ResultType, MaskSize, FilterMask :)

Näherung des Laplace-Operators durch finite Differenzen.

`laplace` filtert die Eingabebilder (`Image`) mittels des Laplace-Operators. Abhängig vom Parameter `FilterMask` werden folgende lokale Approximationen an den Laplace-Operator zugrundegelegt:

'n_4'

$$\begin{matrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{matrix}$$

'n_8'

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}$$

'n_8_isotropic'

$$\begin{matrix} 10 & 22 & 10 \\ 22 & -128 & 22 \\ 10 & 22 & 10 \end{matrix}$$

Für die einzelnen Filtermasken werden folgende Normierungen der Ergebnisgrauwerte, (d.h. Division durch die angegebene Zahl) durchgeführt: 'n_4' Normierung durch 1, 'n_8', Normierung durch 2 und 'n_8_isotropic' Normierung durch 32.

Für eine 3×3 Laplace-Filterung (`MaskSize` = 3) wird der entsprechende Filter direkt angewandt. Für größere Laplace-Filter wird das Eingabebild dagegen zunächst mit dem Gaußfilter (siehe `gauss_image`) oder einem Binomialfilter (siehe `binomial_filter`) der Größe `MaskSize`-2 geglättet. Der Gaußfilter wird durch die oben angegebenen Werte von `ResultType` ausgewählt. Hierbei ist `MaskSize` = 5, 7, 9, 11 oder 13 zu wählen. Der

Binomialfilter wird ausgewählt, indem *'_binomial'* an die oben angegebenen Werte von `ResultType` angehängt wird. Hier kann `MaskSize` zwischen 5 und 39 gewählt werden. Zusätzlich ist es hier möglich, unterschiedliche Glättungen in Spalten- und Zeilenrichtung durch Angabe von zwei Werten in `MaskSize` zu erreichen. Dabei entspricht der erste Wert der Maskenbreite (Glättung in Spaltenrichtung) und der zweite Wert der Maskenhöhe (Glättung in Zeilenrichtung) des Binomialfilters. Somit ist z.B.

```
laplace(O:R:'absolute',MaskSize,N:)
```

für `MaskSize` > 3 äquivalent zu

```
gauss_image(O:G:MaskSize-2:) ▷
laplace(G:R:'absolute',3,N:)
```

und

```
laplace(O:R:'absolute_binomial',MaskSize,N:)
```

is äquivalent zu

```
binomial_filter(O:B:MaskSize-2,MaskSize-2:) ▷
laplace(B:R:'absolute',3,N:)
```

`laplace` liefert entweder den Absolutbetrag des Laplace-gefilterten Eingabebildes (`ResultType` = *'absolute'*) in einem byte/uint2-Bild oder unmittelbar das Filterergebnis (`ResultType` = *'signed'* oder *'signed_clipped'*) mit einem Pixeltyp mit Vorzeichen zurück. Dabei ist bei *'signed_clipped'* der Ausgabebildtyp von derselben Anzahl von Bytes wie der Eingabebildtyp (also int1 oder int2), während bei *'signed'* der Ausgabebildtyp die nächsthöhere Anzahl an Bytes besitzt (also int2 oder int4).

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **`ImageLaplace`** (output_object) .... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / int2 / int2 / int4
  Laplace-gefiltertes Ausgabebild.
- ▷ **`ResultType`** (input_control) ..... string ⇝ *string*
  Typ des Ergebnisbildes, wobei für byte und uint2 der Absolutbetrag zurückgegeben wird.
  Defaultwert: 'absolute'
  Werteliste: `ResultType` ∈ {'absolute', 'signed_clipped', 'signed', 'absolute_binomial', 'signed_clipped_binomial', 'signed_binomial'}
- ▷ **`MaskSize`** (input_control) ..... integer(-array) ⇝ *integer*
  Filtergröße.
  Defaultwert: 3
  Werteliste: `MaskSize` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39}
- ▷ **`FilterMask`** (input_control) ..... string ⇝ *string*
  Filtermaske für den Laplace-Operator
  Defaultwert: 'n_4'
  Werteliste: `FilterMask` ∈ {'n_4', 'n_8', 'n_8_isotropic'}

Beispiel

```
read_image(&Image,"mreut");
laplace(Image,&Laplace,"signed",3,"n_8_isotropic");
zero_crossing(Laplace,&ZeroCrossings);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `laplace` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`zero_crossing`, `dual_threshold`, `threshold`

Alternativen

`diff_of_gauss`, `laplace_of_gauss`, `derivate_gauss`

Siehe auch

`highpass_image`, `edges_image`

Modul

Foundation

`laplace_of_gauss` (Image : ImageLaplace : Sigma :)

LoG-Operator (Laplace of Gaussian).

`laplace_of_gauss` berechnet den Laplaceoperator für beliebige `Sigma`. Die Formel für den Laplace-Operator lautet:

$$\Delta g(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}$$

Die Ableitungen bei `laplace_of_gauss` werden durch Ableitungen der Gaußfunktion angenähert, wodurch sich folgende Formel für den Faltungsoperator ergibt:

$$\Delta G_\sigma(x,y) =$$
$$\frac{1}{2\pi\sigma^4}\left(\frac{x^2+y^2}{2\sigma^2}-1\right)\left[\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\right]$$

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
Eingabebild.

▷ **`ImageLaplace`** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* int2
Laplace des Eingabebildes.

▷ **`Sigma`** (input_control) . . . . . number ⇝ *real* / integer
Glättungsparameter der Gaußfunktion.
Defaultwert: 2.0
Wertevorschläge: `Sigma` ∈ {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0}
Typischer Wertebereich: 0.7 ≤ `Sigma` ≤ 5.0
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Sigma > 0.7 && Sigma <= 25.0

Beispiel

```
read_image(&Image,"mreut");
laplace_of_gauss(Image,&Laplace,2.0);
zero_crossing(Laplace,&ZeroCrossings);
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

zero_crossing, dual_threshold

Alternativen

laplace, diff_of_gauss, derivate_gauss

Siehe auch

derivate_gauss

Modul

Foundation

prewitt_amp (Image : ImageEdgeAmp : :)

Kantendetektion (Amplitude) mit dem Prewitt-Operator.

prewitt_amp berechnet eine Näherung der ersten Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort der Maske A und B eingetragen.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Kantenamplitude.

Beispiel

```
read_image(Image,'fabrik')
prewitt_amp(Image,Prewitt)
threshold(Prewitt,Edges,128,255)
```

Ergebnis

prewitt_amp liefert immer den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

binomial_filter, gauss_filter, sigma_image, median_image, smooth_image

Nachfolger

threshold, gray_skeleton, nonmax_suppression_amp, close_edges, close_edges_length

Alternativen

sobel_amp, kirsch_amp, frei_amp, robinson_amp, roberts

Siehe auch

bandpass_image, laplace_of_gauss

Modul

Foundation

prewitt_dir (Image : ImageEdgeAmp, ImageEdgeDir : :)

Kantendetektion (Amplitude und Richtung) mit dem Prewitt-Operator.

prewitt_dir berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort der Maske A und B eingetragen. In dem Parameter ImageEdgeDir wird die Kantenrichtungen zurückgeliefert. Diese werden in 2 Grad Schritten kodiert, d.h. eine Kantenrichtung von x Grad (im mathematisch positiven Sinn und bezogen auf die Horizontale) wird zu $x/2$ im entsprechenden Ergebnisbild. Außerdem wird auch die Richtung der Helligkeitsänderung berücksichtigt. Bezeichnet $[Ex, Ey]$ den Bildgradienten, ergeben sich folgende Kantenrichtungen r (zurückgeliefert als $r/2$) zwischen 0 und 359 Grad:

Helligkeitszunahme	E_x/E_y	$Kantenrichtung$r
von unten nach oben	$0/+$	0
von rechts unten nach links oben	$-/+$	$]0, 90[$
von rechts nach links	$-/0$	90
von rechts oben nach links unten	$-/-$	$]90, 180[$
von oben nach unten	$0/-$	180
von links oben nach rechts unten	$+/-$	$]180, 270[$
von links nach rechts	$+/0$	270
von links unten nach rechts oben	$+/+$	$]270, 360[$

In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Kantenamplitude.
- ▷ **ImageEdgeDir** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* direction
  Kantenrichtung.

Beispiel

```
read_image(Image,'fabrik')
prewitt_dir(Image,PrewittA,PrewittD)
threshold(PrewittA,Edges,128,255)
```

Ergebnis

`prewitt_dir` liefert immer den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`binomial_filter`, `gauss_filter`, `sigma_image`, `median_image`, `smooth_image`

Nachfolger

`hysteresis_threshold`, `threshold`, `gray_skeleton`, `nonmax_suppression_dir`, `close_edges`, `close_edges_length`

Alternativen

`edges_image`, `sobel_dir`, `robinson_dir`, `frei_dir`, `kirsch_dir`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Foundation

roberts (Image : ImageRoberts : FilterType :)

Kantendetektion mit dem Roberts-Filter.

`roberts` berechnet die erste Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Legt man folgende Maske zugrunde,

$$\begin{matrix} A & B \\ C & D \end{matrix}$$

dann leiten sich die verschiedenen Filtertypen wie folgt ab:

'roberts_max'	$max(\|A-D\|,\|B-C\|)$
'gradient_max'	$max(\|A+B-(C+D)\|,\|A+C-(B+D)\|)$
'gradient_sum'	$\|A+B-(C+D)\|+\|A+C-(B+D)\|$

Tritt ein Überlauf ein, so werden die Werte beschnitten. Das Ergebnis der Berechnung wird in dem Pixel mit den Koordinaten von „D“ abgelegt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Eingabebilder.
- ▷ **ImageRoberts** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Gefilterte Ausgabebilder.
- ▷ **FilterType** (input_control) ..... string ⇝ *string*
  Filtertyp.
  Defaultwert: 'gradient_sum'
  Werteliste: `FilterType` ∈ {'roberts_max', 'gradient_max', 'gradient_sum'}

Beispiel

```
read_image(Image,'fabrik')
roberts(Image,Roberts,'roberts_max')
threshold(Roberts,Margin,128,255)
```

Ergebnis

`roberts` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`binomial_filter`, `gauss_filter`

Nachfolger

`threshold`, `skeleton`

Alternativen

`edges_image`, `sobel_amp`, `frei_amp`, `kirsch_amp`, `prewitt_amp`

Siehe auch

`laplace`, `highpass_image`, `bandpass_image`

Modul

Foundation

robinson_amp (Image : ImageEdgeAmp : :)

Kantendetektion (Amplitude) mit dem Robinson-Operator.

robinson_amp berechnet eine Näherung der ersten Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Mit robinson_amp werden auf ein Eingabebild vier der ursprünglich acht 3×3-Filter von Robinson angewandt. Die anderen 4 Masken entstehen durch Multiplikation der Masken mit -1. Die Masken besitzen nur die Werte 0,1,-1,2,-2.

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

$$\begin{matrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{matrix}$$

$$\begin{matrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{matrix}$$

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort aller Masken eingetragen.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Kantenamplitude.

Beispiel

```
read_image(Image,'fabrik')
robinson_amp(Image,Robinson_amp)
threshold(Robinson_amp,Edges,128,255)
```

Ergebnis

robinson_amp liefert immer den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

binomial_filter, gauss_filter, sigma_image, median_image, smooth_image

Alternativen

sobel_amp, frei_amp, prewitt_amp, roberts

Siehe auch

bandpass_image, laplace_of_gauss

Modul

Foundation

robinson_dir (Image : ImageEdgeAmp, ImageEdgeDir : :)

Kantendetektion (Amplitude und Richtung) mit dem Robinson-Operator.

robinson_dir berechnet eine Näherung der ersten Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Mit robinson_dir werden auf ein Eingabebild vier der ursprünglich acht 3×3-Filter von Robinson angewandt. Die anderen 4 Masken entstehen durch Multiplikation der Masken mit -1. Die Masken besitzen nur die Werte 0,1,-1,2,-2.

$$\begin{matrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{matrix}$$

$$\begin{matrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{matrix}$$

$$\begin{matrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{matrix}$$

$$\begin{matrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort aller Masken eingetragen. In dem Parameter ImageEdgeDir wird die Kantenrichtungen zurückgeliefert als $x/2$. Die Kantenrichtungen entsprechen der Richtung der Maske mit der maximalen Filterantwort.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Kantenamplitude.
- ▷ **ImageEdgeDir** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* direction
  Kantenrichtung.

Beispiel

```
read_image(Image,'fabrik')
robinson_dir(Image,Robinson_dirA,Robinson_dirD)
threshold(Robinson_dirA,Res,128,255)
```

Ergebnis

robinson_dir liefert immer den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

binomial_filter, gauss_filter, sigma_image, median_image, smooth_image

Nachfolger

hysteresis_threshold, threshold, gray_skeleton, nonmax_suppression_dir, close_edges, close_edges_length

Alternativen

edges_image, sobel_dir, kirsch_dir, prewitt_dir, frei_dir

Siehe auch

bandpass_image, laplace_of_gauss

Modul

Foundation

sobel_amp (Image : EdgeAmplitude : FilterType, Size :)

Kantendetektion (Amplitude) mit dem Sobel-Operator.

sobel_amp berechnet eine Art von erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

Diese Masken werden je nach Filtertyp unterschiedlich verwendet. (a und b stehen im folgenden für die Ergebnisse der Faltung mit A und B in einem Bildpunkt):

'sum_sqrt'	$\sqrt{a^2+b^2}/4$
'sum_abs'	$(\|a\|+\|b\|)/4$
'thin_sum_abs'	$(thin(\|a\|)+thin(\|b\|))/4$
'thin_max_abs'	$max(thin(\|a\|),thin(\|b\|))/4$
'x'	$b/4$ (b für Bilder vom Typ *'real'*)
'y'	$a/4$ (a für Bilder vom Typ *'real'*)

$thin(x)$ ist dabei für ein vertikales Maximum (Maske A) bzw. ein horizontales Maximum (Maske B) gleich x und 0 sonst. Damit wird bei *'thin_sum_abs'* und *'thin_max_abs'* eine Verdünnung des Gradientenbildes erreicht. Bei Byte-Bildern bzw. uint2/int2-Bildern wird bei dem Modus *'x'* bzw. *'y'* als Ergebnis ein int1-Bild bzw. int2-Bild zurückgeliefert. Für eine 3×3 Sobel-Filterung (Size = 3) werden die Filter A und B direkt angewandt. Für größere Sobel-Filter wird das Eingabebild dagegen zunächst mit dem Gaußfilter (siehe gauss_image) oder einem Binomialfilter (siehe binomial_filter) der Größe Size-2 geglättet. Der Gaußfilter wird durch die oben angegebenen Werte von FilterType ausgewählt. Hierbei ist Size = 5, 7, 9, 11 oder 13 zu wählen. Der Binomialfilter wird ausgewählt, indem *'_binomial'* an die oben angegebenen Werte von FilterType angehängt wird. Hier kann Size zwischen 5 und 39 gewählt werden. Zusätzlich ist es möglich, unterschiedliche Glättungen in Spalten- und Zeilenrichtung durch Angabe von zwei Werten in Size zu erreichen. Dabei entspricht der erste Wert der Maskenbreite (Glättung in Spaltenrichtung) und der zweite Wert der Maskenhöhe (Glättung in Zeilenrichtung) des Binomialfilters. Der Binomialfilter kann nur für Bilder vom Typ byte, uint2 und real verwendet werden. Da die Glättung die Kantenamplituden reduziert, wird in diesem Fall die Kantenamplitude noch mit 2 multipliziert, um einen Informationsverlust zu vermeiden. Somit ist

```
sobel_amp(I,E,Dir,FilterTyp,S)
```

für Size $>$ 3 konzeptuell äquivalent zu

```
scale_image(I,F,2,0)
gauss_image(F,G,S-2)
sobel_amp(G,E,FilterType,3)
```

bzw. zu

```
scale_image(I,F,2,0)
binomial_filter(F,G,S[0]-2,S[1]-2)
sobel_amp(G,E,FilterType,3).
```

Für `sobel_amp` sind für `FilterType` $=$ *'sum_abs'* spezielle Optimierungen unter Verwendung von SIMD-Technologie implementiert. Ob diese tatsächlich zur Anwendung kommen, kann über den Systemparameter *'sse2_enable'* beziehungsweise *'avx2_enable'* (siehe `set_system`) gesteuert werden. Falls *'sse2_enable'* oder *'avx2_enable'* auf *'true'* gesetzt wird (und die verwendete Hardware SIMD-Technologie unterstützt), wird die Berechnung intern unter Verwendung von SIMD-Technologie durchgeführt. Es ist zu beachten, dass der Beschleunigungseffekt von SIMD-Technologie bei großen, kompakten Eingaberegionen am größten ist. Im Einzelfall kann aber die Ausführung von `sobel_amp`, abhängig von der Eingaberegion und den Hardwareeigenschaften, mit SIMD-Technologie auch länger dauern als ohne.

`sobel_amp` kann für die Filtertypen *'sum_abs'*, *'sum_sqrt'*, *'x'* und *'y'* (sowie deren binomialen Varianten) auf OpenCL-Geräten ausgeführt werden. Dabei ist zu beachten, das bei Gauss-Filterung mit `Size` > 3 das Ergebnis von dem der CPU- Implementierung abweichen kann.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild.
- ▷ **`EdgeAmplitude`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* int1 / int2 / uint2 / real
  Ausgabebild mit Kantenamplitude.
- ▷ **`FilterType`** (input_control) ..... string ⇝ *string*
  Filtertyp.
  Defaultwert: 'sum_abs'
  Werteliste: `FilterType` ∈ {'sum_abs', 'thin_sum_abs', 'thin_max_abs', 'sum_sqrt', 'x', 'y', 'sum_abs_binomial', 'thin_sum_abs_binomial', 'thin_max_abs_binomial', 'sum_sqrt_binomial', 'x_binomial', 'y_binomial'}
- ▷ **`Size`** (input_control) ..... integer(-array) ⇝ *integer*
  Filtergröße.
  Defaultwert: 3
  Werteliste: `Size` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39}

Beispiel

```
read_image(Image,'fabrik')
sobel_amp(Image,Amp,'sum_abs',3)
threshold(Amp,Edg,128,255)
```

Ergebnis

`sobel_amp` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

binomial_filter, gauss_filter, mean_image, anisotropic_diffusion, sigma_image

Nachfolger

threshold, nonmax_suppression_amp, gray_skeleton

Alternativen

frei_amp, roberts, kirsch_amp, prewitt_amp, robinson_amp

Siehe auch

laplace, highpass_image, bandpass_image

Modul

Foundation

sobel_dir (Image : EdgeAmplitude, EdgeDirection : FilterType, Size :)

Kantendetektion (Amplitude und Richtung) mit dem Sobel-Operator.

sobel_dir berechnet eine Art von erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

Diese Masken werden je nach Filtertyp unterschiedlich verwendet. (a und b stehen im folgenden für die Ergebnisse der Faltung mit A und B in einem Bildpunkt):

'sum_sqrt'	$\sqrt{a^2 + b^2}/4$
'sum_abs'	$(\|a\| + \|b\|)/4$

Für eine 3×3 Sobel-Filterung (Size = 3) werden die Filter A und B direkt angewandt. Für größere Sobel-Filter wird das Eingabebild dagegen zunächst mit dem Gaußfilter (siehe gauss_image) oder einem Binomialfilter (siehe binomial_filter) der Größe Size-2 geglättet. Der Gaußfilter wird durch die oben angegebenen Werte von FilterType ausgewählt. Hierbei ist Size = 5, 7, 9, 11 oder 13 zu wählen. Der Binomialfilter wird ausgewählt, indem *'_binomial'* an die oben angegebenen Werte von FilterType angehängt wird. Hier kann Size zwischen 5 und 39 gewählt werden. Zusätzlich ist es möglich, unterschiedliche Glättungen in Spalten- und Zeilenrichtung durch Angabe von zwei Werten in Size zu erreichen. Dabei entspricht der erste Wert der Maskenbreite (Glättung in Spaltenrichtung) und der zweite Wert der Maskenhöhe (Glättung in Zeilenrichtung) des Binomialfilters. Der Binomialfilter kann nur für Bilder vom Typ byte, uint2 und real verwendet werden. Da die Glättung die Kantenamplituden reduziert, wird in diesem Fall die Kantenamplitude noch mit 2 multipliziert, um einen Informationsverlust zu vermeiden. Somit ist

```
sobel_dir(I,Amp,Dir,FilterTyp,S)
```

für Size > 3 konzeptuell äquivalent zu

```
scale_image(I,F,2,0)
gauss_image(F,G,S-2)
sobel_dir(G,Amp,Dir,FilterType,3:)
```

bzw. zu

```
scale_image(I,F,2,0)
binomial_filter(F,G,S[0]-2,S[1]-2)
sobel_dir(G,Amp,Dir,FilterType,3:).
```

Im Parameter `EdgeDirection` wird die Kantenrichtungen zurückgeliefert. Diese werden in 2-Grad-Schritten kodiert, d.h. eine Kantenrichtung von x Grad (im mathematisch positiven Sinn und bezogen auf die Horizontale) wird zu $x/2$ im entsprechenden Ergebnisbild. Außerdem wird auch die Richtung der Helligkeitsänderung berücksichtigt. Bezeichnet $[E_x, E_y]$ den Bildgradienten, ergeben sich folgende Kantenrichtungen r (zurückgeliefert als $r/2$) zwischen 0 und 359 Grad:

Helligkeitszunahme	E_x/E_y	Kantenrichtung r
von unten nach oben	$0/+$	0
von rechts unten nach links oben	$-/+$	$]0, 90[$
von rechts nach links	$-/0$	90
von rechts oben nach links unten	$-/-$	$]90, 180[$
von oben nach unten	$0/-$	180
von links oben nach rechts unten	$+/-$	$]180, 270[$
von links nach rechts	$+/0$	270
von links unten nach rechts oben	$+/+$	$]270, 360[$

In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert.

`sobel_dir` kann auf OpenCL-Geräten ausgeführt werden. Dabei ist zu beachten, dass bei Gauss-Filterung mit `Size` > 3 das Ergebnis von dem der CPU-Implementierung abweichen kann.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Eingabebild.
- ▷ **`EdgeAmplitude`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Kantenamplitude.
- ▷ **`EdgeDirection`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* direction
  Kantenrichtung.
- ▷ **`FilterType`** (input_control) ..... string ⇝ *string*
  Filtertyp.
  Defaultwert: 'sum_abs'
  Werteliste: `FilterType` ∈ {'sum_abs', 'sum_sqrt', 'sum_abs_binomial', 'sum_sqrt_binomial'}
- ▷ **`Size`** (input_control) ..... integer(-array) ⇝ *integer*
  Filtergröße.
  Defaultwert: 3
  Werteliste: `Size` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39}

Beispiel

```
read_image(Image,'fabrik')
sobel_dir(Image,Amp,Dir,'sum_abs',3)
threshold(Amp,Edg,128,255)
```

Ergebnis

`sobel_dir` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

binomial_filter, gauss_filter, mean_image, anisotropic_diffusion, sigma_image

Nachfolger

nonmax_suppression_dir, hysteresis_threshold, threshold

Alternativen

edges_image, frei_dir, kirsch_dir, prewitt_dir, robinson_dir

Siehe auch

roberts, laplace, highpass_image, bandpass_image

Modul

Foundation

10.9 Linien

bandpass_image (Image : ImageBandpass : FilterType :)

Kantendetektion mittels Bandpassfilterung.

bandpass_image dient als Kantenfilter. Der Operator führt eine lineare Filterung mit folgender Filtermatrix durch:

FilterType: *'lines'*
In Gegensatz zu Kantenfiltern sobel_amp detektiert dieser Filter keine Grauwertkanten sondern Linien, also zwei dicht zusammenliegende Kanten.

$$\begin{pmatrix} 0 & -2 & -2 & -2 & 0 \\ -2 & 0 & 3 & 0 & -2 \\ -2 & 3 & 12 & 3 & -2 \\ -2 & 0 & 3 & 0 & -2 \\ 0 & -2 & -2 & -2 & 0 \end{pmatrix}$$

Als Randbehandlungen werden die Grauwerte an den Objekträndern gespiegelt. Über- bzw. Unterlauf von Grauwerten wird beschnitten. Die Ergebnisbilder werden in ImageBandpass abgelegt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
Eingabebilder.

▷ **ImageBandpass** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
Bandpass-gefilterte Bilder.

▷ **FilterType** (input_control) ..... string ⇝ *string*
Filtertyp, derzeit nur 'lines'.
Defaultwert: 'lines'
Werteliste: FilterType ∈ {'lines'}

Beispiel

```
bandpass_image(Image,&LineImage,"lines");
threshold(LineImage,&Lines,60.0,255.0);
skeleton(Lines,&ThinLines);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `bandpass_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`threshold`, `skeleton`

Alternativen

`convol_image`, `topographic_sketch`, `texture_laws`

Siehe auch

`highpass_image`, `gray_skeleton`

Modul

Foundation

lines_color (Image : Lines : Sigma, Low, High, ExtractWidth, CompleteJunctions :)

Extraktion von Farblinien und deren Breite.

`lines_color` extrahiert Farblinien aus dem Eingabebild `Image` und gibt die extrahierten Linien als subpixelgenaue XLD-Konturen in `Lines` zurück. Farblinien werden dabei als dunkle Linien im Amplitudenbild des Farbkantenfilters definiert (siehe `edges_color`). Der verwendete Farbkantenfilter ist bei `lines_color` immer der Canny-Farbkantenfilter. Die benötigten partiellen Ableitungen des Bildes werden also immer mit partiellen Ableitungen von Gaußmasken berechnet (siehe `derivate_gauss`). Die zugehörige Glättung wird durch den Parameter `Sigma` festgelegt.

Durch die Definition von Farblinien als dunkle Linien im Amplitudenbild wird im Gegensatz zu `lines_gauss` bei einkanaligen Bildern nicht unterschieden, ob die Linien heller oder dunkler als die Umgebung sind. Weiterhin werden von `lines_color` auch Treppenstufenlinien zurückgeliefert, d.h. Linien, bei denen der Grauwert der Linie zwischen den Grauwerten in der linken und rechten Umgebung der Linie liegt. Bei mehrkanaligen Bildern kann durch die obige Definition sogar jeder Kanal einen unterschiedlichen Linientyp aufweisen. Bei einem dreikanaligen Bild kann z.B. der erste Kanal eine dunkle Linie, der zweite Kanal eine helle Linie und der dritte Kanal eine Treppenstufenlinie an derselben Position aufweisen.

Wenn `ExtractWidth` auf *'true'* gesetzt wird, so wird für jeden Linienpunkt die Linienbreite extrahiert. Da der Linienextraktor aus differentialgeometrischen Gründen bestimmte Kreuzungspunkte nicht extrahieren kann, wird versucht, diese mit anderen Mitteln zu extrahieren, falls `CompleteJunctions`=*'true'* gesetzt wird.

`lines_color` verknüpft die einzelnen Kantenpunkte mit einem Hysterese-Schwellenwert-artigen Verfahren, wie es auch in `lines_gauss` und `edges_color_sub_pix` verwendet wird, zu Linien. Dabei werden Punkte, deren Amplitude größer als `High` ist, sofort als sichere Linienpunkte akzeptiert. Punkte, deren Amplitude kleiner als `Low` ist, werden sofort verworfen. Alle Punkte, die eine Amplitude zwischen diesen zwei Werten besitzen, werden akzeptiert, wenn sie durch einen Pfad mit sicheren Punkten verbunden sind (siehe `lines_gauss`).

Dabei ist die Amplitude der Punkte als die Linienstärke der dunklen Linie im Amplitudenbild zu verstehen (siehe lines_gauss und lines_facet). Dieser Wert entspricht der dritten Richtungsableitung des geglätteten Eingabebildes in Richtung senkrecht zur Linie.

Bei der Wahl der Schwellenwerte High und Low ist zu beachten, dass die dritte Richtungsableitung von der Amplitude und Breite der Linie, sowie von der Größe des Glättungsparameters Sigma abhängt. Dabei ist die Abhängigkeit von der Amplitude der Linie linear, d.h., je größer die Amplitude, desto größer die dritte Ableitung. Für die Breite der Linie besteht ein umgekehrter Zusammenhang: Je breiter die Linie ist, desto kleiner ist die dritte Ableitung. Analoges gilt für die Abhängigkeit von Sigma: Je größer Sigma gewählt wird, desto kleiner wird die dritte Ableitung. Das bedeutet, dass für starke Glättung entsprechend kleinere Werte für High und Low gewählt werden müssen.

Die extrahierten Linien werden in einer topologisch „sauberen“ Struktur in Lines zurückgeliefert. Das bedeutet, dass die Linien an Kreuzungspunkten sauber aufgetrennt werden.

lines_color definiert folgende Attribute für jeden Linienpunkt, falls ExtractWidth auf *'false'* gesetzt wurde:

'angle'	Der Winkel der Richtung senkrecht zur Linie (so orientiert, dass die Normalenvektoren auf die rechte Seite der Linie
'response'	Die Größe der dritten Ableitung

Falls ExtractWidth auf *'true'* gesetzt wurde, werden zusätzlich noch folgende Attribute definiert:

'width_left'	Die Linienbreite links von der Linie
'width_right'	Die Linienbreite rechts von der Linie

Diese Attribute können mit get_contour_attrib_xld ausgelesen werden. Weitere Informationen zu Konturattributen können der Referenz von get_contour_attrib_xld entnommen werden.

Achtung

Im allgemeinen, aber besonders, wenn die Linienbreite extrahiert werden soll, sollte Sigma $\geq w/\sqrt{3}$ gewählt werden, wobei w die Breite (der halbe Durchmesser) der zu extrahierenden Linien im Bild ist. Als unterste Grenze *muss* Sigma $\geq w/2.5$ gewählt werden. Falls z.B. Linien mit einer Breite von 4 Pixel (Durchmesser 8 Pixel) extrahiert werden sollen, sollte Sigma ≥ 2.3 gewählt werden. Falls erwartet wird, dass in mindestens einem der Kanäle des Eingabebildes auch Treppenstufenlinien vorkommen und diese auch extrahiert werden sollen, sollte zusätzlich noch Sigma $\leq w$ gewählt werden. Dies muss beachtet werden, da Treppenstufenlinien bei zu großer Glättung zu normalen Stufenkanten werden und somit nicht mehr als dunkle Linien im Amplitudenbild des Farbkantenfilters erscheinen.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **Lines** (output_object) ..... xld_cont-array ⇝ *object*
  Extrahierte Linien.
- ▷ **Sigma** (input_control) ..... number ⇝ *real* / integer
  Stärke der Gaußschen Glättung.
  Defaultwert: 1.5
  Wertevorschläge: Sigma ∈ {1, 1.2, 1.5, 1.8, 2, 2.5, 3, 4, 5}
  Typischer Wertebereich: $0.7 \leq$ Sigma ≤ 20
  Empfohlene Schrittweite: 0.1
- ▷ **Low** (input_control) ..... number ⇝ *real* / integer
  Untere Schwelle für Hysterese-Schwellenwertoperation.
  Defaultwert: 3
  Wertevorschläge: Low ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10}
  Typischer Wertebereich: $0 \leq$ Low
  Empfohlene Schrittweite: 0.5
  Restriktion: Low >= 0

▷ **High** (input_control) . . . . . number ⇝ *real* / integer
Obere Schwelle für Hysterese-Schwellenwertoperation.
Defaultwert: 8
Wertevorschläge: High ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25}
Typischer Wertebereich: 0 ≤ High
Empfohlene Schrittweite: 0.5
Restriktion: High >= 0 && High >= Low

▷ **ExtractWidth** (input_control) . . . . . string ⇝ *string*
Soll die Linienbreite extrahiert werden?
Defaultwert: 'true'
Werteliste: ExtractWidth ∈ {'true', 'false'}

▷ **CompleteJunctions** (input_control) . . . . . string ⇝ *string*
Sollen die Kreuzungspunkte vervollständigt werden?
Defaultwert: 'true'
Werteliste: CompleteJunctions ∈ {'true', 'false'}

Komplexität

Der Bedarf an temporären Speicher ist abhängig von der Höhe H der Region von Image.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert lines_color den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

gen_polygons_xld

Alternativen

lines_gauss, lines_facet

Siehe auch

edges_color, edges_color_sub_pix

Literatur

C. Steger: „Subpixel-Precise Extraction of Lines and Edges"; International Archives of Photogrammetry and Remote Sensing, vol. XXXIII, part B3; pp. 141-156; 2000.

C. Steger: „An Unbiased Detector of Curvilinear Structures"; IEEE Transactions on Pattern Analysis and Machine Intelligence; vol. 20, no. 2; pp. 113-125; 1998.

C. Steger: „Unbiased Extraction of Curvilinear Structures from 2D and 3D Images"; Herbert Utz Verlag, München; 1998.

Modul

2D Metrology

lines_facet (Image : Lines : MaskSize, Low, High, LightDark :)

Erkennen von Linien mit dem Facet-Modell.

Mit lines_facet können Linien (gekrümmt-lineare Strukturen) aus einem Bild Image extrahiert werden. Die extrahierten Linien werden in Lines als subpixel-genaue XLD-Konturen zurückgegeben. Der Parameter LightDark bestimmt, ob helle oder dunkle Linien extrahiert werden sollen.

Zur Extraktion werden in jedem Punkt des Bildes unter Verwendung des Facet-Modells die Parameter eines quadratischen Polynoms in x und y als Least-Squares-Anpassung an die Bilddaten berechnet. Der Parameter MaskSize bestimmt dabei die Größe des Bereichs, über den angepasst wird. Große Werte für MaskSize sorgen für eine

stärkere Glättung der Eingabedaten, können aber zu ungenauerer Lokalisation der Linien und zu stark schwingenden Linien führen. Mit Hilfe der Parameter des Polynoms wird in jedem Bildpunkt die Linienrichtung bestimmt. Bildpunkte, die ein lokales Maximum in der zweiten Richtungsableitung senkrecht zur Linienrichtung besitzen, werden als Linienpunkte markiert. Die gefundenen Linienpunkte werden hierauf zu Konturen zusammengefasst. Dabei werden Punkte, die eine zweite Richtungsableitung, die größer als `High` ist, sofort als sichere Linienpunkte akzeptiert. Punkte, deren zweite Ableitung kleiner als `Low` ist, werden sofort verworfen. Alle Punkte, die eine zweite Ableitung zwischen diesen zwei Werten besitzen, werden akzeptiert, wenn sie durch einen Pfad mit sicheren Punkten verbunden sind. Dies ist ähnlich zu einer Hysterese-Schwellenwert-Operation mit unbegrenzter Pfadlänge (siehe `hysteresis_threshold`). Allerdings wird diese Funktion nicht benutzt, da sonst keine subpixel-genaue Extraktion möglich wäre.

Für die Wahl der Schwellenwerte gilt sinngemäß das bei der Beschreibung von `lines_gauss` gesagte. Ein Wert von Sigma = 1.5 dort entspricht in etwa einer Maskengröße `MaskSize` von 5 hier.

Die extrahierten Linien werden in einer topologisch „sauberen“ Struktur in `Lines` zurückgeliefert. Das bedeutet, dass die Linien an Kreuzungspunkten sauber aufgetrennt werden.

`lines_facet` definiert folgende Attribute für jeden Linienpunkt:

'angle'	Der Winkel der Richtung senkrecht zur Linie
'response'	Die Größe der zweiten Ableitung

Diese Attribute können mit `get_contour_attrib_xld` ausgelesen werden. Weitere Informationen zu Konturattributen können der Referenz von `get_contour_attrib_xld` entnommen werden.

Achtung

Je kleiner der Einzugsbereich `MaskSize` des Operators gewählt wird, desto mehr kurze, zerstückelte Linien werden gefunden. Dadurch kann die Verarbeitungszeit beträchtlich ansteigen.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
  Eingabebild.
- ▷ **`Lines`** (output_object) ..... xld_cont-array ⇝ *object*
  Extrahierte Linien.
- ▷ **`MaskSize`** (input_control) ..... integer ⇝ *integer*
  Größe der zu verwendenden Maske zur Bestimmung der Parameter des Facet-Modells.
  Defaultwert: 5
  Werteliste: `MaskSize` ∈ {3, 5, 7, 9, 11}
- ▷ **`Low`** (input_control) ..... number ⇝ *real* / integer
  Untere Schwelle für Hysterese-Schwellenwertoperation.
  Defaultwert: 3
  Wertevorschläge: `Low` ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10}
  Typischer Wertebereich: 0 ≤ `Low` ≤ 20
  Empfohlene Schrittweite: 0.5
  Restriktion: Low >= 0
- ▷ **`High`** (input_control) ..... number ⇝ *real* / integer
  Obere Schwelle für Hysterese-Schwellenwertoperation.
  Defaultwert: 8
  Wertevorschläge: `High` ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25}
  Typischer Wertebereich: 0 ≤ `High` ≤ 35
  Empfohlene Schrittweite: 0.5
  Restriktion: High >= 0 && High >= Low
- ▷ **`LightDark`** (input_control) ..... string ⇝ *string*
  Helle oder dunkle Linien extrahieren.
  Defaultwert: 'light'
  Werteliste: `LightDark` ∈ {'dark', 'light'}

Beispiel

```
* Detection of lines in an aerial image
read_image(Image,'mreut4_3')
lines_facet(Image,Lines,5,3,8,'light')
dev_display(Lines)
```

Komplexität

Sei A die Anzahl von Pixeln in der Region von `Image`. Dann ist die Laufzeitkomplexität $O(A * MaskSize)$.

Der Bedarf an temporären Speicher ist abhängig von der Höhe H der Region von `Image` und der Breite W von `Image`. Sei $S = W * H$, dann benötigt `lines_facet` mindestens $55 * S$ Bytes an temporärem Speicher bei der Ausführung.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `lines_facet` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`gen_polygons_xld`

Alternativen

`lines_gauss`

Siehe auch

`bandpass_image`, `dyn_threshold`, `topographic_sketch`

Literatur

A. Busch: „Fast Recognition of Lines in Digital Images Without User-Supplied Parameters“. In H. Ebner, C. Heipke, K.Eder, eds., „Spatial Information from Digital Photogrammetry and Computer Vision“, International Archives of Photogrammetry and Remote Sensing, Vol. 30, Part 3/1, pp. 91-97, 1994.

Modul

2D Metrology

lines_gauss (Image : Lines : Sigma, Low, High, LightDark, ExtractWidth, LineModel, CompleteJunctions :)

Erkennen von Linien und deren Breite.

Mit `lines_gauss` können Linien (gekrümmt-lineare Strukturen) aus einem Bild `Image` extrahiert werden. Die extrahierten Linien werden in `Lines` als subpixel-genaue XLD-Konturen zurückgegeben. Der Parameter `LightDark` bestimmt, ob helle oder dunkle Linien extrahiert werden sollen. Wenn `ExtractWidth` auf *'true'* gesetzt wird, so wird für jeden Linienpunkt die Linienbreite extrahiert. Wenn `LineModel` auf einen Wert ungleich *'none'* gesetzt wird, kompensiert `lines_gauss` die Effekte von asymmetrischen Linien (Linien mit unterschiedlichem Kontrast auf beiden Seiten der Linie), und korrigiert die Position und Breite der Linie. Für das zur Korrektur verwendete Linienmodell kann zwischen mit `LineModel` zwischen balkenförmigen Linien (`LineModel` = *'bar-shaped'*), parabolischen Linien (`LineModel` = *'parabolic'*) und Gaußschen Linien (`LineModel` = *'gaussian'*) ausgewählt werden. Balkenförmige Linien sind für die meisten Anwendungen die richtige Wahl. Für die Extraktion von röhrenförmigen Objekten im Durchlicht (z.B. Blutgefäße in Röntgenaufnahmen) bieten sich die anderen zwei Modi an. Dabei sollte das parabolische Linienmodell für Bilder verwendet werden, in denen die Linien sehr scharf abgebildet sind, und das Gaußsche Linienmodell für Bilder, in denen die Linien weniger scharf abgebildet sind. Der Parameter `LineModel` wird nur beachtet, falls `ExtractWidth`=*'true'*. Da der Linienextraktor aus differentialgeometrischen Gründen bestimmte Kreuzungspunkte nicht extrahieren kann, wird versucht, diese mit anderen Mitteln zu extrahieren, falls `CompleteJunctions`=*'true'* gesetzt wird.

Zur Extraktion werden in jedem Punkt des Bildes unter Verwendung der partiellen Ableitungen einer Gaußschen Glättungsmaske die Parameter eines quadratischen Polynoms in x und y berechnet. Der Parameter `Sigma` bestimmt dabei die Stärke der Glättung. Große Werte für `Sigma` sorgen für eine stärkere Glättung der Eingabedaten, können aber zu ungenauerer Lokalisation der Linien führen. Im Gegensatz zu `lines_facet` ist aber kein Schwingen der Linien für große Glättung zu beobachten. Ein weiterer Vorteil gegenüber `lines_facet` ist die genauere Lokalisation der Linien. Mit Hilfe der Parameter des Polynoms wird in jedem Bildpunkt die Linienrichtung bestimmt. Bildpunkte, die ein lokales Maximum in der zweiten Richtungsableitung senkrecht zur Linienrichtung besitzen, werden als Linienpunkte markiert. Die gefundenen Linienpunkte werden hierauf zu Konturen zusammengefasst. Dabei werden Punkte, die eine zweite Richtungsableitung, die größer als `High` ist, sofort als sichere Linienpunkte akzeptiert. Punkte, deren zweite Ableitung kleiner als `Low` ist, werden sofort verworfen. Alle Punkte, die eine zweite Ableitung zwischen diesen zwei Werten besitzen, werden akzeptiert, wenn sie durch einen Pfad mit sicheren Punkten verbunden sind. Dies ist ähnlich zu einer Hysterese-Schwellenwert-Operation mit unbegrenzter Pfadlänge (siehe `hysteresis_threshold`). Allerdings wird diese Funktion nicht benutzt, da sonst keine subpixel-genaue Extraktion möglich wäre.

Bei der Wahl der Schwellenwerte `High` und `Low` ist zu beachten, dass die zweite Richtungsableitung von der Amplitude und Breite der Linie, sowie von der Größe des Glättungsparameters `Sigma` abhängt. Dabei ist die Abhängigkeit von der Amplitude der Linie linear, d.h. je größer die Amplitude, desto größer die zweite Ableitung. Für die Breite der Linie besteht ein näherungsweise exponentieller Zusammenhang: Je breiter die Linie ist, desto kleiner ist die zweite Ableitung. Analoges gilt für die Abhängigkeit von `Sigma`: Je größer `Sigma` gewählt wird, desto kleiner wird die zweite Ableitung. Das bedeutet, dass für starke Glättung entsprechend kleinere Werte für `High` und `Low` gewählt werden müssen. Zwei Beispiele sollen dies verdeutlichen: Wenn im Bild 5 Pixel breite Linien mit einer Amplitude größer als 100 und einer Glättung `Sigma` = 1.5 extrahiert werden sollen, so sollte `High` größer als 14 gewählt werden. Wenn dagegen 10 Pixel breite Linien mit einer Amplitude größer als 100 und einer Glättung `Sigma` = 3 gefunden werden sollen, so sollte `High` größer als 3.5 gesetzt werden. Als Werte für `Low` empfehlen sich Werte zwischen 0.25 `High` und 0.5 `High`.

Die Parameter `Low` und `High` können aus den entsprechende Grauwertkontrastwerten der zu extrahierenden Linien (ContrastLow und ContrastHigh) und dem gewählten Wert für `Sigma` wie folgt berechnet werden:

$$\begin{pmatrix} \texttt{Low} \\ \texttt{High} \end{pmatrix} = -2 \cdot \begin{pmatrix} ContrastLow \\ ContrastHigh \end{pmatrix} \cdot \frac{w}{\sqrt{2\pi} \cdot \texttt{Sigma}^3} \cdot e^{-\frac{w^2}{2 \cdot \texttt{Sigma}^2}}$$

Die extrahierten Linien werden in einer topologisch sauberen Struktur in `Lines` zurückgeliefert. Das bedeutet, dass die Linien an Kreuzungspunkten sauber aufgetrennt werden.

`lines_gauss` definiert folgende Attribute für jeden Linienpunkt, falls `ExtractWidth` auf *'false'* gesetzt wurde:

'angle': Der Winkel der Richtung senkrecht zur Linie

'response': Die Größe der zweiten Ableitung

Falls `ExtractWidth` auf *'true'* gesetzt wurde, werden zusätzlich zu *'angle'* und *'response'* noch folgende Attribute definiert:

'width_left': Die Linienbreite links von der Linie

'width_right': Die Linienbreite rechts von der Linie

Falls `ExtractWidth` auf *'true'* und `LineModel` auf einen Wert ungleich *'none'* gesetzt wurde, werden zusätzlich zu *'angle'*, *'response'*, *'width_left'* und *'width_right'* noch folgende Attribute definiert:

'asymmetry': Die Asymmetrie des Linienpunkts

'contrast': Der Kontrast des Linienpunkts

Dabei ist der Wert der Asymmetrie so gewählt, dass er positive Werte annimmt, wenn die Asymmetrie, d.h. der kleinere Gradient, auf der rechten Seite der Linie liegt, und dass er negative Werte annimmt, wenn die Asymmetrie auf der linken Seite der Linie liegt.

Der Kontrast ergibt sich aus der Differenz zwischen dem Grauwert der Linie und dem Grauwert des Hintergrundes. Der Wert für den Kontrast ist positiv, wenn helle Linien extrahiert werden und negativ, wenn dunkle Linien extrahiert werden. Der zurückgegebene Kontrast kann größere Werte annehmen als der maximal darstellbare Grauwert des Eingabebildes, insbesondere dann, wenn das durch `LineModel` festgelegte Linienmodell nicht im Bild vorliegt. So kann der Kontrast beispielsweise bei byte-Bildern größer als 255 sein.

Alle Attribute können mit `get_contour_attrib_xld` ausgelesen werden. Weitere Informationen zu Konturattributen können der Referenz von `get_contour_attrib_xld` entnommen werden.

`lines_gauss` kann auf OpenCL-Geräten ausgeführt werden.

Achtung

Im allgemeinen, aber besonders, wenn die Linienbreite extrahiert werden soll, sollte `Sigma` $\geq w/\sqrt{3}$ gewählt werden, wobei w die Breite (der halbe Durchmesser) der zu extrahierenden Linien im Bild ist. Als unterste Grenze *muss* `Sigma` $\geq w/2.5$ gewählt werden. Falls z.B. Linien mit einer Breite von 4 Pixel (Durchmesser 8 Pixel) extrahiert werden sollen, sollte `Sigma` ≥ 2.3 gewählt werden. Wird `Sigma` zu klein gewählt, so werden die Attribute *'width_left'*, *'width_right'*, *'asymmetry'* und *'contrast'* auf Null gesetzt.

`lines_gauss` wird durch SSE2 Instruktionen beschleunigt, falls der Systemparameter *'sse2_enable'* auf *'true'* gesetzt ist (dies ist Standard, sofern SSE2 auf Ihrem Rechner verfügbar ist). Diese Implementierung ist aufgrund numerischer Probleme im Vergleich zur reinen C Version etwas ungenauer. Falls für Ihre Anwendung Genauigkeit wichtiger ist als Geschwindigkeit können Sie *'sse2_enable'* auf *'false'* setzen, bevor Sie `lines_gauss` aufrufen. Dadurch verwendet `lines_gauss` keine SSE2 Beschleunigungen. Vergessen Sie nicht *'sse2_enable'* danach wieder auf *'true'* zu setzen.

Wird `lines_gauss` auf OpenCL-Geräten ausgeführt, gelten die selben Einschränkungen wie für `derivate_gauss`: `Sigma` muss so gewählt werden, dass die resultierende Filtermaske kleiner als 129 Pixel groß ist. Es ist weiterhin zu beachten, dass das Ergebnis von dem der CPU- Implementierung abweichen kann.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
  Eingabebild.
- ▷ **`Lines`** (output_object) ..... xld_cont-array ⇝ *object*
  Extrahierte Linien.
- ▷ **`Sigma`** (input_control) ..... number ⇝ *real* / integer
  Stärke der Gaußschen Glättung.
  Defaultwert: 1.5
  Wertevorschläge: `Sigma` ∈ {1, 1.2, 1.5, 1.8, 2, 2.5, 3, 4, 5}
  Typischer Wertebereich: 0.7 ≤ `Sigma` ≤ 20
  Empfohlene Schrittweite: 0.1
- ▷ **`Low`** (input_control) ..... number ⇝ *real* / integer
  Untere Schwelle für Hysterese-Schwellenwertoperation.
  Defaultwert: 3
  Wertevorschläge: `Low` ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10}
  Typischer Wertebereich: 0 ≤ `Low` ≤ 20
  Empfohlene Schrittweite: 0.5
  Restriktion: Low >= 0
- ▷ **`High`** (input_control) ..... number ⇝ *real* / integer
  Obere Schwelle für Hysterese-Schwellenwertoperation.
  Defaultwert: 8
  Wertevorschläge: `High` ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25}
  Typischer Wertebereich: 0 ≤ `High` ≤ 35
  Empfohlene Schrittweite: 0.5
  Restriktion: High >= 0 && High >= Low
- ▷ **`LightDark`** (input_control) ..... string ⇝ *string*
  Helle oder dunkle Linien extrahieren.
  Defaultwert: 'light'
  Werteliste: `LightDark` ∈ {'dark', 'light'}
- ▷ **`ExtractWidth`** (input_control) ..... string ⇝ *string*
  Soll die Linienbreite extrahiert werden?
  Defaultwert: 'true'
  Werteliste: `ExtractWidth` ∈ {'true', 'false'}

▷ **LineModel** (input_control) ..... string ⇝ *string*
Linienmodell, das zur Korrektur der Linienposition und -breite verwendet wird.
Defaultwert: 'bar-shaped'
Werteliste: `LineModel` ∈ {'none', 'bar-shaped', 'parabolic', 'gaussian'}

▷ **CompleteJunctions** (input_control) ..... string ⇝ *string*
Sollen die Kreuzungspunkte vervollständigt werden?
Defaultwert: 'true'
Werteliste: `CompleteJunctions` ∈ {'true', 'false'}

Beispiel

```
* Detection of lines in an aerial image
read_image(Image,'mreut4_3')
lines_gauss(Image,Lines,1.5,3,8,'light','true','bar-shaped','true')
dev_display(Lines)
```

Komplexität

Sei A die Anzahl von Pixeln in der Region von `Image`. Dann ist die Laufzeitkomplexität $O(A * Sigma)$.

Der Bedarf an temporären Speicher ist abhängig von der Höhe H der Region von `Image` und der Breite W von `Image`. Sei $S = W * H$, dann benötigt `lines_gauss` mindestens $55 * S$ Bytes an temporärem Speicher bei der Ausführung.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `lines_gauss` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`gen_polygons_xld`

Alternativen

`lines_facet`

Siehe auch

`bandpass_image`, `dyn_threshold`, `topographic_sketch`

Literatur

C. Steger: „Extracting Curvilinear Structures: A Differential Geometric Approach“. In B. Buxton, R. Cipolla, eds., „Fourth European Conference on Computer Vision“, Lecture Notes in Computer Science, Volume 1064, Springer Verlag, pp. 630-641, 1996.

C. Steger: „Extraction of Curved Lines from Images“. In „13th International Conference on Pattern Recognition“, Volume II, pp. 251-255, 1996.

C. Steger: „An Unbiased Detector of Curvilinear Structures“. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 2, pp. 113-125, 1998.

C. Steger: „Unbiased extraction of lines with parabolic and Gaussian profiles“. Computer Vision and Image Understanding, vol. 117, no. 2, pp. 97-112, 2013.

Modul

2D Metrology

10.10 Match

exhaustive_match (Image, RegionOfInterest,
ImageTemplate : ImageMatch : Mode :)

Matching zwischen einem Template und einem Bild.

exhaustive_match führt ein Matching zwischen ImageTemplate und Image innerhalb des Bildbereichs RegionOfInterest durch. Dabei wird ImageTemplate über alle Punkte von Image, die innerhalb der RegionOfInterest liegen, geschoben. Abhängig vom Parameter Mode wird ein Matching-Kriterium berechnet. Die Ergebniswerte werden in ImageMatch abgelegt.

Folgende Matching-Kriterien (Mode) stehen zur Verfügung:

'norm_correlation'

$$\texttt{ImageMatch}[i][j] = 255 \cdot \frac{\sum_{u,v} (\texttt{Image}[i-u][j-v] \cdot \texttt{ImageTemplate}[l-u][c-v])}{\sqrt{\sum_{u,v} (\texttt{Image}[i-u][j-v]^2) \cdot \sum_{u,v} (\texttt{ImageTemplate}[l-u][c-v]^2)}}$$

wobei $X[i][j]$ den Grauwert in der i-ten Zeile und j-ten Spalte des Bildes X bezeichnet. (l, c) ist der Schwerpunkt der Region von ImageTemplate. u und v werden so gewählt, dass alle Punkte des Templates erreicht werden, i, j laufen über die RegionOfInterest. An den Bildrändern werden nur die Teile von ImageTemplate berücksichtigt, die innerhalb des Bildes liegen (d.h. u und v werden entsprechend eingeschränkt). Werden hohe Werte in ImageMatch zurückgegeben ist die Übereinstimmung hoch.

Wertebereich: [*0 ... 255* (beste Übereinstimmung)].

'dfd' Berechnung der mittleren „displaced frame difference“:

$$\texttt{ImageMatch}[i][j] = \frac{\sum_{u,v} |\texttt{Image}[i-u][j-v] - \texttt{ImageTemplate}[l-u][c-v]|}{AREA(ImageTemplate)}$$

Bezeichnungen wie bei *'norm_correlation'*. $AREA(X)$ steht für die Fläche der Region X. Werden niedrige Werte in ImageMatch zurückgegeben ist die Übereinstimmung hoch.

Wertebereich: [*0* (beste Übereinstimmung) *... 255*].

Sowohl die normierte Korrelation, als auch die „displaced frame difference“ sind in ihrer Berechnung (abhängig von der Fläche des ImageTemplate) sehr aufwendig. Es ist daher wichtig, die Eingaberegion (RegionOfInterest) möglichst einzugrenzen, d.h. den Filter nur in einer möglichst eng umrissenen „region of interest“ anzuwenden.

Qualitativ liefern beide Modi vergleichbare Ergebnisse. Der Modus *'dfd'* ist jedoch um etwa Faktor 3.5 schneller.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild.
- ▷ **RegionOfInterest** (input_object) ..... region ⇝ *object*
  Suchbereich im Eingabebild.
- ▷ **ImageTemplate** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Dieser Bildbereich wird innerhalb von RegionOfInterest mit Image „gematcht“.
- ▷ **ImageMatch** (output_object) ..... image(-array) ⇝ *object :* byte
  Ergebnisbild: Werte des Matching-Kriteriums.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Gewünschtes Matching-Kriterium.
  Defaultwert: 'dfd'
  Werteliste: Mode ∈ {'norm_correlation', 'dfd'}

Beispiel

```
read_image(Image,'monkey')
dev_display(Image)
draw_rectangle2(WindowHandle,Row,Column,Phi,Length1,Length2)
```

```
gen_rectangle2(Rectangle,Row,Column,Phi,Length1,Length2)
reduce_domain(Image,Rectangle,Template)
exhaustive_match(Image,Image,Template,ImageMatch,'dfd')
invert_image(ImageMatch,ImageInvert)
local_max(Image,Maxima)
union1(Maxima,AllMaxima)
add_channels(AllMaxima,ImageInvert,FitMaxima)
threshold(FitMaxima,BestFit,230.0,255.0)
dev_display(BestFit)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert exhaustive_match den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

draw_region, draw_rectangle1

Nachfolger

local_max, threshold

Alternativen

exhaustive_match_mg

Modul

Foundation

exhaustive_match_mg (Image, ImageTemplate : ImageMatch : Mode, Level, Threshold :)

Matching zwischen einem Template und einem Bild in einer Auflösungspyramide.

exhaustive_match_mg ist eine Ergänzung zum Operator exhaustive_match, die ein Matching zwischen dem Bild Image und dem Template ImageTemplate durchführt. Dabei wird ImageTemplate über alle Punkte der Region von Image geschoben, abhängig vom Parameter Mode ein Matching-Kriterium berechnet und die Ergebniswerte in ImageMatch abgelegt.

In der Regel interessieren von solcherart gefilterten Bildern aber nur die Bereiche mit guten Matchingergebnissen. Die Größe des Suchbereichs, also die Region des Eingabebildes Image, bestimmt maßgeblich das Laufzeitverhalten des Matching-Filters. Daher wird bei exhaustive_match_mg mit reduzierter Bildauflösung zunächst eine „region of interest“ bestimmt, in der mit guten Matching-Ergebnissen zu rechnen ist, und im Anschluss daran nur innerhalb dieser Teilregion in normaler Auflösung das eigentliche Matching (vgl. exhaustive_match) durchgeführt. Dazu werden die Gauß-Pyramiden von Image und ImageTemplate aufgebaut (insbesondere werden auch die entsprechenden Regionen mittransformiert). Dann wird auf jeder Ebene der Auflösungspyramiden - beginnend mit der Startebene Level - das Matching innerhalb der aktuellen „region of interest“ ausgeführt. Dabei ist die „region of interest“ im Startlevel gleich der Region des Eingabebildes Image. Nach der Filterung wird mittels einer Schwellenwert-Operation eine neue „region of interest“ bestimmt und auf die nächste Auflösungsstufe transformiert:

threshold(..0,Threshold..), falls Mode = ’dfd’
threshold(..Threshold,255..), falls Mode = ’norm_correlation’

In höchster Auflösung (Level 0) wird dann schließlich das abschließende Matching in der ermittelten „region of interest“ berechnet. Das Ausgabebild ImageMatch enthält das entsprechende Filterergebnis und als Region die endgültige „region of interest“, die mit einer Schwellenwert-Operation auf dem Ergebnisbild bestimmt wird.

exhaustive_match_mg ist daher kein reiner Filter, sondern ist auch zur Klasse der Regionentransformationen zu zählen.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
Eingabebild.

▷ **ImageTemplate** (input_object) ..... singlechannelimage ⇝ *object :* byte
Der Definitionsbereich dieses Bildes wird mit Image „gematcht“.

▷ **ImageMatch** (output_object) ..... image(-array) ⇝ *object :* byte
Ergebnisbild und -region: Werte des Matching-Kriteriums innerhalb der ermittelten „region of interest“.
Parameteranzahl: ImageMatch == Image

▷ **Mode** (input_control) ..... string ⇝ *string*
Gewünschtes Matching-Kriterium.
Defaultwert: ’dfd’
Werteliste: Mode ∈ {’norm_correlation’, ’dfd’}

▷ **Level** (input_control) ..... integer ⇝ *integer*
Startlevel in der Auflösungspyramide (höchste Auflösung: Level 0).
Defaultwert: 1
Wertevorschläge: Level ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8}
Restriktion: Level <= ld(width(Image)) && Level <= ld(height(Image)) && Level <= ld(width(ImageTemplate)) && Level <= ld(height(ImageTemplate))

▷ **Threshold** (input_control) ..... integer ⇝ *integer*
Schwelle zur Bestimmung der „region of interest“.
Defaultwert: 30
Wertevorschläge: Threshold ∈ {5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190, 195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250}
Typischer Wertebereich: 0 ≤ Threshold ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5

Beispiel

```
read_image(&Image,"monkey");
disp_image(Image,WindowHandle);
draw_rectangle2(WindowHandle,&Row,&Column,&Phi,&Length1,&Length2);
gen_rectangle2(&Rectangle,Row,Column,Phi,Length1,Length2);
reduce_domain(Image,Rectangle,&Template);
exhaustive_match_mg(Image,Template,&ImageMatch,'dfd'1,30);
invert_image(ImageMatch,&ImageInvert);
local_max(ImageInvert,&BestFit);
disp_region(BestFit,WindowHandle);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert exhaustive_match_mg den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system (’no_object_result’,<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

draw_region, draw_rectangle1

Nachfolger

threshold, local_max

Alternativen

exhaustive_match

Siehe auch

gen_gauss_pyramid

Modul

Foundation

gen_gauss_pyramid (Image : ImagePyramid : Mode, Scale :)

Berechnung einer Gauß-Pyramide.

gen_gauss_pyramid berechnet eine Pyramide von verkleinerten Bildern. Das Maß, um das jedes Nachfolgerbild verkleinert wird, gibt der Parameter Scale an. Beispielsweise wird mit einem Wert von 0.5 für Scale die Kantenlänge von Image auf die Hälfte verkürzt. Dies entspricht gerade der „normalen“ Pyramide.

Der Parameter Mode gibt die Art der Mittelung an. Eine Beschreibung hierzu ist bei affine_trans_image zu finden. Für den Fall, dass Scale gleich 0.5 ist, stehen zusätzlich die Werte *'min'* und *'max'* zu Verfügung. Hierbei wird das Minimum bzw. das Maximum der vier Punkte verwendet.

Es ist zu beachten, dass jede Ebene als ein eigenes Bild, d.h als eigenständiges Bildobjekt mit einer Matrix und eigenem Definitionsbereich ausgegeben wird. Einzelne oder mehrere Ebenen können mit select_obj bzw. copy_obj selektiert werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **ImagePyramid** (output_object) ..... (multichannel-)image-array ⇝ *object :* byte / uint2 / real
  Ausgabebilder.
  Parameteranzahl: ImagePyramid > Image
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Art der Filtermaske.
  Defaultwert: 'weighted'
  Werteliste: Mode ∈ {'nearest_neighbor', 'constant', 'weighted', 'min', 'max'}
- ▷ **Scale** (input_control) ..... real ⇝ *real*
  Verkleinerungsfaktor.
  Defaultwert: 0.5
  Wertevorschläge: Scale ∈ {0.2, 0.3, 0.4, 0.5, 0.6}
  Typischer Wertebereich: 0.1 ≤ Scale ≤ 0.9
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: 0.1 < Scale && Scale < 0.9

Beispiel

```
gen_gauss_pyramid(Image,Pyramid,"weighted",0.5);
count_obj(Pyramid,&num);
for (i=1; i<=num; i++)
{
  select_obj(Pyramid,&Single,i);
  disp_image(Single,WindowHandle);
  clear(Single);
}
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Kanalebene.

Nachfolger

image_to_channels, count_obj, select_obj, copy_obj

Alternativen

zoom_image_size, zoom_image_factor

Siehe auch

affine_trans_image

Modul

Foundation

monotony (Image : ImageMonotony : :)

Berechnung des Monotonieoperators.

monotony berechnet den Monotonieoperator. Hierzu wird die Anzahl der Punkte in der 8-Nachbarschaft gezählt, die echt kleiner als der aktuelle Grauwert sind. Diese Anzahl wird im Ausgabebild eingetragen.

Bei einem echten Maximum erhält man den Wert 8, bei einem Minimum oder auf einem Plateau ergibt sich der Wert 0. Bei einem Grat oder einem Hang erhält man entsprechende Zwischenwerte.

Um lokale Minima zu untersuchen, muss das Eingabebild vorher nur mit dem Befehl invert_image invertiert werden.

Der Monotonieoperator wird häufig als Vorbereitung für Matching-Verfahren eingesetzt, da er invariant gegen Helligkeitsänderungen ist.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
Eingabebild.

▷ **ImageMonotony** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
Ergebnis des Monotonieoperators.
Parameteranzahl: ImageMonotony == Image

Beispiel

```
/* searching the strict maximums */
gauss_filter(Image,&Gauss,5);
monotony(Gauss,&Monotony);
threshold(Monotony,Maxima,8.0,8.0);
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

binomial_filter, gauss_filter, median_image, mean_image, smooth_image, invert_image

Nachfolger

threshold, exhaustive_match, disp_image

Alternativen

local_max, topographic_sketch, corner_response

Modul

Foundation

10.11 Optischer Fluss

derive_vector_field (VectorField : Result : Sigma, Component :)

Falten eines Vektorfeldes mit Hilfe von Ableitungen der Gaußfunktion.

derive_vector_field glättet die Komponenten eines Vektorfeldes mit Hilfe von Ableitungen der Gaußfunktion und berechnet verschiedene daraus abgeleitete Größen. derive_vector_field akzeptiert nur Vektorfelder die den semantischen Typ 'vector_field_relative' haben. Das Vektorfeld VectorField $F(r,c) = (u(r,c), v(r,c))$ ist wie in optical_flow_mg definiert. Dabei ist Sigma der Parameter der Gaußfunktion (d.h. es bestimmt die Glättung). Falls in Sigma ein einzelner Wert angegeben wird, ist die Glättung in Spalten- und Zeilenrichtung gleich. Falls zwei Werte in Sigma übergeben werden, wird mit dem ersten Wert die Glättung in Spaltenrichtung und mit dem zweiten Wert die Glättung in Zeilenrichtung festgelegt. Mögliche Werte für Component sind:

'curl': Die Rotation des Vektorfeldes. Eine Anwendung von *'curl'* ist es, optische Flussfelder zu analysieren. Bildlich gesprochen steht *'curl'* für die Rotation, die ein kleines Boot hätte, falls das Vektorfeld eine Flüssigkeit wäre.

$$curl = \frac{\partial u(r,c)}{\partial c} - \frac{\partial v(r,c)}{\partial r}$$

'divergence': Die Divergenz des Vektorfeldes. Eine Anwendung von *'divergence'* ist es, optische Flussfelder zu analysieren. Bildlich gesprochen steht *'divergence'* für die Senken und Quellen die das Vektorfeld hätte, wenn es eine Flüssigkeit wäre.

$$div = \frac{\partial u(r,c)}{\partial r} + \frac{\partial v(r,c)}{\partial c}$$

Wenn derive_vector_field im Kontext von photometrischem Stereo verwendet wird, bietet der Operator zwei weitere Parameter, die insbesondere für die Verarbeitung des Gradientenfeldes, das von photometric_stereo geliefert wird, geeignet sind. In diesem Fall wird das Vektorfeld als Gradientenfeld der Oberfläche interpretiert.

Im Folgenden wird das Eingabevektorfeld als $G(r,c) = \nabla f = (\frac{\partial f(r,c)}{\partial r}, \frac{\partial f(r,c)}{\partial c})$

angenommen, wobei die erste und zweite Komponente des Eingabegradientenfeldes das Gradientenfeld der Oberfläche $f(r,c)$ ist. In den folgenden Formeln steht f_rc für die erste Ableitung in Spaltenrichtung der ersten Komponente des Gradientenfeldes.

'mean_curvature': Mittlere Krümmung H. Eine Anwendung von *'mean_curvature'* ist es, Vektorfelder zu verarbeiten, die von photometric_stereo zurückgegeben werden. Nach dem Filtern können kleine Kratzer oder Erhebungen leichter segmentiert werden.

$$\begin{aligned}
A &= (1 + \frac{\partial f(r,c)^2}{\partial r}) \frac{\partial^2 f(r,c)}{\partial c^2} \\
B &= \frac{\partial f(r,c)}{\partial r} \frac{\partial f(r,c)}{\partial c} (\frac{\partial^2 f(r,c)}{\partial r \partial c} + \frac{\partial^2 f(r,c)}{\partial c \partial r}) \\
C &= (1 + \frac{\partial f(r,c)^2}{\partial c}) \frac{\partial^2 f(r,c)}{\partial r^2} \\
D &= (1 + \frac{\partial f(r,c)^2}{\partial r} + \frac{\partial f(r,c)^2}{\partial c})^{\frac{3}{2}} \\
H &= \frac{A - B + C}{D}
\end{aligned}$$

'gauss_curvature': Gaußkrümmung K. Eine Anwendung von *'gauss_curvature'* ist es, Vektorfelder zu verarbeiten, die von `photometric_stereo` zurückgegeben werden. Nach dem Filtern können kleine Kratzer oder Erhebungen leichter segmentiert werden. Wenn die Oberfläche, die dem Vektorfeld zugrunde liegt, abwickelbar ist, ist die Gaußkrümmung gleich Null.

$$K = \frac{\frac{\partial f(r,c)^2}{\partial r \partial r} * \frac{\partial f(r,c)^2}{\partial c \partial c} - \frac{\partial f(r,c)^2}{\partial r \partial c} * \frac{\partial f(r,c)^2}{\partial c \partial r}}{(1 + \frac{\partial f(r,c)^2}{\partial r} + \frac{\partial f(r,c)^2}{\partial c})^2}$$

Parameter

- ▷ **`VectorField`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* vector_field
  Eingabevektorfeld.
- ▷ **`Result`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Gefilterte Bilder.
- ▷ **`Sigma`** (input_control) ..... real(-array) ⇝ *real*
  Sigma der Gaußfunktion.
  Defaultwert: 1.0
  Wertevorschläge: `Sigma` ∈ {0.7, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich: 0.2 ≤ `Sigma` ≤ 50.0
  Restriktion: 0.01 <= Sigma <= 50.0
- ▷ **`Component`** (input_control) ..... string ⇝ *string*
  Zu berechnende Ableitung oder zu berechnendes Merkmal.
  Defaultwert: 'mean_curvature'
  Werteliste: `Component` ∈ {'curl', 'divergence', 'mean_curvature', 'gauss_curvature'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `derivate_vector_field` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Domainebene.
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`optical_flow_mg`, `photometric_stereo`

Nachfolger

`threshold`

Modul

Foundation

`optical_flow_mg` `( ImageT1, ImageT2 : VectorField : Algorithm, SmoothingSigma, IntegrationSigma, FlowSmoothness, GradientConstancy, MGParamName, MGParamValue : )`

Berechnung des optischen Flusses zwischen zwei Bildern.

`optical_flow_mg` berechnet den optischen Fluss zwischen zwei Bildern. Der optische Fluss stellt die Bewegungsinformation zwischen zwei aufeinanderfolgenden Bildern einer monokularen Bildfolge dar. Die Bewegung in den Bildern kann durch Objekte entstehen, die sich zwischen den zwei Aufnahmen in der Welt bewegt haben oder dadurch, dass sich die Kamera zwischen den zwei Aufnahmen bewegt hat (oder beides). Die Projektion dieser 3D-Bewegungen in die 2D-Bildebene wird als Optischer Fluss bezeichnet.

Die beiden aufeinanderfolgenden Bilder der Bildfolge werden in `ImageT1` und `ImageT2` übergeben. Der berechnete optische Fluss wird in `VectorField` zurückgeliefert. Die Vektoren im Vektorfeld `VectorField` stellen

die Bewegung zwischen ImageT1 und ImageT2 in der Bildebene dar. Der zu einem Punkt (r, c) in ImageT1 in ImageT2 zugeordnete Punkt ist also gegeben durch $(r', c') = (r + u(r, c), c + v(r, c))$, wobei $u(r, c)$ und $v(r, c)$ den Wert der Zeilen- bzw. Spaltenkomponente des Vektorfeldbildes VectorField am Punkt (r, c) bezeichnen.

Der Parameter Algorithm erlaubt es, zwischen drei verschiedenen Verfahren zur Berechnung des optischen Flusses auszuwählen. Alle drei Verfahren sind mit Hilfe von numerischen Mehrgitterverfahren implementiert, um eine effiziente Berechnung der zugrundeliegenden partiellen Differentialgleichungen zu ermöglichen.

Für Algorithm = *'fdrig'* wird das Verfahren von Brox, Bruhn, Papenberg und Weickert verwendet. Dieses Verfahren ist flussgetrieben, robust, isotrop und mit einem Gradientenkonstanzterm (**f**low-**d**riven, **r**obust, **i**sotropic with **g**radient constancy).

Für Algorithm = *'ddraw'* wird eine robuste Variante des Verfahrens von Nagel und Enkelmann mit Warping verwendet. Dieses Verfahren ist datengetrieben, robust, anisotrop und verwendet (im Gegensatz zum Originalansatz) Warping (**d**ata-**d**riven, **r**obust, **a**nisotropic with **w**arping).

Für Algorithm = *'clg'* wird das kombinierte lokale-globale Verfahren (**c**ombined **l**ocal-**g**lobal) von Bruhn, Weickert, Feddern, Kohlberger und Schnörr verwendet.

In allen Verfahren können die Eingabebilder zunächst mit einem Gaußfilter mit einer Standardabweichung von SmoothingSigma geglättet werden (siehe derivate_gauss).

Alle drei Verfahren gehören zu der Klasse der sogenannten Variationsansätze, die den optischen Fluss durch die Minimierung eines geeigneten Energiefunktionals berechnen. Im allgemeinen haben solche Energiefunktionale die Form

$$E(\mathbf{w}) = E_{\mathrm{D}}(\mathbf{w}) + \alpha E_{\mathrm{S}}(\mathbf{w}),$$

wobei $\mathbf{w} = (u, v, 1)$ das gesuchte Verschiebungsvektorfeld (mit einem Zeitschritt von 1 in der dritten Koordinate) ist. Die Bildfolge wird als kontinuierliche Funktion $f(\mathbf{x})$ mit $\mathbf{x} = (r, c, t)$ betrachtet, wobei (r, c) für den Ort im Bildbereich steht und t die Zeit angibt. Weiterhin bezeichnen $E_{\mathrm{D}}(\mathbf{w})$ den Datenterm, $E_{\mathrm{S}}(\mathbf{w})$ den Glattheitsterm und α einen Regularisierungsparameter, der die Glattheit der Lösung bestimmt. Der Regularisierungsparameter α wird in FlowSmoothness übergeben. Während im Datenterm Annahmen über die Konstanz von Objektmerkmalen in aufeinanderfolgenden Bildern formuliert werden, z.B. die Konstanz des Grauwerts oder der ersten räumlichen Ableitung des Grauwerts, verkörpert der Glattheitsterm Annahmen über die (stückweise) Glattheit der Lösung, d.h. die Glattheit des gesuchten Verschiebungsvektorfelds.

Das **FDRIG-Verfahren** basiert auf der Minimierung eines Energiefunktionals, das folgende Annahmen beinhaltet:

Konstanz des Grauwerts: Es wird angenommen, dass zueinander gehörende Pixel in aufeinanderfolgenden Bildern einer Bildfolge den gleichen Grauwert haben, also dass $f(r + u, c + v, t + 1) = f(r, c, t)$ gilt. Dies kann in Vektorschreibweise auch kompakter als $f(\mathbf{x} + \mathbf{w}) = f(\mathbf{x})$ formuliert werden.

Konstanz der räumlichen Grauwertableitungen: Es wird angenommen, dass zueinander gehörende Pixel in aufeinanderfolgenden Bildern einer Bildfolge darüber hinaus auch die gleichen räumlichen Grauwertableitungen haben, also dass zusätzlich $\nabla_2 f(x+u, y+v, t+1) = \nabla_2 f(x, y, t)$ gilt, wobei $\nabla_2 f = (\partial_x f, \partial_y f)$. Dies kann kompakter als $\nabla_2 f(\mathbf{x} + \mathbf{w}) = \nabla_2 f(\mathbf{x})$ formuliert werden. Gegenüber der Grauwertkonstanz bietet die Gradientenkonstanz den Vorteil, dass sie invariant gegenüber additiven globalen Helligkeitsschwankungen ist.

Große Verschiebungen: Es wird außerdem angenommen, dass große Verschiebungen, d.h. Verschiebungen, die größer als ein Pixel sind, auftreten können. Unter dieser Annahme ist es sinnvoll, bewusst auf die Linearisierung der Konstanzannahmen im Modell zu verzichten, die üblicherweise in der Literatur vorgeschlagen werden.

Statistische Robustheit im Datenterm: Um starken Verletzungen der Konstanzannahmen durch Ausreißer weniger Bedeutung beizumessen, werden diese statistisch robust bestraft, d.h. die übliche nicht-robuste quadratische Bestrafung $\Psi_{\mathrm{D}}(s^2) = s^2$ wird durch eine lineare Bestrafung mittels $\Psi_{\mathrm{S}}(s^2) = \sqrt{s^2 + \epsilon^2}$ ersetzt, wobei $\epsilon = 0.001$ eine feste Regularisierungskonstante ist.

Diskontinuitätenerhaltung im Flussfeld I: In Bezug auf die Lösung wird angenommen, dass diese stückweise glatt ist. Während die eigentliche Glattheit durch Bestrafungen der ersten Flussableitungen $|\nabla_2 u|^2 + |\nabla_2 v|^2$ erreicht wird, sorgt die Verwendung einer robusten (linearen) Straffunktion $\Psi_{\mathrm{S}}(s^2) = \sqrt{s^2 + \epsilon^2}$ mit $\epsilon = 0.001$ für die gewünschten Erhaltung von Bewegungskanten im zu berechnenden Flussfeld. Dieser Typ des Glattheitsterms wird als flussgetrieben isotrop bezeichnet

Unter der Berücksichtigung all dieser Annahmen lässt sich das Energiefunktional für das FDRIG-Verfahren als

$$E_{\mathrm{FDRIG}}(\mathbf{w}) = \int \Psi_{\mathrm{S}}\Big(\underbrace{|f(\mathbf{x}+\mathbf{w})-f(\mathbf{x})|^2}_{\text{Grauwertkonstanz}} + \gamma \underbrace{|\nabla_2 f(\mathbf{x}+\mathbf{w})-\nabla_2 f(\mathbf{x})|^2}_{\text{Gradientenkonstanz}} \Big) drdc + \alpha \int \Psi_{\mathrm{S}} \Big(\underbrace{|\nabla_2 u(\mathbf{x})|^2 + |\nabla_2 v(\mathbf{x})|^2}_{\text{Glattheitsannahme}} \Big) drdc$$

schreiben. Hierbei ist α der in `FlowSmoothness` übergebene Regularisierungsparameter und γ das in `GradientConstancy` übergebene Gewicht der Gradientenkonstanz. Diese beiden Gewichte, die die Modellparameter des FDRIG-Verfahrens darstellen, werden weiter unten noch genauer beschrieben.

Das **DDRAW-Verfahren** basiert auf der Minimierung eines Energiefunktionals, das folgende Annahmen beinhaltet:

Konstanz des Grauwertes: Es wird angenommen, dass zueinander gehörende Pixel in aufeinanderfolgenden Bildern einer Bildfolge den gleichen Grauwert haben, also dass $f(\mathbf{x}+\mathbf{w}) = f(\mathbf{x})$ gilt.

Große Verschiebungen: Es wird außerdem angenommen, dass große Verschiebungen, d.h. Verschiebungen, die größer als ein Pixel sind, auftreten können. Unter dieser Annahme ist es sinnvoll, bewusst auf die Linearisierung der Konstanzannahmen im Modell zu verzichten, die üblicherweise in der Literatur vorgeschlagen werden.

Statistische Robustheit im Datenterm: Um starken Verletzungen der Konstanzannahmen durch Ausreißer weniger Bedeutung beizumessen, werden diese statistisch robust bestraft, d.h. die übliche nicht-robuste quadratische Bestrafung $\Psi_{\mathrm{D}}(s^2) = s^2$ wird durch eine lineare Bestrafung mittels $\Psi_{\mathrm{S}}(s^2) = \sqrt{s^2+\epsilon^2}$ ersetzt, wobei $\epsilon = 0.001$ eine feste Regularisierungskonstante ist.

Diskontinuitätenerhaltung im Flussfeld II: In Bezug auf die Lösung wird angenommen, dass diese stückweise glatt ist. Im Gegensatz zum FDRIG-Verfahren, das Diskontinuitäten überall zulässt, respektiert das DDRAW-Verfahren nur Diskontinuitäten an Kanten im Originalbild. Dabei wird die lokale Glattheit so gesteuert, dass das Flussfeld über Bildkanten hinweg scharf ist, entlang von Bildkanten jedoch glatt. Dieser Typ des Glattheitsterms nennt sich datengetrieben anisotrop.

Alle Annahmen des DDRAW-Verfahrens lassen sich in einem einzigen Energiefunktional vereinigen:

$$E_{\mathrm{DDRAW}}(\mathbf{w}) = \int \Psi_{\mathrm{S}}\Big(\underbrace{|f(\mathbf{x}+\mathbf{w})-f(\mathbf{x})|^2}_{\text{Grauwertkonstanz}} \Big) drdc + \alpha \int \Big(\underbrace{\nabla_2 u(\mathbf{x})^\top P_{\mathrm{NE}}(\nabla_2 f(\mathbf{x}))\, \nabla_2 u(\mathbf{x}) + \nabla_2 v(\mathbf{x})^\top P_{\mathrm{NE}}(\nabla_2 f(\mathbf{x}))\, \nabla_2 v(\mathbf{x})}_{\text{Glattheitsannahme}} \Big) drdc,$$

wobei $P_{\mathrm{NE}}(\nabla_2 f(\mathbf{x}))$ eine normalisierte Projektionsmatrix orthogonal zu $\nabla_2 f(\mathbf{x})$ ist, für die gilt:

$$P_{\mathrm{NE}}(\nabla_2 f(\mathbf{x})) = \frac{1}{|\nabla_2 f(\mathbf{x})|^2 + 2\epsilon_{\mathrm{S}}^2} \begin{pmatrix} f_c^2(\mathbf{x}) + \epsilon_{\mathrm{S}}^2 & -f_r(\mathbf{x}) f_c(\mathbf{x}) \\ -f_r(\mathbf{x}) f_c(\mathbf{x}) & f_r^2(\mathbf{x}) + \epsilon_{\mathrm{S}}^2 \end{pmatrix}.$$

Diese Matrix sorgt dafür, dass nur in Richtung der Bildkanten eine Glattheit des Flussfelds angenommen wird. Bezüglich der Glattheit über die Bildkanten hinweg werden jedoch keine Annahmen formuliert, so dass genau in diese Richtung Diskontinuitäten in der Lösung auftreten dürfen. In diesem Zusammenhang dient $\epsilon_{\mathrm{S}} = 0.001$ als Regularisierungsparameter, der verhindert, dass die Projektionsmatrix $P_{\mathrm{NE}}(\nabla_2 f(\mathbf{x}))$ singulär wird. Im Gegensatz zum FDRIG-Verfahren gibt es beim DDRAW-Verfahren nur einen Modellparameter: den Regularisierungsparameter α. Wie zuvor erwähnt, wird dieser unten noch genauer beschrieben.

Wie den beiden zuvor besprochenen Verfahren liegen auch dem **CLG-Verfahren** bestimmte Annahmen zu Grunde:

Konstanz des Grauwertes: Es wird angenommen, dass zueinander gehörende Pixel in aufeinanderfolgenden Bildern einer Bildfolge den gleichen Grauwert haben, also dass $f(\mathbf{x}+\mathbf{w}) = f(\mathbf{x})$ gilt.

Kleine Verschiebungen: Im Gegensatz zu beiden vorigen Verfahren wird außerdem angenommen, dass nur kleine Verschiebungen auftreten können, d.h. Verschiebungen im Bereich weniger Pixel. Dies erlaubt eine Linearisierung der Konstanzannahmen im Modell. Dies führt auf die Approximation $f(\mathbf{x}) + \nabla_3 f(\mathbf{x})^\top \mathbf{w}(\mathbf{x}) = f(\mathbf{x})$, d.h. es soll gelten: $\nabla_3 f(\mathbf{x})^\top \mathbf{w}(\mathbf{x}) = 0$. Hierbei ist mit $\nabla_3 f(\mathbf{x})$ der Gradient über sowohl den Ort als auch die Zeit bezeichnet.

Lokale Konstanz der Lösung: Des weiteren geht man davon aus, dass das zu berechnende Flussfeld lokal konstant ist. Dies erlaubt die Integration der Bilddaten im Datenterm über die jeweilige Umgebung eines Bildpunktes. Dies wiederum erhöht die Robustheit des Verfahrens gegenüber Rauschen. Mathematisch lässt sich dies durch eine Umformulierung des quadratischen Datenterms erreichen: $(\nabla_3 f(\mathbf{x})^\top \mathbf{w}(\mathbf{x}))^2 = \mathbf{w}(\mathbf{x})^\top \nabla_3 f(\mathbf{x}) \nabla_3 f(\mathbf{x})^\top \mathbf{w}(\mathbf{x})$. Durch lokale Gauß-gewichtete Integration über eine Nachbarschaft der Größe ρ (übergeben in `IntegrationSigma`) ergibt sich dann folgender Datenterm: $\mathbf{w}(\mathbf{x})^\top G_\rho * (\nabla_3 f(\mathbf{x}) \nabla_3 f(\mathbf{x})^\top)\, \mathbf{w}(\mathbf{x})$. Hierbei bezeichnet $G_\rho * \ldots$ eine Faltung der 3×3 Matrix $\nabla_3 f(\mathbf{x}) \nabla_3 f(\mathbf{x})^\top$ mit einem Gaußfilter mit einer Standardabweichung von ρ (siehe `derivate_gauss`).

Allgemeine Glattheit des Flussfelds: Schließlich wird im Bezug auf die Lösung angenommen, dass sie überall im Bildbereich glatt ist. Dieser spezielle Typ des Glattheitsterms wird als homogen bezeichnet.

Auch im Fall des CLG-Verfahrens lassen sich alle Annahmen in einem gemeinsamem Energiefunktional formulieren. Dieses lautet:

$$E_{\mathrm{CLG}}(\mathbf{w}) = \int \underbrace{\left(\mathbf{w}(\mathbf{x})^\top G_\rho * (\nabla_3 f(\mathbf{x}) \nabla_3 f(\mathbf{x})^\top)\, \mathbf{w}(\mathbf{x})\right)}_{\text{Grauwertkonstanz}} dr dc + \alpha \int \Big(\underbrace{|\nabla_2 u(\mathbf{x})|^2 + |\nabla_2 v(\mathbf{x})|^2}_{\text{Glattheitsannahme}} \Big) dr dc\,,$$

Die zugehörigen Modellparameter sind der Regularisierungsparameter α sowie die Integrationsskala ρ (übergeben in `IntegrationSigma`), die die Größe der zu integrierenden Nachbarschaft im Datenterm angibt. Diese beiden Parameter sind weiter unten genauer beschrieben.

Um das gewünschte Verschiebungsvektorfeld zwischen zwei aufeinanderfolgenden Bildern einer Bildfolge mit dem FDRIG-, dem DDRAW- oder dem CLG-Verfahren zu berechnen, muss diejenige Lösung gesucht werden, die die Annahmen des jeweiligen Verfahren am besten erfüllt. Aus mathematischer Sicht bedeutet das, dass man an einer Minimierung der oben beschriebenen Energiefunktionale interessiert ist. Bei dieser Minimierung spielen im Rahmen der ersten beiden Funktionale (FDRIG und DDRAW) sogenannte Coarse-to-Fine-Warping-Strategien eine wichtige Rolle, da sie es erlauben, große Verschiebungen zu schätzen. Sie sind damit ein geeignetes Mittel, um den bewussten Verzicht der beiden Verfahren auf eine Linearisierung der Konstanzannahmen im Datenterm numerisch umzusetzen.

Um große Verschiebungen schätzen zu können, machen Coarse-to-Fine-Warping-Strategien von zwei Konzepten Gebrauch, die beide eng miteinander verzahnt sind: Die sukzessive Verfeinerung der Problemstellung (Coarse-to-Fine) und die fortlaufende Kompensation des aktuelles Bildpaares um bereits berechnete Verschiebungen (Warping). Algorithmisch lässt sich eine solche Coarse-to-Fine-Warping-Strategie wie folgt formulieren:

1. Zunächst werden beide Bilder des aktuellen Bildpaares auf eine sehr grobe Auflösungsstufe heruntergesampelt.

2. Dann wird das Verschiebungsvektorfeld auf dieser groben Auflösung berechnet.

3. Dieses Vektorfeld wird auf der nächstfeineren Auflösungsstufe benötigt: Es wird dort aus dem zweiten Bild der Bildfolge herausgerechnet, d.h. die Problemstellung auf der feineren Auflösungsstufe wird um das bereits berechnete Flussfeld kompensiert. Dieser Schritt wird auch als Warping bezeichnet.

4. Das so modifizierte Problem (Differenzproblem) wird nun auf der feineren Auflösungsstufe gelöst, d.h. das Verschiebungsvektorfeld wird dort berechnet.

5. Die Schritte 3-4 werden wiederholt, bis die ursprüngliche Auflösung erreicht wird.

6. Das letztendliche Ergebnis wird durch Addition der Verschiebungsvektorfelder aller Auflösungsstufen gebildet.

Diese inkrementelle Berechnung des Verschiebungsvektorfelds bietet folgenden Vorteil: Während das Coarse-to-Fine-Konzept sicherstellt, dass die Verschiebungen auf der gröbsten Auflösungsstufe sehr klein sind, sorgt die Warping-Strategie dafür, dass das auch für die zu berechnenden Verschiebungsinkremente (Verschiebungsvektorfelder der Differenzprobleme) so bleibt. Da kleine Verschiebungen viel genauer berechnet werden können als größere, nimmt durch den Einsatz einer solchen Coarse-to-Fine-Warping-Strategie die Schätzqualität im allgemeinen deutlich zu. Jedoch muss anstelle eines einzelnen Korrespondenzproblems eine Hierarchie von ebensolchen Problemen gelöst werden. Im Rahmen des CLG-Verfahrens ist solch eine Coarse-to-Fine-Warping-Strategie nicht notwendig. Dort geht man bereits im Modell von kleineren Verschiebungen aus.

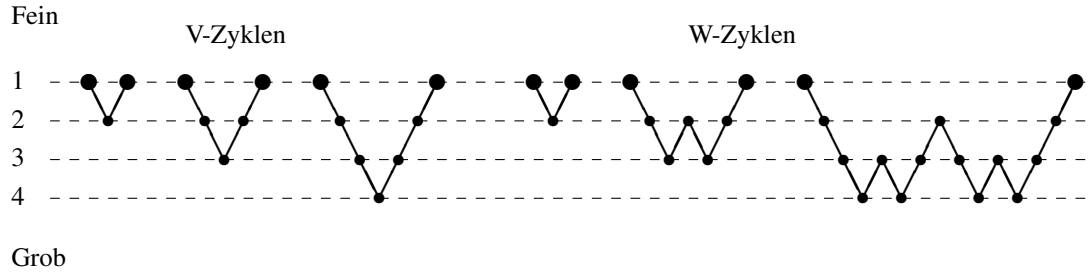
Die maximale Anzahl der Auflösungsstufen (Warpingebene), das Auflösungsverhältnis zwischen zwei aufeinanderfolgenden Auflösungsstufen, sowie die feinste Auflösungsstufe lassen sich sowohl beim FDRIG- als auch beim DDRAW-Verfahren spezifizieren. Nähere Details hierzu sind weiter unten zu finden.

Die Minimierung von Funktionalen ist aus mathematischer Sicht sehr eng mit der Minimierung von Funktionen verwandt: So wie eine Nullstelle der ersten Ableitung eine notwendige Bedingung für ein Minimum einer Funktion ist, ist die Erfüllung der sogenannten Euler-Lagrange-Gleichungen eine notwendige Bedingung für die minimierende Funktion eines Funktionals (die minimierende Funktion entspricht hier dem gesuchten Verschiebungsvektorfeld). Die Euler-Lagrange-Gleichungen sind partielle Differentialgleichungen. Durch die Diskretisierung

dieser Euler-Lagrange-Gleichungen mit Hilfe finiter Differenzen entstehen im Fall des FDRIG- und des DDRAW-Verfahrens große dünn besetzte nichtlineare Gleichungssysteme. Aufgrund der Verwendung der oben beschriebenen Coarse-to-Fine-Warping-Strategien muss ein solches Gleichungssystem für jede Auflösungsstufe, d.h. für jede Warpingebene, gelöst werden. Im Fall des CLG-Verfahrens entsteht hingegen ein einzelnes großes dünn besetztes lineares Gleichungssystem. großes dünn besetztes lineares Gleichungssystem.

Um ein effizientes Lösen dieser nichtlinearen Gleichungssysteme zu gewährleisten, greifen sowohl das FDRIG- als auch das DDRAW-Verfahren auf sogenannte bidirektionale Mehrgitterverfahren zurück. Aus numerischer Sicht gehören diese Strategien zu den schnellsten Verfahren zum Lösen linearer und nichtlinearer Gleichungssysteme überhaupt. Im Gegensatz zu herkömmlichen nicht-hierarchischen iterativen Verfahren, wie z.B. den verschiedenen linearen und nichtlinearen Gauß-Seidel-Varianten, bieten solche Mehrgitterstrategien den Vorteil, Korrekturen der Lösung effizient auf gröberen Auflösungsstufen berechnen zu können. Dies führt wiederum zu einer erheblich schnelleren Konvergenz. Die grundlegende Idee von Mehrgitterverfahren besteht darüber hinaus darin, dass diese Korrekturschritte hierarchisch ausgeführt werden, d.h. dass die Berechnung des Fehlers auf der groben Auflösungsstufe selbst auf die gleiche Strategie zurückgreift und ihren eigenen Fehler (also den Fehler des Fehlers) durch Korrekturschritte auf einer noch gröberen Auflösungsstufe effizient berechnet. Je nachdem, ob ein oder zwei Fehlerkorrekturschritte pro Zyklus gemacht werden, spricht man von sogenannten V- oder W-Zyklen. Die zugehörigen Auflösungsdurchlaufstrategien sehen mit zwei bis vier Auflösungsstufen wie folgt aus:

Bidirektionales Mehrgitterverfahren



Hierbei werden Iterationen auf dem Originalproblem mit großen Markierungen symbolisiert, während kleine Markierungen für Iterationen auf Fehlerkorrekturproblemen stehen.

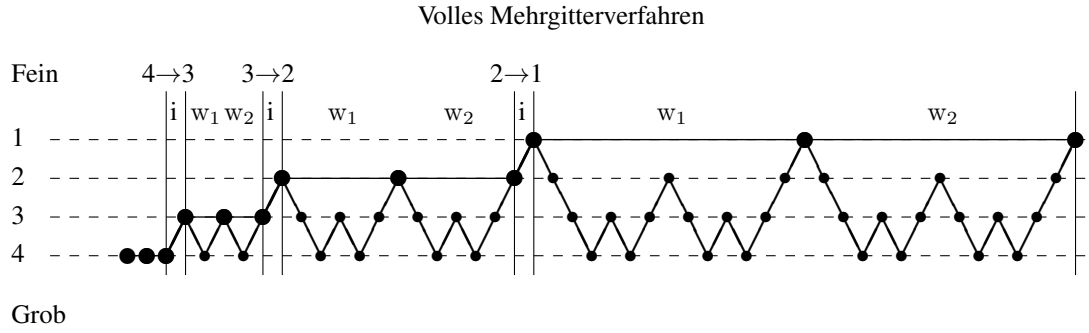
Algorithmisch lässt sich ein solcher Korrekturzyklus wie folgt beschreiben:

1. In einem ersten Schritt werden einige wenige Iterationen mit einem iterativen linearen oder nichtlinearen Basislöser (z.B. einer Gauß-Seidel-Variante) gemacht. Dieser Schritt nennt sich Fehlervorglättungsiterationsschritt.

2. In einem zweiten Schritt wird der aktuelle Fehler berechnet, um die bisherige Lösung (die Lösung nach Schritt 1) zu korrigieren. Aus Effizienzgründen findet diese Fehlerberechnung auf einer gröberen Auflösungsstufe statt. Dieser Schritt, der mehrfach wiederholt werden kann, nennt sich Grobgitterkorrekturschritt.

3. In einem letzten Schritt werden erneut einige wenige Iterationen mit dem gleichen iterativen linearen oder nichtlinearen Basislöser wie in Schritt 1 gemacht. Dieser Schritt nennt sich Fehlernachglättungsiterationsschritt.

Darüber hinaus besteht noch die Möglichkeit einer hierarchischen Initialisierung. Ausgehend von einer sehr groben Variante des ursprünglichen (nicht)linearen Gleichungssystems wird die Lösung sukzessive verfeinert. Dabei werden interpolierte Lösungen von gröberen Varianten des Gleichungssystems als Initialisierung der jeweils nächstfeineren Variante verwendet. Auf jeder Auflösungsstufe selbst kommen dann die zuvor beschriebenen V- oder W-Korrekturzyklen zum Einsatz, um das dortige (nicht)lineare Gleichungssystem effizient zu lösen. Die entsprechenden Mehrgitterverfahren werden in der Literatur als volle Mehrgitterverfahren bezeichnet. Diese Vorgehensweise kann wie folgt veranschaulicht werden:



Dieses Beispiel stellt ein volles Mehrgitterverfahren mit zwei W-Korrekturzyklen pro Auflösungsstufe der hierarchischen Initialisierung dar. Die Interpolationsschritte der Lösung von einer Stufe zur nächsten sind mit i bezeichnet, die beiden W-Korrekturzyklen mit w_1 und w_2. Während Iterationen auf dem Originalproblem mit großen Markierungen symbolisiert werden, stehen kleine Markierungen für Iterationen auf Fehlerkorrekturproblemen.

Im Rahmen der Mehrgitterimplementation des FDRIG-Verfahrens, des DDRAW- sowie des CLG-Verfahrens lassen sich folgende Parameter frei wählen: ob eine hierarchische Initialisierung stattfindet; die Anzahl der Grobgitterkorrekturschritte; die Anzahl der Korrekturzyklen; die maximale Anzahl an Korrekturebenen (Auflösungsstufen); die Anzahl der Fehlervorglättungsiterationen; die Anzahl der Fehlernachglättungsiterationen. Diese Parameter werden weiter unten noch genauer erklärt.

Als Basislöser für das FDRIG-Verfahren wird eine punktgekoppelte Fixpunktvariante des linearen Gauß-Seidel-Verfahrens verwendet, als Basislöser für das DDRAW-Verfahren eine alternierend liniengekoppelte Fixpunktvariante des gleichen Typs. Die Anzahl der Fixpunktschritte lässt sich dabei bei beiden Verfahren ebenfalls durch einen zusätzlichen Parameter spezifizieren. Der Basislöser für das CLG-Verfahren ist ein punktgekoppeltes lineares Gauß-Seidel-Verfahren. Der Transfer von Daten zwischen verschiedenen Auflösungsstufen erfolgt durch flächenbasierte Interpolation bzw. durch flächenbasierte Mittelung.

Nachdem die verwendeten Verfahren beschrieben wurden, können die Auswirkungen der einzelnen Parameter detailliert beschrieben werden.

Die Eingabebilder werden mit ihren Definitionsbereichen (Regionen) in `ImageT1` und `ImageT2` übergeben. Die Berechnung des Vektorfeldes `VectorField` erfolgt auf dem kleinsten umschließenden Rechteck des Schnittes der Definitionsbereiche von `ImageT1` und `ImageT2`. Der Definitionsbereich von `VectorField` ist der Schnitt der beiden Definitionsbereiche. Daher kann durch Angabe von eingeschränkten Definitionsbereichen in `ImageT1` und `ImageT2` die Verarbeitung fokussiert werden, und u.U. Laufzeit gespart werden. Allerdings ist dabei zu beachten, dass alle Verfahren eine globale Lösung des optischen Flusses berechnen. Daraus folgt insbesondere, dass eine Lösung auf einem eingeschränkten Definitionsbereich nicht die eingeschränkte Lösung auf dem vollen Definitionsbereich sein muss (und kann).

`SmoothingSigma` beschreibt die Standardabweichung des Gaußkerns, mit dem jedes Eingangsbild gefaltet wird. Je größer der Wert für `SmoothingSigma`, desto größer ist die Tiefpasswirkung des Gaußkerns, d.h. desto glatter wird das vorverarbeitete Bild. Im Normalfall stellt ein Wert von `SmoothingSigma` = 0.8 eine gute Wahl dar, aber auch andere Werte im Intervall $[0, 2]$ sind denkbar. Größere Standardabweichungen sollten nur in Betracht gezogen werden, falls das Eingangsbild stark verrauscht ist. Dabei ist jedoch zu beachten, dass größere Werte für `SmoothingSigma` eine leichte Erhöhung der Laufzeit zur Folge haben.

`IntegrationSigma` beschreibt die Standardabweichung ρ des Gaußkerns G_ρ, der zur lokalen Integration der Nachbarschaftsinformation im Datenterm verwendet wird. Dieser Parameter wird nur beim CLG-Verfahren verwendet und hat bei den anderen beiden Verfahren keinen Einfluss auf das Ergebnis. Im Normalfall stellt ein Wert von `IntegrationSigma` = 1.0 eine gute Wahl dar, aber auch andere Werte im Intervall $[0, 3]$ sind denkbar. Größere Standardabweichungen sollten nur in Betracht gezogen werden, falls das Eingangsbild stark verrauscht ist. Dabei ist jedoch zu beachten, dass größere Werte für `IntegrationSigma` eine leichte Erhöhung der Laufzeit zur Folge haben.

`FlowSmoothness` spezifiziert das Gewicht α des Glattheitsterms gegenüber dem Datenterm. Je größer der Wert für `FlowSmoothness` ist, desto glatter wird das berechnete Flussfeld. Dabei sollte beachtet werden, dass zu kleine Werte für `FlowSmoothness` trotz statistisch robuster Bestrafungsfunktionen zu unbrauchbaren Ergebnissen führen können, insbesondere dann, wenn die verwendete Warping-Strategie zu viel Information jenseits des Bildbereichs vorhersagen muss. Bei der Verwendung von Byte-Bildern mit einem Wertebereich von $[0, 255]$ liefern Werte für `FlowSmoothness` von etwa 20 im Fall des flussgetriebenen FDRIG-Verfahrens und Werte von etwa

1000 im Fall des bildgetriebenen DDRAW-Verfahrens bzw. des homogenen CLG-Verfahrens im Normalfall gute Ergebnisse.

`GradientConstancy` beschreibt das Gewicht γ der Gradientenkonstanz gegenüber der Grauwertkonstanz. Dieser Parameter wird nur beim FDRIG-Verfahren verwendet. Bei den anderen beiden Verfahren hat er keinen Einfluss auf das Ergebnis. Für Grauwertbilder mit einem Wertebereich von $[0, 255]$ ist meist ein Wert um `GradientConstancy` = 5 sinnvoll, da beide Konstanzannahmen in ähnlichem Ausmaß berücksichtigt werden. Bei starken Beleuchtungswechseln können jedoch auch wesentlich größere Werte für `GradientConstancy` erforderlich sein, um gute Ergebnisse zu erzielen. Dabei ist zu beachten, dass bei großen Werten für das Gradientengewicht auch der Glattheitsparameter `FlowSmoothness` entsprechend größer gewählt werden muss.

Die Parameter für den Mehrgitterlöser und die Coarse-to-Fine-Warping-Strategie lassen sich mit den generischen Parametern `MGParamName` und `MGParamValue` einstellen. Im Normalfall ist es ausreichend, einen der vier weiter unten beschriebenen Standardparametersätze über `MGParamName` = *'default_parameters'* und `MGParamValue` = *'very_accurate'*, *'accurate'*, *'fast_accurate'* oder *'fast'* auszuwählen. Falls die Parameter einzeln angegeben werden sollen, ist in `MGParamName` und `MGParamValue` jeweils ein gleichlanges Tupel von Werten zu übergeben. Dabei müssen die Werte für die in `MGParamName` angegebenen Parameter an der jeweiligen Stelle des Tupels `MGParamValue` spezifiziert werden.

Mit `MGParamName` = *'warp_zoom_factor'* kann das Auflösungsverhältnis zwischen zwei aufeinanderfolgenden Warpingebenen in der Coarse-to-Fine-Warping-Hierarchie festgelegt werden. Dabei muss *'warp_zoom_factor'* im offenen Intervall $(0, 1)$ gewählt werden. Aus Performanzgründen wird üblicherweise *'warp_zoom_factor'* = 0.5 gewählt, d.h. die Pixelanzahl der Eingangsbilder wird von einer Warpingebene zur nächsten in jede Richtung halbiert. Dies führt zu einem zusätzlichen Rechenaufwand von etwa 33% gegenüber einem Verfahren ohne Warping. Werte für *'warp_zoom_factor'* näher bei 1 können zu qualitativ leicht besseren Ergebnissen führen, benötigen aber unverhältnismäßig viel Rechenzeit, z.B. einen Mehraufwand von 426% bei *'warp_zoom_factor'* = 0.9.

Mit `MGParamName` = *'warp_levels'* kann die Warpinghierarchie auf eine maximale Anzahl von Ebenen beschränkt werden. Für *'warp_levels'* = 0 wird die maximal mögliche Anzahl von Ebenen gewählt. Sollte die Bildgröße bei einem gegebenen Auflösungsverhältnis *'warp_zoom_factor'* die gewählte Anzahl an Ebenen nicht zulassen, wird die maximal mögliche Anzahl an Ebenen verwendet. Im Normalfall sollte *'warp_levels'* auf 0 gesetzt werden.

Mit `MGParamName` = *'warp_last_level'* kann die Anzahl der Warpingebenen an, für die *kein* Bewegungsinkrement mehr berechnet werden soll, festgelegt werden. Normalerweise wird diese Anzahl entweder auf 1 oder 2 gesetzt, d.h. es wird entweder auf jeder Warpingebene ein Bewegungsinkrement berechnet, oder die feinste Warpingebene wird bei der Berechnung ausgelassen. Da im letzten Fall nur auf halber Auflösung gerechnet wird, kann die gewonnene Rechenzeit zur genaueren Bestimmung der Lösung verwendet werden, d.h. es kann z.B. ein volles Mehrgitterverfahren mit zusätzlichen Iterationen in Betracht gezogen werden. Die genauere Lösung wird dann auf die volle Auflösung interpoliert.

Die drei Parameter, die die Coarse-to-Fine-Warping-Strategie bestimmen, haben nur bei dem FDRIG- und dem DDRAW-Verfahren einen Einfluss auf die Lösung. Bei dem CLG-Verfahren werden sie ignoriert.

Mit `MGParamName` = *'mg_solver'* kann die allgemeine Mehrgitterstrategie zur Lösung des (nicht)linearen Gleichungssystems (in jeder Warpingebene) festgelegt werden. Für *'mg_solver'* = *'multigrid'* wird ein normales Mehrgitterverfahren (*ohne* Coarse-to-Fine-Initialisierung) verwendet, während für *'mg_solver'* = *'full_multigrid'* ein volles Mehrgitterverfahren (*mit* Coarse-to-Fine-Initialisierung) verwendet wird. Da zwischen zwei aufeinanderfolgenden Ebenen der Coarse-to-Fine-Initialisierung eine Auflösungsreduktion um den Faktor 0.5 verwendet wird (dieser Wert ist im Gegensatz zu der Auflösungsreduktion bei der Warping-Strategie fest kodiert), bedeutet die Verwendung eines vollen Mehrgitterverfahrens einen Mehraufwand von etwa 33% gegenüber einem normalen Mehrgitterverfahren. Bei der Wahl von *'mg_solver'* = *'full_multigrid'* ergeben sich typischerweise numerisch genauere Ergebnisse als für *'mg_solver'* = *'multigrid'*.

Mit `MGParamName` = *'mg_cycle_type'* kann festgelegt werden, ob ein V- oder ein W-Korrekturzyklus pro Mehrgitterebene verwendet werden. Bei der verwendeten Auflösungsreduktion um den Faktor 0.5 zwischen zwei aufeinanderfolgenden Ebenen des jeweiligen Korrekturzyklus erhöht die Verwendung eines W-Zyklus an Stelle eines V-Zyklus den Rechenaufwand um 50%. Bei der Wahl von *'mg_cycle_type'* = *'w'* ergeben sich typischerweise numerisch genauere Ergebnisse als für *'mg_cycle_type'* = *'v'*.

Mit `MGParamName` = *'mg_levels'* kann die Mehrgitterhierarchie sowohl für die Coarse-to-Fine-Initialisierung als auch für die eigentlichen V- oder W-Korrekturzyklen auf eine maximale Anzahl von Ebenen beschränkt werden. Für *'mg_levels'* = 0 wird die maximal mögliche Anzahl von Ebenen gewählt. Sollte die Bildgröße die gewählte Anzahl an Ebenen nicht zulassen, wird die maximal mögliche Anzahl an Ebenen verwendet. Im Normalfall sollte *'mg_levels'* auf 0 gesetzt werden.

Mit MGParamName = *'mg_cycles'* kann die gesamte Anzahl von V- oder W-Korrekturzyklen, die durchgeführt werden, festgelegt werden. Im Falle der Verwendung eines vollen Mehrgitterverfahrens bezieht sich *'mg_cycles'* auf *jede* Ebene der Coarse-to-Fine-Initialisierung. Im Normalfall sind ein oder zwei Zyklen ausreichend, um eine hinreichend genaue Lösung des Gleichungssystems zu erhalten. Je größer *'mg_cycles'* gewählt wird, desto genauer werden typischerweise die numerischen Ergebnisse. Dieser Parameter geht fast linear in die Laufzeit ein, d.h. eine Verdopplung der Zyklen führt in etwa auch zu einer Verdopplung des Rechenaufwands.

Mit MGParamName = *'mg_pre_relax'* kann die Anzahl der Iterationen festgelegt werden, die auf jeder Ebene eines V- oder W-Korrekturzyklus mit dem iterativen Basislöser *vor* der eigentlichen Fehlerkorrektur durchgeführt werden. Im Normalfall sind ein oder zwei Fehlervorglättungsiterationen ausreichend. Je größer *'mg_pre_relax'* gewählt wird, desto genauer werden typischerweise die numerischen Ergebnisse.

Mit MGParamName = *'mg_post_relax'* kann die Anzahl der Iterationen festgelegt werden, die auf jeder Ebene eines V- oder W-Korrekturzyklus mit dem iterativen Basislöser *nach* der eigentlichen Fehlerkorrektur durchgeführt werden. Im Normalfall sind ein oder zwei Fehlernachglättungsiterationen ausreichend. Je größer *'mg_post_relax'* gewählt wird, desto genauer werden typischerweise die numerischen Ergebnisse.

Wie bei der Erhöhung der Anzahl der Korrekturzyklen geht eine gleichzeitige Erhöhung der Anzahl der Fehlervor- und Fehlernachglättungsiterationen asymptotisch linear in die Laufzeit ein, allerdings fallen keine zusätzlichen Restriktions- und Prolongationsoperationen (Verkleinerungen und Vergrößerungen der Fehlerkorrekturbilder) an. Für eine moderate Erhöhung nimmt der Mehraufwand deshalb nur leicht zu.

Mit MGParamName = *'mg_inner_iter'* kann die Anzahl der Iterationen zum Lösen der linearen Gleichungssysteme innerhalb jeder Fixpunktiteration des nichtlinearen Basislösers festgelegt werden. Im Normalfall ist eine Iteration ausreichend, um eine ausreichende Konvergenzgeschwindigkeit des Mehrgitterverfahrens zu erreichen. Die Zunahme des Rechenaufwandes bei Erhöhung der Iterationsanzahl ist noch etwas geringer als bei der Erhöhung der Fehlervor- und Fehlernachglättungsiterationen. Dieser Parameter beeinflusst nur das FDRIG- und das DDRAW-Verfahren, da beim CLG-Verfahren kein nichtlineares Gleichungssystem gelöst werden muss.

Wie oben beschrieben, ist es im Normalfall ausreichend, einen der Standardparametersätze für die oben beschriebenen Parameter über MGParamName = *'default_parameters'* und MGParamValue = *'very_accurate'*, *'accurate'*, *'fast_accurate'* oder *'fast'* auszuwählen. Dabei können im Bedarfsfall noch einzelne Parameter nach der Auswahl der Standardparametersätze modifiziert werden, indem nach dem Wert *'default_parameters'* in MGParamName noch eine Untermenge der obigen Parameter angegeben wird und in MGParamValue ein entsprechender Wert übergeben wird (z.B. MGParamName = *['default_parameters','warp_zoom_factor']* und MGParamValue = *['accurate',0.6]*).

Im einzelnen entsprechen die Standardparametersätze folgenden Werten:

'default_parameters' = *'very_accurate'*: *'warp_zoom_factor'* = 0.5, *'warp_levels'* = 0, *'warp_last_level'* = 1, *'mg_solver'* = *'full_multigrid'*, *'mg_cycle_type'* = *'w'*, *'mg_levels'* = 0, *'mg_cycles'* = 1, *'mg_pre_relax'* = 2, *'mg_post_relax'* = 2, *'mg_inner_iter'* = 1.

'default_parameters' = *'accurate'*: *'warp_zoom_factor'* = 0.5, *'warp_levels'* = 0, *'warp_last_level'* = 1, *'mg_solver'* = *'multigrid'*, *'mg_cycle_type'* = *'v'*, *'mg_levels'* = 0, *'mg_cycles'* = 1, *'mg_pre_relax'* = 1, *'mg_post_relax'* = 1, *'mg_inner_iter'* = 1.

'default_parameters' = *'fast_accurate'*: *'warp_zoom_factor'* = 0.5, *'warp_levels'* = 0, *'warp_last_level'* = 2, *'mg_solver'* = *'full_multigrid'*, *'mg_cycle_type'* = *'w'*, *'mg_levels'* = 0, *'mg_cycles'* = 1, *'mg_pre_relax'* = 2, *'mg_post_relax'* = 2, *'mg_inner_iter'* = 1.

'default_parameters' = *'fast'*: *'warp_zoom_factor'* = 0.5, *'warp_levels'* = 0, *'warp_last_level'* = 2, *'mg_solver'* = *'multigrid'*, *'mg_cycle_type'* = *'v'*, *'mg_levels'* = 0, *'mg_cycles'* = 1, *'mg_pre_relax'* = 1, *'mg_post_relax'* = 1, *'mg_inner_iter'* = 1.

Es ist zu beachten, dass die beiden Modi *'fast_accurate'* und *'fast'* für das CLG-Verfahren identisch zu den Modi *'very_accurate'* und *'accurate'* sind, da das CLG-Verfahren keine Coarse-to-Fine-Warping-Strategie verwendet.

Parameter

- ▷ **ImageT1** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 1.
- ▷ **ImageT2** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 2.
- ▷ **VectorField** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* vector_field
  Optischer Fluss.

▷ **Algorithm** (input_control) . . . . . string ⇝ *string*
Algorithmus zur Bestimmung des optischen Flusses.
Defaultwert: 'fdrig'
Werteliste: `Algorithm` ∈ {'fdrig', 'ddraw', 'clg'}

▷ **SmoothingSigma** (input_control) . . . . . real ⇝ *real*
Standardabweichung für initiale Gaußsche Glättung.
Defaultwert: 0.8
Wertevorschläge: `SmoothingSigma` ∈ {0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0}
Restriktion: SmoothingSigma >= 0.0

▷ **IntegrationSigma** (input_control) . . . . . real ⇝ *real*
Standardabweichung des Integrationsfilters.
Defaultwert: 1.0
Wertevorschläge: `IntegrationSigma` ∈ {0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0}
Restriktion: IntegrationSigma >= 0.0

▷ **FlowSmoothness** (input_control) . . . . . real ⇝ *real*
Gewicht des Glattheitsterms gegenüber dem Datenterm.
Defaultwert: 20.0
Wertevorschläge: `FlowSmoothness` ∈ {10.0, 20.0, 30.0, 50.0, 70.0, 100.0, 200.0, 300.0, 500.0, 700.0, 1000.0, 1500.0, 2000.0}
Restriktion: FlowSmoothness >= 0.0

▷ **GradientConstancy** (input_control) . . . . . real ⇝ *real*
Gewicht der Gradientenkonstanz gegenüber der Grauwertkonstanz.
Defaultwert: 5.0
Wertevorschläge: `GradientConstancy` ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 15.0, 20.0, 30.0, 40.0, 50.0, 70.0, 100.0}
Restriktion: GradientConstancy >= 0.0

▷ **MGParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Parametername(n) für das Mehrgitterverfahren.
Defaultwert: 'default_parameters'
Werteliste: `MGParamName` ∈ {'default_parameters', 'mg_solver', 'mg_cycle_type', 'mg_levels', 'mg_cycles', 'mg_pre_relax', 'mg_post_relax', 'mg_inner_iter', 'warp_levels', 'warp_zoom_factor', 'warp_last_level'}

▷ **MGParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / real / integer
Parameterwert(e) für das Mehrgitterverfahren.
Defaultwert: 'accurate'
Wertevorschläge: `MGParamValue` ∈ {'very_accurate', 'accurate', 'fast_accurate', 'fast', 'multigrid', 'full_multigrid', 'v', 'w', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}

Beispiel

```
grab_image (ImageT1, AcqHandle)
while (true)
  grab_image (ImageT2, AcqHandle)
  optical_flow_mg (ImageT1, ImageT2, VectorField, 'fdrig', 0.8, 1, 10, \
                   5, 'default_parameters', 'accurate')
  threshold (VectorField, Region, 1, 10000)
  copy_obj (ImageT2, ImageT1, 1, 1)
endwhile
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `optical_flow_mg` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

threshold, vector_field_length

Siehe auch

unwarp_image_vector_field

Literatur

T. Brox, A. Bruhn, N. Papenberg, and J. Weickert: High accuracy optical flow estimation based on a theory for warping. In T. Pajdla and J. Matas, editors, Computer Vision - ECCV 2004, volume 3024 of Lecture Notes in Computer Science, pages 25–36. Springer, Berlin, 2004.

A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnörr: Variational optical flow computation in real-time. IEEE Transactions on Image Processing, 14(5):608-615, May 2005.

H.-H. Nagel and W. Enkelmann: An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(5):565-593, September 1986.

Ulrich Trottenberg, Cornelis Oosterlee, Anton Schüller: Multigrid. Academic Press, Inc., San Diego, 2000.

Modul

Foundation

unwarp_image_vector_field (Image, VectorField : ImageUnwarped : :)

Entzerrung eines Bildes mittels eines Vektorfeldes.

unwarp_image_vector_field entzerrt das Bild Image mit Hilfe des Vektorfeldes VectorField und gibt das entzerrte Bild in ImageUnwarped zurück. Das Vektorfeld muss vom semantischen Typ 'vector_field_relative' sein und wird typischerweise mit optical_flow_mg bestimmt. Somit kann mit unwarp_image_vector_field das zweite Eingabebild von optical_flow_mg auf das erste Eingabebild entzerrt werden. Es ist zu beachten, dass somit das Vektorfeld eine inverse Transformationsvorschrift vom Zielbild des Vektorfeldes in das Ursprungsbild darstellt.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **VectorField** (input_object) ..... singlechannelimage(-array) ⇝ *object :* vector_field
  Eingabevektorfeld.
- ▷ **ImageUnwarped** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Entzerrtes Bild.

Beispiel

```
optical_flow_mg (Image1, Image2, VectorField, 'fdrig', 0.8, 1, 20, \
                 5, 'default_parameters', 'accurate')
unwarp_image_vector_field (Image2, VectorField, ImageUnwarped)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert unwarp_image_vector_field den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Domainebene.

- Automatisch parallelisiert auf Tupelebene.

Vorgänger

optical_flow_mg

Modul

Foundation

vector_field_length (VectorField : Length : Mode :)

Berechnung der Länge der Vektoren eines Vektorfeldes.

vector_field_length berechnet die Länge der Vektoren des in VectorField übergebenen Vektorfeldes und gibt sie in Length zurück. vector_field_length akzeptiert nur Vektorfelder des semantischen Typs 'vector_field_relative'. Mit Mode kann die Art der Berechnung der Längen ausgewählt werden. Für Mode = *'length'* wird die Euklidische Länge der Vektoren berechnet. Für Mode = *'squared_length'* wird das Quadrat der Länge der Vektoren berechnet. Dies vermeidet intern die Berechnung der Wurzel, was auf vielen Prozessoren eine sehr aufwendige Operation ist, und spart somit auf diesen Prozessoren Rechenzeit. Es ist zu beachten, dass das VectorField in relativen Koordinaten angegeben werden muss, wie es z.B. optical_flow_mg liefert.

Parameter

- ▷ **VectorField** (input_object) ..... singlechannelimage(-array) ⇝ *object :* vector_field
  Eingabevektorfeld.
- ▷ **Length** (output_object) ..... singlechannelimage(-array) ⇝ *object :* real
  Länge der Vektoren des Vektorfeldes.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus zur Berechnung der Länge der Vektoren.
  Defaultwert: 'length'
  Werteliste: Mode ∈ {'length', 'squared_length'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert vector_field_length den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Domainebene.
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

optical_flow_mg

Nachfolger

threshold

Modul

Foundation

10.12 Punkte

corner_response (Image : ImageCorner : Size, Weight :)

Suche von Ecken in Bildern.

corner_response ist ein Operator, um Grauwertecken in einem Bild zu extrahieren. Die Formel für die Berechnung lautet:

$$\begin{aligned} R(x,y) &= A(x,y) \cdot B(x,y) - C^2(x,y) - \texttt{Weight} \cdot (A(x,y) + B(x,y))^2 \\ A(x,y) &= W(u,v) * (\nabla_x I(x,y))^2 \\ B(x,y) &= W(u,v) * (\nabla_y I(x,y))^2 \\ C(c,y) &= W(u,v) * (\nabla_x I(x,y) \nabla_y I(x,y)) \end{aligned}$$

wobei I das Eingabe- und R das Ergebnisbild der Filterung darstellt. Zur Glättung (W) wird gauss_image, zur Berechnung der Ableitung (∇) wird sobel_amp verwendet.

Die Corner-Response-Funktion ist invariant gegenüber Rotation. Um eine günstige Abhängigkeit der Funktion $R(x,y)$ von den lokalen Gradienten zu erhalten, ist der Parameter Weight auf 0.04 zu setzen. Hiermit ergeben sich nur an Grauwertecken positive Werte für $R(x,y)$, während gerade Kanten negative Werte erhalten. Der Ausgabebildtyp ist identisch mit dem Eingabebildtyp. Bei Byte-Bildern als Eingabe werden also die negativen Werte auf 0 gesetzt. Falls dies nicht erwünscht ist, sollte das Bild mit convert_image_type in ein Real- oder Int2-Bild konvertiert werden.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / int2 / real
  Eingabebild.
- ▷ **ImageCorner** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / uint2 / int2 / real
  Ergebnis der Filterung.
  Parameteranzahl: ImageCorner == Image
- ▷ **Size** (input_control) ..... integer ⇝ *integer*
  Gewünschte Filtergröße der Gaußmaske.
  Defaultwert: 3
  Wertevorschläge: Size ∈ {3, 5, 7, 9, 11}
- ▷ **Weight** (input_control) ..... real ⇝ *real*
  Gewichtung.
  Defaultwert: 0.04
  Typischer Wertebereich: $0.0 \leq \texttt{Weight} \leq 0.3$
  Minimale Schrittweite: 0.001
  Empfohlene Schrittweite: 0.01

Beispiel

```
read_image(&Fabrik,"fabrik");
corner_response(Fabrik,&CornerResponse,3,0.04);
local_max(CornerResponse,&LocalMax);
disp_image(Fabrik,WindowHandle);
set_color(WindowHandle,"red");
disp_region(LocalMax,WindowHandle);
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

local_max, threshold

Siehe auch

gauss_filter, sobel_amp, convert_image_type

Literatur

C.G. Harris, M.J. Stephens, „A combined corner and edge detector“; Proc. of the 4th Alvey Vision Conference; August 1988; pp. 147-152.

H. Breit, „Bestimmung der Kameraeigenbewegung und Gewinnung von Tiefendaten aus monokularen Bildfolgen“; Diplomarbeit am Lehrstuhl für Nachrichtentechnik der TU München; 30. September 1990.

Modul

Foundation

dots_image (Image : DotImage : Diameter, FilterType, PixelShift :)

Hervorheben von kreisförmigen Punkten.

dots_image hebt kreisförmige Punkte des Durchmessers Diameter in dem Eingabebild Image hervor. dots_image ist damit insbesondere zur Segmentierung von Nadeldrucken, z.B. in OCR-Anwendungen, geeignet. Die Hervorhebung geschieht durch Faltung mit einer an die Punktgröße angepassten Filtermaske. Für Diameter $= 5$ sieht sie z.B. wie folgt aus:

$$\frac{1}{336}\begin{pmatrix} & & -21 & -21 & -21 & & \\ & -21 & 16 & 16 & 16 & -21 & \\ -21 & 16 & 16 & 16 & 16 & 16 & -21 \\ -21 & 16 & 16 & 16 & 16 & 16 & -21 \\ -21 & 16 & 16 & 16 & 16 & 16 & -21 \\ & -21 & 16 & 16 & 16 & -21 & \\ & & -21 & -21 & -21 & & \end{pmatrix}$$

Mit dem Parameter FilterType kann gewählt werden, ob dunkle (*'dark'*), helle (*'light'*) oder alle (*'all'*) Punkte hervorgehoben werden sollen. Der PixelShift kann verwendet werden, um den Kontrast des Ausgabebildes zu erhöhen (PixelShift > 0) oder mögliche abgeschnittene Extremalwerte zu dämpfen (PixelShift $= -1$).

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **DotImage** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Ausgabebild.
- ▷ **Diameter** (input_control) . . . . . integer ⇝ *integer*
  Durchmesser der hervorzuhebenden Punkte.
  Defaultwert: 5
  Werteliste: Diameter ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23}
- ▷ **FilterType** (input_control) . . . . . string ⇝ *string*
  Detektion aller, heller oder dunkler Punkte.
  Defaultwert: 'light'
  Werteliste: FilterType ∈ {'dark', 'light', 'all'}
- ▷ **PixelShift** (input_control) . . . . . integer ⇝ *integer*
  Shift der Filterantwort.
  Defaultwert: 0
  Werteliste: PixelShift ∈ {-1, 0, 1, 2}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

threshold

Alternativen

laplace, laplace_of_gauss, diff_of_gauss, derivate_gauss, convol_image

Modul

Foundation

points_foerstner (Image : : SigmaGrad, SigmaInt, SigmaPoints, ThreshInhom, ThreshShape, Smoothing, EliminateDoublets : RowJunctions, ColumnJunctions, CoRRJunctions, CoRCJunctions, CoCCJunctions, RowArea, ColumnArea, CoRRArea, CoRCArea, CoCCArea)

Suche nach signifikanten Punkten mittels des Förstner-Operators.

points_foerstner extrahiert signifikante Punkte aus einem Bild. Signifikante Punkte sind Punkte im Bild, die sich von ihrer Umgebung unterscheiden, also Punkte, an denen sich das Bildsignal zweidimensional ändert. Diese Änderung tritt zum einen an den Schnittpunkte von Grauwertkanten auf (sogenannte Kreuzungspunkte), zum anderen an Stellen, deren Farbe oder Helligkeit sich von ihrer Umgebung unterscheidet (sogenannte Flächenpunkte).

Die Punktextraktion erfolgt in zwei Stufen: Im ersten Schritt werden die Punktregionen, also inhomogene Regionen mit hoher Isotropie, aus dem Eingabebild extrahiert. Dazu wird die geglättete Matrix

$$M = S * \begin{pmatrix} \sum_{c=1}^{n} I_{x,c}^2 & \sum_{c=1}^{n} I_{x,c} I_{y,c} \\ \sum_{c=1}^{n} I_{x,c} I_{y,c} & \sum_{c=1}^{n} I_{y,c}^2 \end{pmatrix}$$

berechnet, wobei $I_{x,c}$ und $I_{y,c}$ für die ersten Ableitungen der einzelnen Bildkanäle und S für eine Glättung stehen. Wird Smoothing auf *'gauss'* gesetzt, dann werden die Ableitungen mit einem Gaußfilter der Größe SigmaGrad berechnet und die Glättung erfolgt durch einen Gaußfilter der Größe SigmaInt. Wird Smoothing auf *'mean'* gesetzt, werden die Ableitungen mit einem 3×3 Sobelfilter berechnet (SigmaGrad wird ignoriert) und die Glättung erfolgt durch einen SigmaInt $\times$ SigmaInt Mittelwertfilter. Dabei ist

$$\textit{Inhomogenität} = \mathrm{Spur} M$$

das Maß für die Inhomogenität im Bild und

$$\textit{Isotropie} = 4 \cdot \frac{\mathrm{Det} M}{(\mathrm{Spur} M)^2}$$

das Maß der Isotropie der Textur im Bild. Bildpunkte, deren Inhomogenität größer oder gleich ThreshInhom und deren Isotropie gleichzeitig größer oder gleich ThreshShape ist, werden im Anschluss weiter untersucht.

Im zweiten Schritt werden für diese Punkte zwei Optimierungsfunktionen berechnet. Diese Optimierungsfunktionen geben im Wesentlichen für jeden Bildpunkt die durchschnittlichen Abstände zu den Kanten- (Kreuzungspunkte) bzw. Gradientenrichtungen (Flächenpunkte) innerhalb eines Beobachtungsfensters um den Punkt wieder. Für Smoothing = *'gauss'* erfolgt die Durchschnittsbildung mit einem Gaußfilter der Größe SigmaPoints. Für Smoothing = *'mean'* erfolgt die Durchschnittsbildung mit einem SigmaPoints $\times$ SigmaPoints Mittelwertfilter. Die lokalen Minima der Optimierungsfunktionen ergeben die signifikanten Punkte. Ihre Position wird subpixelgenau in (RowJunctions, ColumnJunctions) und (RowArea, ColumnArea) zurückgegeben.

Neben der Position werden für jeden gefundenen Punkt auch die Elemente CoRRJunctions, CoRCJunctions und CoCCJunctions (bzw. CoRRArea, CoRCArea und CoCCArea) seiner Kovarianzmatrix zurückgegeben. Diese erlaubt Rückschlüsse auf die Genauigkeit der gefundenen Punktposition. Um die tatsächlichen Werte zu erhalten, ist es notwendig, im Eingabebild die Stärke des Rauschens zu ermitteln und anschließend alle Komponenten der Kovarianzmatrix mit der Varianz des Rauschens zu multiplizieren. (Die Stärke des Rauschens ermittelt man z.B., indem man auf Bildbereiche mit konstantem Grauwert intensity oder auf Bildbereiche mit einem rampenförmigen Grauwertverlauf plane_deviation anwendet. In beiden Fällen wird die Stärke des Rauschens im Parameter Deviation zurückgegeben.) Das Beispielprogram

%HALCONEXAMPLES%\hdevelop\Filter\Points\ points_foerstner_ellipses.hdev

illustriert diesen Sachverhalt.

Es liegt in der Natur dieses Operators, dass an Ecken häufig zwei separate Punkte erkannt werden: Ein Kreuzungspunkt im Schnittpunkt der Kanten der Ecke und ein Flächenpunkt etwas weiter im Inneren der Ecke. Solche Dopplungen werden automatisch eliminiert, wenn man EliminateDoublets auf *'true'* setzt. Dazu werden alle möglichen Paare von jeweils einem Kreuzungs- und Flächenpunkt untersucht. Liegen die Punkte innerhalb des Beobachtungsfensters der Optimierungsfunktion des jeweiligen anderen Punktes, dann wird für beide Punkte die Genauigkeit ihrer Position berechnet und der Punkt mit der geringeren Genauigkeit verworfen. Ist EliminateDoublets *'false'*, so werden alle gefundenen Punkte zurückgegeben.

Achtung

Es ist zu beachten, dass für Smoothing = *'mean'*, nur ungerade Werte für SigmaInt und SigmaPoints zulässig sind. Gerade Werte werden automatisch durch den nächstgrößeren ungeraden Wert ersetzt.

points_foerstner mit Smoothing = *'gauss'* wird durch SSE2 Instruktionen beschleunigt, falls der Systemparameter *'sse2_enable'* auf *'true'* gesetzt ist (dies ist Standard, sofern SSE2 auf Ihrem Rechner verfügbar ist). Diese Implementierung ist aufgrund numerischer Probleme im Vergleich zur reinen C Version etwas ungenauer (für 'byte' Bilder bewegt sich der Unterschied von RowJunctions und ColumnJunctions in einer Größenordnung von $1.0e-5$). Falls für Ihre Anwendung Genauigkeit wichtiger ist als Geschwindigkeit können Sie *'sse2_enable'* auf *'false'* setzen, bevor Sie points_foerstner aufrufen. Dadurch verwendet points_foerstner keine SSE2 Beschleunigungen. Vergessen Sie nicht *'sse2_enable'* danach wieder auf *'true'* zu setzen.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **SigmaGrad** (input_control) ..... number ⇝ *real* / integer
  Stärke der Glättung zur Berechnung des Gradienten. Für Smoothing = 'mean' wird SigmaGrad ignoriert.
  Defaultwert: 1.0
  Wertevorschläge: SigmaGrad ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
  Typischer Wertebereich: $0.7 \leq$ SigmaGrad ≤ 50.0
  Empfohlene Schrittweite: 0.1
  Restriktion: SigmaGrad > 0.0
- ▷ **SigmaInt** (input_control) ..... number ⇝ *real* / integer
  Stärke der Glättung zur Integration der Gradienten.
  Defaultwert: 2.0
  Wertevorschläge: SigmaInt ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
  Typischer Wertebereich: $0.7 \leq$ SigmaInt ≤ 50.0
  Empfohlene Schrittweite: 0.1
  Restriktion: SigmaInt > 0.0
- ▷ **SigmaPoints** (input_control) ..... number ⇝ *real* / integer
  Stärke der Glättung in den Optimierungsfunktionen.
  Defaultwert: 3.0
  Wertevorschläge: SigmaPoints ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
  Typischer Wertebereich: $0.7 \leq$ SigmaPoints ≤ 50.0
  Empfohlene Schrittweite: 0.1
  Restriktion: SigmaPoints >= SigmaInt && SigmaPoints > 0.6

▷ **ThreshInhom** (input_control) ..... number ⇝ *real* / integer
Schwellenwert zur Bestimmung inhomogener Bildbereiche.
Defaultwert: 200
Wertevorschläge: ThreshInhom ∈ {50, 100, 200, 500, 1000}
Restriktion: ThreshInhom >= 0.0

▷ **ThreshShape** (input_control) ..... real ⇝ *real*
Schwellenwert zur Bestimmung der Punktregionen.
Defaultwert: 0.3
Wertevorschläge: ThreshShape ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7}
Typischer Wertebereich: 0.01 ≤ ThreshShape ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: 0.0 <= ThreshShape && ThreshShape <= 1.0

▷ **Smoothing** (input_control) ..... string ⇝ *string*
Verwendetes Glättungsverfahren.
Defaultwert: 'gauss'
Werteliste: Smoothing ∈ {'gauss', 'mean'}

▷ **EliminateDoublets** (input_control) ..... string ⇝ *string*
Beseitigung mehrfach erkannter Punkte.
Defaultwert: 'false'
Werteliste: EliminateDoublets ∈ {'false', 'true'}

▷ **RowJunctions** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinaten der gefundenen Kreuzungspunkte.

▷ **ColumnJunctions** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinaten der gefundenen Kreuzungspunkte.

▷ **CoRRJunctions** (output_control) ..... number-array ⇝ *real*
Zeilenanteil der Kovarianzmatrix der gefundenen Kreuzungspunkte.

▷ **CoRCJunctions** (output_control) ..... number-array ⇝ *real*
Gemischter Anteil der Kovarianzmatrix der gefundenen Kreuzungspunkte.

▷ **CoCCJunctions** (output_control) ..... number-array ⇝ *real*
Spaltenanteil der Kovarianzmatrix der gefundenen Kreuzungspunkte.

▷ **RowArea** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinaten der gefundenen Flächenpunkte.

▷ **ColumnArea** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinaten der gefundenen Flächenpunkte.

▷ **CoRRArea** (output_control) ..... number-array ⇝ *real*
Zeilenanteil der Kovarianzmatrix der gefundenen Flächenpunkte.

▷ **CoRCArea** (output_control) ..... number-array ⇝ *real*
Gemischter Anteil der Kovarianzmatrix der gefundenen Flächenpunkte.

▷ **CoCCArea** (output_control) ..... number-array ⇝ *real*
Spaltenanteil der Kovarianzmatrix der gefundenen Flächenpunkte.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert points_foerstner den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

gen_cross_contour_xld, disp_cross

Alternativen

points_harris, points_lepetit, points_harris_binomial

Literatur

W. Förstner, E. Gülch: „A Fast Operator for Detection and Precise Location of Distinct Points, Corners and Circular features“. In Proceedings of the Intercommission Conference on Fast Processing of Photogrametric Data, Interlaken, pp. 281-305, 1987.

W. Förstner: „Statistische Verfahren für die automatische Bildanalyse und ihre Bewertung bei der Objekterkennung und -vermessung“. Volume 370, Series C, Deutsche Geodätische Kommission, München, 1991.

W. Förstner: „A Framework for Low Level Feature Extraction“. European Conference on Computer Vision, LNCS 802, pp. 383-394, Springer Verlag, 1994.

C. Fuchs: „Extraktion polymorpher Bildstrukturen und ihre topologische und geometrische Gruppierung“. Volume 502, Series C, Deutsche Geodätische Kommission, München, 1998.

Modul

Foundation

points_harris (Image : : SigmaGrad, SigmaSmooth, Alpha, Threshold : Row, Column)

Suche nach signifikanten Punkten mittels des Harris-Operators.

points_harris extrahiert signifikante Punkte aus einem Bild. Der Harris Operator-basiert auf der geglätteten Matrix

$$M = G_\sigma * \begin{pmatrix} \sum_{c=1}^{n} I_{x,c}^2 & \sum_{c=1}^{n} I_{x,c} I_{y,c} \\ \sum_{c=1}^{n} I_{x,c} I_{y,c} & \sum_{c=1}^{n} I_{y,c}^2 \end{pmatrix},$$

wobei G_σ für eine Gaußsche Glättung der Größe SigmaSmooth und $I_{x,c}$ und $I_{y,c}$ für die ersten Ableitungen der einzelnen Bildkanäle stehen. Die Ableitungen werden mittels eines Gaußfilters der Größe SigmaGrad implementiert. Die resultierenden Punkte sind dann die positiven lokalen Maxima von

$$\mathrm{Det}M - \texttt{Alpha} * (\mathrm{Spur}M)^2.$$

Nötigenfalls können die extrahierten Punkte auf diejenigen mit einer minimalen Filterantwort von Threshold eingeschränkt werden. Die Koordinaten der Punkte werden mit Subpixelgenauigkeit berechnet.

Achtung

points_harris wird durch SSE2 Instruktionen beschleunigt, falls der Systemparameter *'sse2_enable'* auf *'true'* gesetzt ist (dies ist Standard, sofern SSE2 auf Ihrem Rechner verfügbar ist). Diese Implementierung ist aufgrund numerischer Probleme im Vergleich zur reinen C Version etwas ungenauer (für 'byte' Bilder bewegt sich der Unterschied von Row und Column in einer Größenordnung von $1.0e-5$). Falls für Ihre Anwendung Genauigkeit wichtiger ist als Geschwindigkeit können Sie *'sse2_enable'* auf *'false'* setzen, bevor Sie points_harris aufrufen. Dadurch verwendet points_harris keine SSE2 Beschleunigungen. Vergessen Sie nicht *'sse2_enable'* danach wieder auf *'true'* zu setzen.

points_harris kann auf OpenCL-Geräten ausgeführt werden, sofern SigmaGrad und SigmaSmooth zu einer Filtergröße kleiner gleich 129 Pixeln führen. Dies entspricht einem Wert kleiner 20.7 für beide Parameter. Wie bei der SSE2-Variante kann es zu geringfügigen Unterschieden zwischen dem Ergebnis der OpenCL-Implementierung und dem der reinen C-Implementierung kommen.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **SigmaGrad** (input_control) . . . . . real ⇝ *real*
Stärke der Glättung zur Berechnung des Gradienten.
Defaultwert: 0.7
Wertevorschläge: SigmaGrad ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
Typischer Wertebereich: 0.7 ≤ SigmaGrad ≤ 50.0
Empfohlene Schrittweite: 0.1
Restriktion: SigmaGrad > 0.0

▷ **SigmaSmooth** (input_control) . . . . . real ⇝ *real*
Stärke der Glättung zur Integration der Gradienten.
Defaultwert: 2.0
Wertevorschläge: SigmaSmooth ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
Typischer Wertebereich: 0.7 ≤ SigmaSmooth ≤ 50.0
Empfohlene Schrittweite: 0.1
Restriktion: SigmaSmooth > 0.0

▷ **Alpha** (input_control) . . . . . real ⇝ *real*
Gewichtung der quadrierten Spur der Matrix der quadrierten Gradienten.
Defaultwert: 0.08
Wertevorschläge: Alpha ∈ {0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08}
Typischer Wertebereich: 0.001 ≤ Alpha ≤ 0.1
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 0.01
Restriktion: Alpha > 0.0

▷ **Threshold** (input_control) . . . . . number ⇝ *real* / integer
Minimale Filterantwort der Punkte.
Defaultwert: 1000.0
Restriktion: Threshold >= 0.0

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinaten der gefundenen Punkte.

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinaten der gefundenen Punkte.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert points_harris den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

gen_cross_contour_xld

Alternativen

points_foerstner, points_lepetit, points_harris_binomial

Literatur

C. Harris, M. Stephens: „A combined corner and edge detector“. Proceedings of the 4th Alvey Vision Conference, pp. 147-151, 1988.

V. Gouet, N.Boujemaa: „Object-based queries using color points of interest“. IEEE Workshop on Content-Based Access of Image and Video Libraries, CVPR/CBAIVL 2001, Hawaii, USA, 2001.

Modul

Foundation

points_harris_binomial (Image : : MaskSizeGrad, MaskSizeSmooth, Alpha, Threshold, Subpix : Row, Column)

Suche nach signifikanten Punkten mittels der binomialen Approximation des Harris-Operators.

points_harris_binomial extrahiert signifikante Punkte aus dem Bild Image. Der Harris Operator-basiert auf der geglätteten Matrix

$$M = G_\sigma * \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} ,$$

wobei G_σ für eine Binomiale Glättung der Größe MaskSizeSmooth und I_x und I_y für die ersten Ableitungen des Bildes stehen. Die Ableitungen werden mittels eines Sobelfilters der Größe MaskSizeGrad implementiert. Die resultierenden Punkte sind dann die positiven lokalen Maxima von

$$\mathrm{Det}M - \mathtt{Alpha} * (\mathrm{Spur}M)^2.$$

Nötigenfalls können die extrahierten Punkte auf diejenigen mit einer minimalen Filterantwort von Threshold eingeschränkt werden. Die Koordinaten der Punkte können durch setzen des Parameters Subpix auf 'on' (default) mit Subpixelgenauigkeit berechnet werden. Die Subpixelverfeinerung ist jedoch durch setzen des Parameters auf 'off' auch abschaltbar.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild.
- **MaskSizeGrad** (input_control) ..... integer ⇝ *integer*
  Stärke der Glättung (Binomial) zur Berechnung des Gradienten.
  Defaultwert: 5
  Wertevorschläge: MaskSizeGrad ∈ {3, 5, 7, 9, 11, 15, 21, 31}
  Typischer Wertebereich: 3 ≤ MaskSizeGrad ≤ 50
  Empfohlene Schrittweite: 2
  Restriktion: MaskSizeGrad > 0
- **MaskSizeSmooth** (input_control) ..... integer ⇝ *integer*
  Stärke der Glättung zur Integration der Gradienten.
  Defaultwert: 15
  Wertevorschläge: MaskSizeSmooth ∈ {3, 5, 7, 9, 11, 15, 21, 31}
  Typischer Wertebereich: 3 ≤ MaskSizeSmooth ≤ 50
  Empfohlene Schrittweite: 2
  Restriktion: MaskSizeSmooth > 0
- **Alpha** (input_control) ..... real ⇝ *real*
  Gewichtung der quadrierten Spur der Matrix der quadrierten Gradienten.
  Defaultwert: 0.08
  Wertevorschläge: Alpha ∈ {0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08}
  Typischer Wertebereich: 0.001 ≤ Alpha ≤ 0.1
  Minimale Schrittweite: 0.001
  Empfohlene Schrittweite: 0.01
  Restriktion: Alpha > 0.0
- **Threshold** (input_control) ..... number ⇝ *real* / integer
  Minimale Filterantwort der Punkte.
  Defaultwert: 1000.0
  Restriktion: Threshold >= 0.0
- **Subpix** (input_control) ..... string ⇝ *string*
  Subpixelgenauigkeit der Filterantworten an oder abschalten.
  Defaultwert: 'on'
  Werteliste: Subpix ∈ {'on', 'off'}

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinaten der gefundenen Punkte.

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinaten der gefundenen Punkte.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `points_harris` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`gen_cross_contour_xld`

Alternativen

`points_foerstner`, `points_harris`, `points_lepetit`, `points_sojka`

Literatur

C. Harris, M. Stephens: „A combined corner and edge detector“. Proceedings of the 4th Alvey Vision Conference, pp. 147-151, 1988.

V. Gouet, N.Boujemaa: „Object-based queries using color points of interest“. IEEE Workshop on Content-Based Access of Image and Video Libraries, CVPR/CBAIVL 2001, Hawaii, USA, 2001.

Modul

Foundation

points_lepetit (Image : : Radius, CheckNeighbor, MinCheckNeighborDiff, MinScore, Subpix : Row, Column)

Suche nach signifikanten Punkten mittels des Lepetit-Operators.

`points_lepetit` extrahiert signifikante Punkte wie z.B. Ecken aus dem Bild `Image`. Dazu wird `Image` zunächst mit einem 3x3 Medianfilter geglättet. Anschließend werden alle Grauwerte auf einem Kreis mit dem Radius `Radius` um einen Interestpunktkandidaten (m) betrachtet. Es werden die absoluten Differenzen zweier auf dem Kreis diagonal gegenüberliegenden Grauwerte (m1, m2) zu dem des zentralen Pixels m gebildet. Mindestens eine dieser Differenzen muss dabei größer als `MinCheckNeighborDiff` sein. Alle sich gegenüber liegenden Kreispunkte müssen diese Bedingung erfüllen. Um die Detektion von Punkten entlang von leicht gekrümmten Kanten (Aliasing) zu verhindern, ist es möglich, zusätzliche Differenzen von `CheckNeighbor` Kreisnachbarpixeln von m1 und m2 zum Zentrum zu bilden, die ebenfalls obiges Kriterium erfüllen. Durch Bildung aller Grauwertdifferenzen der Kreispunkte zum Zentrum wird ein mittlerer Grauwertunterschied berechnet. Dieser Wert muss größer `MinScore` sein und ermöglicht eine Restriktion auf Punkte hohen Kontrasts. Mit Hilfe der Berechnung aller mittleren Grauwertunterschiede in der Achternachbarschaft des Interestpunktes ist es möglich eine quadratische Gleichung anzupassen. Durch Ermittlung des Maximums dieser Funktion kann die Position des Interestpunktes auf Subpixelgenauigkeit verfeinert werden. Durch das Setzen des Parameters `Subpix` auf 'interpolation' (Default) oder 'none' kann dieses Verhalten an- bzw. ausgeschaltet werden. Die so ermittelten Punkte werden in `Row` und `Column` übergeben. Der Operator `points_lepetit` kann für eine besonders schnelle Extraktion signifikanter Punkte verwendet werden. Jedoch sind die extrahierten Punkte weniger Robust, als solche, die z.B. mit Hilfe von `points_harris` gefunden werden.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **Radius** (input_control) ..... integer ⇝ *integer*
  Der Radius des Kreises.
  Defaultwert: 3
  Wertevorschläge: `Radius` ∈ {3, 5, 6, 7, 8, 9, 10, 15}
- ▷ **CheckNeighbor** (input_control) ..... integer ⇝ *integer*
  Die Anzahl von untersuchten Nachbarpixeln auf dem Kreis.
  Defaultwert: 1
  Wertevorschläge: `CheckNeighbor` ∈ {1, 2, 3, 5}
- ▷ **MinCheckNeighborDiff** (input_control) ..... integer ⇝ *integer*
  Schwellenwert der Grauwertdifferenz zu jedem einzelnen Kreispunkt.
  Defaultwert: 15
  Wertevorschläge: `MinCheckNeighborDiff` ∈ {10, 15, 20, 25, 30, 35, 40, 45, 60, 80}
- ▷ **MinScore** (input_control) ..... integer ⇝ *integer*
  Schwellenwert der Grauwertdifferenz zu allen Kreispunkten.
  Defaultwert: 30
  Wertevorschläge: `MinScore` ∈ {5, 10, 15, 20, 25, 30}
- ▷ **Subpix** (input_control) ..... string ⇝ *string*
  Subpixelgenaue Punktkoordinaten.
  Defaultwert: 'interpolation'
  Werteliste: `Subpix` ∈ {'none', 'interpolation'}
- ▷ **Row** (output_control) ..... point.y-array ⇝ *integer* / real
  Zeilenkoordinaten der detektierten Punkte.
- ▷ **Column** (output_control) ..... point.x-array ⇝ *integer* / real
  Spaltenkoordinaten der detektierten Punkte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`gauss_filter`

Alternativen

`points_foerstner`, `points_harris`, `points_harris_binomial`, `points_sojka`

Modul

Foundation

points_sojka (Image : : MaskSize, SigmaW, SigmaD, MinGrad, MinApparentness, MinAngle, Subpix : Row, Column)

Findet Ecken mit dem Sojka-Operator.

`points_sojka` definiert eine Ecke als den Schnittpunkt zweier gerader, nicht kollinearer Grauwertkanten. Es wird für jeden Bildpunkt im Eingabebild `Image` eine Nachbarschaft von `MaskSize` × `MaskSize` Punkten betrachtet, um zu entscheiden, ob es sich dabei um einen Eckpunkt handeln kann. Innerhalb dieser Umgebung werden wiederum nur diejenigen Bildbereiche berücksichtigt, die für diese Entscheidung relevant sind. Pixel, in denen der Gradient nicht mindestens `MinGrad` beträgt, werden von vornherein verworfen.

Weiterhin werden von den verbliebenen Punkten nur diejenigen verwendet, die zu einer der beiden Grauwertkanten gehören, die die Ecke bilden. Dazu wird die sogenannte $Apparentness$ berechnet, die ein Maß für die Wahrscheinlichkeit ist, dass der untersuchte Punkt tatsächlich ein Eckpunkt ist. Sie wird im Wesentlichen durch die Anzahl der relevanten Punkte und die Stärke der Grauwertgradienten in ihnen bestimmt. Ein Punkt wird nur

dann als Ecke akzeptiert, wenn seine *Apparentness* mindestens MinApparentness ist. Typische Werte für MinApparentness liegen in der Größenordnung von wenigen Vielfachen von MinGrad.

Um die *Apparentness* zu berechnen, wird jeder Punkt noch nach zwei Kriterien gewichtet: Zum Einen wird der Abstand des Punktes zur (vermuteten) idealen Grauwertkante abgeschätzt und der Punkt mit einer Gaußfunktion der Größe SigmaD entsprechend diesem Abstand gewichtet. D.h. Pixel, die auf Grund der Diskretisierung des Eingabebildes von der idealen Grauwertkante weiter entfernt sind, haben einen geringeren Einfluss auf das Ergebnis, als solche mit einem geringeren Abstand. Im Allgemeinen ist es nicht notwendig, den Defaultwert *0.75* von SigmaD zu ändern. Zum Anderen wird der Einfluss eines Maskenpunkts mit einer Gaußfunktion der Größe SigmaW entsprechend seiner Entfernung zum möglichen Eckpunkt gewichtet. Um ein vernünftiges Verhältnis der Größe der Gewichtungsfunktion zur Maskengröße MaskSize zu erhalten, sollte SigmaW in etwa ein Viertel bis die Hälfte von MaskSize betragen.

Als weiteres Entscheidungskriterium wird noch der Winkel, um den sich die Richtung der Grauwertkanten im Eckpunkt verändert, berechnet. Ein Punkt kann nur als Ecke akzeptiert werden, wenn dieser Winkel größer als MinAngle ist.

Die Positionen der gefundenen Eckpunkte werden in (Row, Column) zurückgegeben. Je nach dem ob Subpix *'true'* oder *'false'* ist, werden Row und Column subpixelgenau berechnet oder nicht.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
  Eingabebild.
- ▷ **MaskSize** (input_control) ..... integer ⇝ *integer*
  Gewünschte Filtergröße.
  Defaultwert: 9
  Werteliste: MaskSize ∈ {5, 7, 9, 11, 13}
- ▷ **SigmaW** (input_control) ..... number ⇝ *real* / integer
  Sigma der Gewichtungsfunktion entsprechend der Entfernung zum möglichen Eckpunkt.
  Defaultwert: 2.5
  Wertevorschläge: SigmaW ∈ {2.0, 2.2, 2.4, 2.5, 2.6, 2.8, 3.0}
  Restriktion: 2.0 <= SigmaW && SigmaW <= 3.0
- ▷ **SigmaD** (input_control) ..... number ⇝ *real* / integer
  Sigma der Gewichtungsfunktion für den Abstand zur idealen Grauwertkante.
  Defaultwert: 0.75
  Wertevorschläge: SigmaD ∈ {0.6, 0.7, 0.75, 0.8, 0.9, 1.0}
  Restriktion: 0.6 <= SigmaD && SigmaD <= 1.0
- ▷ **MinGrad** (input_control) ..... number ⇝ *real* / integer
  Schwelle für die Stärke das Gradienten.
  Defaultwert: 30.0
  Wertevorschläge: MinGrad ∈ {20.0, 15.0, 30.0, 35.0, 40.0}
- ▷ **MinApparentness** (input_control) ..... number ⇝ *real* / integer
  Schwelle für *Apparentness*.
  Defaultwert: 90.0
  Wertevorschläge: MinApparentness ∈ {30.0, 60.0, 90.0, 150.0, 300.0, 600.0, 1500.0}
- ▷ **MinAngle** (input_control) ..... angle.rad ⇝ *real*
  Schwelle für die Richtungsänderung in einem Eckpunkt (Bogenmaß).
  Defaultwert: 0.5
  Restriktion: 0.0 <= MinAngle && MinAngle <= pi
- ▷ **Subpix** (input_control) ..... string ⇝ *string*
  Subpixelgenaue Berechnung der Eckpunkte.
  Defaultwert: 'false'
  Werteliste: Subpix ∈ {'false', 'true'}
- ▷ **Row** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der gefundenen Eckpunkte.
- ▷ **Column** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der gefundenen Eckpunkte.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `points_sojka` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Literatur

Eduard Sojka: „A New and Efficient Algorithm for Detecting the Corners in Digital Images“. Pattern Recognition, Luc Van Gool (Editor), LNCS 2449, pp. 125-132, Springer Verlag, 2002.

Modul

Foundation

10.13 Rauschen

`add_noise_distribution` `( Image : ImageNoise : Distribution : )`

Synthetisches Verrauschen eines Bildes.

`add_noise_distribution` Überlagert das Eingabebild (`Image`) mit Rauschen, das der Verteilung von `Distribution` entspricht. Es erfolgt ein Clipping der resultierenden Grauwerte auf den Bereich des entsprechenden Pixeltyps.

Die Verteilung wird in einem Tupel der Länge 513 übergeben. Die einzelnen Werte dieses Tupels legen die Häufigkeit der Pixel fest, die mit einer bestimmten Amplitude verrauscht werden. Diese Amplitude ist durch die Position des Häufigkeitswertes im Tupel festgelegt. Der mittlere Wert, d.h., der Wert an der Position 256 des Tupels legt die Häufigkeit der Pixel fest, deren Grauwert nicht verändert wird. Der Wert an der Position 255 des Tupels legt die Häufigkeit der Pixel fest, deren Grauwert um 1 verringert wird. Der Wert an der Position 254 legt die Häufigkeit der Pixel fest, deren Grauwert um 2 verringert wird, usw. Analog hierzu legt der Wert an der Position 257 die Häufigkeit der Pixel fest, deren Grauwert um 1 erhöht wird.

Wenn maximal ein Wert an einer Position kleiner als 256 und maximal ein Wert an einer Position größer als 256 ungleich Null ist, so stellt die Verteilung `Distribution` eine Salz und Pfeffer-Verteilung dar. Im Falle einer Salz und Pfeffer-Verteilung werden die verrauschten Pixel auf den minimalen (Pfeffer) bzw. maximalen (Salz) Grauwert gesetzt der in dem Ausgabebild `ImageNoise` darstellbar ist, wenn die Häufigkeit von Pfeffer durch den Wert an der Position 0 bzw. die Häufigkeit von Salz durch den Wert an der Position 512 im Tupel definiert ist.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2
Bild, das verrauscht werden soll.

▷ **`ImageNoise`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2
Verrauschtes Bild.
Parameteranzahl: ImageNoise == Image

▷ **`Distribution`** (input_control) ..... distribution.values-array ⇝ *real*
Rauschverteilung.
Parameteranzahl: 513

Beispiel

```
read_image(Image,'mreut')
dev_display (Image)
sp_distribution(30,30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
dev_display (ImageNoise)
```

Ergebnis

add_noise_distribution liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

gauss_distribution, sp_distribution, noise_distribution_mean

Alternativen

add_noise_white

Siehe auch

sp_distribution, gauss_distribution, noise_distribution_mean, add_noise_white

Modul

Foundation

add_noise_white (Image : ImageNoise : Amp :)

Synthetisches Verrauschen eines Bildes.

add_noise_white überlagert das Bild (Image) mit im Intervall [-Amp,Amp] gleichverteiltem, mittelwertfreiem, weißem Rauschen, das mittels der C-Funktion „drand48“ mit zeitabhängigem Seed generiert wird. Anschließend erfolgt ein Clipping der resultierenden Grauwerte auf den Bereich des entsprechenden Pixeltyps.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Bild, das verrauscht werden soll.

▷ **ImageNoise** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Verrauschtes Bild.
Parameteranzahl: ImageNoise == Image

▷ **Amp** (input_control) ..... real ⇝ *real*
Additives Rauschen (in [-Amp,Amp] gleichverteilt)
Defaultwert: 60.0
Wertevorschläge: Amp ∈ {1.0, 2.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
Typischer Wertebereich: $1.0 \leq \text{Amp} \leq 1000.0$
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 10.0
Restriktion: Amp > 0

Beispiel

```
read_image(Image,'fabrik')
dev_display (Image)
add_noise_white(Image,ImageNoise,90)
dev_display (ImageNoise)
```

Ergebnis

add_noise_white liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Alternativen

add_noise_distribution

Siehe auch

add_noise_distribution, noise_distribution_mean, gauss_distribution, sp_distribution

Modul

Foundation

gauss_distribution (: : Sigma : Distribution)

Erzeugung einer Gaußschen Rauschverteilung.

gauss_distribution erzeugt eine Rauschverteilung mit der Verteilung einer Gaußglocke. Dabei gibt der Parameter Sigma die Standardabweichung des Rauschens an. Das Ergebnis (Distribution) dient in der Regel als Eingabe für den Befehl add_noise_distribution.

Parameter

▷ **Sigma** (input_control) ..... real ⇝ *real*
Standardabweichung der Verteilung.
Defaultwert: 2.0
Wertevorschläge: Sigma ∈ {1.5, 2.0, 3.0, 5.0, 10.0}
Typischer Wertebereich: 0.0 ≤ Sigma ≤ 100.0
Minimale Schrittweite: 0.1
Empfohlene Schrittweite: 1.0

▷ **Distribution** (output_control) ..... distribution.values-array ⇝ *real*
Zu berechnende Gaußverteilung.
Parameteranzahl: 513

Beispiel

```
read_image(Image,'fabrik')
dev_display (Image)
gauss_distribution(30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
dev_display (ImageNoise)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

add_noise_distribution

Alternativen

sp_distribution, noise_distribution_mean

Siehe auch

sp_distribution, add_noise_white, noise_distribution_mean

Modul

Foundation

noise_distribution_mean (ConstRegion, Image : : FilterSize : Distribution)

Rauschverteilung eines Bildes bestimmen.

noise_distribution_mean berechnet die Rauschverteilung in einem Ausschnitt des Bildes Image. Der Parameter ConstRegion gibt den Bereich in dem Bild an, der nicht strukturiert ist. Die Grauwerte sollten hier also näherungsweise konstant sein. Die Abweichungen in den Grauwerten werden hier nur durch Rauschen erzeugt. Die Rauschverteilung wird bestimmt, indem das Bild mit dem Mittelwertfilter (mean_image) geglättet und dann die punktweise Differenzen der Grauwerte (Original minus Mittelwertbild) bestimmt werden. Die Häufigkeit der auftretenden Differenzen werden in dem Parameter Distribution übergeben.

Achtung

Es ist zu beachten, dass die Region in ConstRegion nicht zu dicht an einen Grauwertgradienten angrenzt, da durch die Mittelung auch die Nachbarschaft mit einbezogen wird. Konkret heißt das, dass der Abstand vom Rand zu einer Kante so groß wie die Maskengröße der Mittelung FilterSize sein muss.

Parameter

- ▷ **ConstRegine** (input_object) ..... region(-array) ⇝ *object*
  Zu analysierende Regionen mit konstanten Grauwerten.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Zugehöriges Bild.
- ▷ **FilterSize** (input_control) ..... integer ⇝ *integer*
  Filtergröße des Mittelwertfilters.
  Defaultwert: 21
  Wertevorschläge: FilterSize ∈ {5, 11, 15, 21, 31, 51, 101}
  Typischer Wertebereich: 3 ≤ FilterSize ≤ 501 (lin)
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
- ▷ **Distribution** (output_control) ..... distribution.values-array ⇝ *real*
  Rauschverteilung aller Eingaberegionen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

draw_region, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, threshold, erosion_circle, binomial_filter, gauss_filter, smooth_image, sub_image

Nachfolger

add_noise_distribution

Siehe auch

mean_image, gauss_distribution

Modul

Foundation

sp_distribution (: : PercentSalt, PercentPepper : Distribution)

Erzeugung einer Salz- und Pfeffer- Rauschverteilung.

sp_distribution erzeugt eine Rauschverteilung mit Rauschwerten von 0 und 255. Die Parameter PercentSalt und PercentPepper geben dabei an, wie viel Prozent weiße und schwarze Punkte auftreten sollen. Die Summe der Parameter muss kleiner als 100 sein. Das Ergebnis (Distribution) dient in der Regel als Eingabe für den Befehl add_noise_distribution.

Parameter

▷ **PercentSalt** (input_control) . . . . . number ⇝ *real* / integer
Prozentanteil von Salz im Rauschen.
Defaultwert: 5.0
Wertevorschläge: `PercentSalt` ∈ {1.0, 2.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0}
Typischer Wertebereich: 0.0 ≤ `PercentSalt` ≤ 100.0
Minimale Schrittweite: 0.1
Empfohlene Schrittweite: 1.0
Restriktion: 0.0 <= PercentSalt && PercentSalt <= 100.0

▷ **PercentPepper** (input_control) . . . . . number ⇝ *real* / integer
Prozentanteil von Pfeffer im Rauschen.
Defaultwert: 5.0
Wertevorschläge: `PercentPepper` ∈ {1.0, 2.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0}
Typischer Wertebereich: 0.0 ≤ `PercentPepper` ≤ 100.0
Minimale Schrittweite: 0.1
Empfohlene Schrittweite: 1.0
Restriktion: 0.0 <= PercentPepper && PercentPepper <= 100.0

▷ **Distribution** (output_control) . . . . . distribution.values-array ⇝ *real*
Zu berechnende Verteilung.
Parameteranzahl: 513

Beispiel

```
read_image(Image,'fabrik')
dev_display (Image)
sp_distribution(30,30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
dev_display (ImageNoise)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`add_noise_distribution`

Alternativen

`gauss_distribution`, `noise_distribution_mean`

Siehe auch

`gauss_distribution`, `noise_distribution_mean`, `add_noise_white`

Modul

Foundation

10.14 Sonstiges

convol_image (Image : ImageResult : FilterMask, Margin :)

Berechnung der Korrelation eines Bildes mit einer linearen Maske.

`convol_image` berechnet die Korrelation des Eingabebildes `Image` mit einer Filtermaske. Die Filtermatrix, die durch `FilterMask` definiert ist, kann dabei entweder aus einer Datei oder einem Tupel generiert werden. Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (`Margin`) gewählt werden:

Grauwert	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
'continued'	Fortsetzung der Randpunkte.
'cyclic'	zyklische Fortsetzung der Bildränder.
'mirrored'	Spiegelung der Bildpunkte an den Bildrändern.

Für alle Positionen im Bild wird die Korrelation des Bildes mit der gegebenen Filtermaske berechnet. Ein evtl. auftretender Über- bzw. Unterlauf von Grauwerten wird beschnitten. Falls Filter verwendet werden, die negative Ausgabewerte erzeugen (z.B. Ableitungsfilter), sollte das Eingabebild ein Bild vom Typ int2 oder real sein.

Der Bezugspunkt der Filtermatrix, d.h. das Pixel, das an der Position liegt, für die gerade die Korrelation berechnet wird, wird wie folgt bestimmt: Erst wird die Region aller Filterpixel bestimmt, die ein Gewicht ungleich 0 haben. Dann wird der Schwerpunkt dieser Region berechnet und gerundet. Dieser Wert ist der Bezugspunkt der Filtermatrix.

Wird in `FilterMask` ein Dateiname übergeben, wird die Filtermatrix aus einer Textdatei mit folgender Struktur eingelesen:

⟨Matrixgröße⟩
⟨Inverser Gewichtungsfaktor⟩
⟨Matrix⟩

In der ersten Zeile steht die Größe der Filtermatrix, die durch zwei Zahlen (durch Leerzeichen getrennt) angegeben wird (z.B. 3 3 für 3×3). Dabei ist die erste Zahl die Höhe der Filtermaske und die zweite Zahl ihre Breite. In der nächsten Zeile steht eine ganze Zahl, durch die das Ergebnis der Matrizenberechnung für einen Bildpunkt dividiert wird. Ab der nächsten Zeile steht die Filtermatrix, die durch ganze Zahlen (getrennt durch Leerzeichen) angegeben wird. Pro Textzeile wird eine Matrixzeile angegeben. Die Dateiendung für die Filtermatrix ist in HALCON 'fil'. Diese kann beim Aufruf weggelassen werden. Soll die Filtermaske aus einem Tupel generiert werden, muss das in `FilterMask` übergebene Tupel ebenfalls dem oben beschriebenen Format genügen, mit dem Unterschied, dass in diesem Fall kein Zeilenumbruch notwendig ist.

Angenommen, man möchte beispielsweise die Filtermaske

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

verwenden. Soll die Filtermaske aus einer Datei generiert werden, muss die Datei wie folgt aussehen:

```
3 3
16
1 2 1
2 4 2
1 2 1
```

Soll die Filtermaske dagegen aus einem Tupel generiert werden, muss für `FilterMask` folgendes Tupel übergeben werden:

[3,3,16,1,2,1,2,4,2,1,2,1]

Es ist außerdem möglich, zwei Vektoren anstelle einer Matrix in `FilterMask` zu übergeben:

[MaskHeight,MaskWidth,Weight,V1,V2]

Das äußere Produkt dieser beiden Vektoren bildet dann die Filtermatrix:

$M := V1 * V1^T$

Das heißt, obige Matrix könnte auch wie folgt übergeben werden:

[3,3,16,1,2,1,1,2,1]

Ist `FilterMask` separierbar (dies wird automatisch detektiert), benutzt `convol_image` eine spezielle Implementierung, die signifikant schneller ist als die Filterung mit nicht separierbaren Masken.

Ist *'sse2_enable'* auf *'true'* gesetzt (und dieses SIMD Instruktionsset verfügbar), werden die internen Berechnungen für 'byte' und 'real' Bilder mit SIMD Technologie durchgeführt. Ist *'sse41_enable'* auf *'true'* gesetzt (und dieses SIMD Instruktionsset verfügbar), werden die internen Berechnungen für 'int2' und 'uint2' Bilder mit SIMD Technologie durchgeführt. Ist *'avx_enable'* auf *'true'* gesetzt (und dieses SIMD Instruktionsset verfügbar), werden die internen Berechnungen für 'real' Bilder mit AVX SIMD Technologie durchgeführt.

Achtung

Es ist zu beachten, dass `convol_image` nicht die Faltung des Eingabebildes mit der gegebenen Filtermaske berechnet, sondern die Korrelation. Die Filtermaske wird also nicht gespiegelt, sondern direkt verwendet.

`convol_image` kann bei Verwendung einer quadratischen 3x3 oder 5x5 Filtermaske und der Randbehandlung *'mirrored'* auf OpenCL-Geräten ausgeführt werden.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Bilder, deren Korrelation mit der Filtermaske berechnet werden.
- ▷ **`ImageResult`** (output_object) ..... multichannel-image(-array) ⇝ *object :* byte / int2 / uint2 / real
  Ergebnis der Korrelation.
- ▷ **`FilterMask`** (input_control) ..... filename.read(-array) ⇝ *string* / integer / real
  Filtermaske als Dateiname oder Tupel.
  Defaultwert: 'sobel'
  Wertevorschläge: `FilterMask` ∈ {'sobel', 'laplace4', 'lowpas_3_3'}
  Dateiendung: `.fil`
- ▷ **`Margin`** (input_control) ..... string ⇝ *string* / integer / real
  Randbehandlung.
  Defaultwert: 'mirrored'
  Wertevorschläge: `Margin` ∈ {'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255}

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Modul

Foundation

`deviation_n ( Image : ImageDeviation : : )`

Berechnet die Standardabweichung über mehrere Kanäle.

`deviation_n` erzeugt pixelweise die Standardabweichung eines mehrkanaligen Grauwertbildes. Für jeden Koordinatenpunkt p wird zunächst der Erwartungswert als arithmetisches Mittel berechnet:

$$E = \frac{\sum g_c(p)}{n}$$

wobei $g_c(p)$ den Grauwert im Kanal c an der Stelle p bezeichnet. n ist die Anzahl der Kanäle im Bild.

Die Standardabweichung selbst wird berechnet als:

$$s = 2 \cdot \sqrt{\frac{1}{n-1} \cdot \sum (g_c(p) - E)^2}$$

Hinweis: Die berechneten Werte sind eine Schätzung der Standardabweichung aus einer Stichprobe. Der vorangestellte Faktor 2 dient der besseren Ausnutzung des Wertebereichs des Ausgabebildformats. Das Ausgabebild hat einen Grauwertkanal.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Mehrkanal-Grauwertbild.
- ▷ **ImageDeviation** (output_object) . . . . . image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Ergebnis der Berechnung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Domainebene.
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`compose2`, `compose3`, `compose4`, `compose5`, `add_channels`

Siehe auch

`mean_n`

Modul

Foundation

expand_domain_gray (InputImage : ExpandedImage : ExpansionRange :)

Expansion des Definitionsbereiches eines Bildes inklusive der Grauwerte.

`expand_domain_gray` expandiert die Grauwerte am Rand des Definitionsbereiches nach außen. Das Maß dieser Expansion wird durch den Parameter `ExpansionRange` festgelegt, der den Expansionsradius in Pixel beschreibt. Alle Filter in HALCON — die stets auf den Definitionsbereich des Bildes angewendet werden — beziehen abhängig von ihrer Filterweite auch eine bestimmte Anzahl von Grauwerten außerhalb des Definitionsbereiches mit in ihre Berechnung ein. Dies kann im Randbereich des Definitionsbereiches zu unerwünschten Effekten führen. Dieser Nachteil macht sich vor allem dann bemerkbar, wenn sich Vordergrund (Definitionsbereich) und Hintergrund des Bildes in ihrer Helligkeit stark voneinander unterscheiden. In diesem Fall kann es z.B. bei Anwendung eines Glättungsfilters zu unerwünschten Aufhellungen oder Abdunkelungen im Randbereich des Definitionsbereiches kommen. Um dies zu vermeiden, wird durch `expand_domain_gray` der Definitionsbereich im Vorfeld künstlich erweitert, indem die Grauwerte seiner Randpixel nach außen fortgesetzt werden. Mit der Expansion der Grauwerte wird auch der Definitionsbereich entsprechend erweitert, um die neu besetzten Pixel zu kennzeichnen. Daher ist es in vielen Fällen sinnvoll, nach `expand_domain_gray` ein `reduce_domain` oder `change_domain` auszuführen und anschließend die Filteroperation auf den reduzierten Definitionsbereich anzuwenden. Der Wert für `ExpansionRange` sollte so gewählt werden, dass die ganze Region, inklusive ihrer Dilatation durch die Filtermaske, initialisierte Werte hat. Üblicherweise führt dies zu Werten für `ExpansionRange` von halber Filterlänge oder mehr. In Verwendung mit dem Filter `mean_image` auf Bildern des Typs real wird empfohlen, den Wert für `ExpansionRange` wie im unten gegebenen Beispiel zu bestimmen. Zur Geschwindigkeitsoptimierung verwendet dieser Filter die Werte der Pixel im kleinsten die dilatierte Region umschließenden Rechteck.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **InputImage** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
  Eingabebild mit zu erweiterndem Definitionsbereich.
- ▷ **ExpandedImage** (output_object) . . . . . image(-array) ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
  Ausgabebild mit neu berechneten Grauwerten im erweiterten Definitionsbereich.

▷ **ExpansionRange** (input_control) ..... integer ⇝ *integer*
Radius der Grauwertexpansion in Pixel.
Defaultwert: 2
Wertevorschläge: ExpansionRange ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16}
Restriktion: ExpansionRange >= 1

Beispiel

```
read_image(Fabrik, 'fabrik')
gen_rectangle2(Rectangle_Label,243,320,-1.55,62,28)
reduce_domain(Fabrik, Rectangle_Label, Fabrik_Label)
* Character extraction without gray value expansion:
MaskSize := 31
mean_image(Fabrik_Label,Label_Mean_normal,MaskSize, MaskSize)
dyn_threshold(Fabrik_Label,Label_Mean_normal,Characters_normal,10,'dark')
dev_display(Fabrik)
dev_display(Characters_normal)
* The characters in the border region are not extracted !
stop()
* Character extraction with gray value expansion:
get_domain (Fabrik_Label, Domain)
expand_domain_gray (Fabrik_Label, Expanded_Fabrik_Label, MaskSize)
get_domain (Expanded_Fabrik_Label, Expanded_Domain)
shape_trans (Expanded_Domain, RegionRect, 'rectangle1')
difference (RegionRect, Expanded_Domain, Region_Difference)
overpaint_region (Expanded_Fabrik_Label, Region_Difference, 0, 'fill')
reduce_domain (Expanded_Fabrik_Label, Domain, Reduced_Expanded_Fabrik_Label)
mean_image (Reduced_Expanded_Fabrik_Label, Label_Mean_Expanded,\
MaskSize, MaskSize)
dyn_threshold(Fabrik_Label,Label_Mean_Expanded,\
Characters_expanded,10,'dark')
dev_display(Fabrik)
dev_display(Characters_expanded)
```

Komplexität

Sei L der Umfang des Definitionsbereiches. Die Laufzeit-Komplexität ist dann näherungsweise $O(L) * $ ExpansionRange.

Ergebnis

expand_domain_gray liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

reduce_domain

Nachfolger

reduce_domain, mean_image, dyn_threshold

Siehe auch

reduce_domain, mean_image

Modul

Foundation

gray_inside (Image : ImageDist : :)

Berechnung des niedrigstmöglichen Grauwerts auf einem beliebigen Weg zum Bildrand für jeden Bildpunkt.

gray_inside „sucht“ sich für jeden Bildpunkt den günstigsten Weg zum Bildrand, d.h. den Weg, auf dem möglichst niedrige Grauwerte zu überwinden sind. Die Differenz des Grauwertes des jeweiligen Punktes zum maximalen Grauwert auf diesem Weg wird für jeden Punkt berechnet und als Ergebnisbild geliefert. Helle Bildbereiche im Ergebnisbild bedeuten also, dass diese Bereiche (die im Originalbild typischerweise eher dunkel sind) von helleren Bereichen eingeschlossen sind. Dunkle bzw. schwarze Bereiche im Ergebnisbild bedeuten, dass zwischen ihnen und dem Bildrand nur geringfügig hellere bzw. dunklere Bildbereiche existieren (was nicht heißen muss, dass sie nur von dunkleren Bereichen umgeben sind; es reicht bereits ein kleiner dunkler „Ausgang“). Der Wert 0 (schwarz) im Ergebnisbild bedeutet, dass entweder gleich helle oder dunklere Grauwerte auf dem Weg zum Bildrand auftreten.

Für die Durchführung wird zunächst die Operation watersheds benutzt, die eine Segmentierung des Bildes in Basins und Watersheds durchführt. Wenn man sich das zu bearbeitende Bild als Grauwertgebirge vorstellt, bilden die Basins die Täler und Becken (mit niedrigeren Grauwerten) , die „Gebirgskämme“ sind die Watersheds. Zunächst werden die Watersheds jeweils benachbarten Regionen zugeteilt, so dass nur noch Basins vorhanden sind. Der Rand des Definitionsbereiches des Ausgangsbildes wird nun nach dem kleinsten Grauwert durchsucht und die Region(Basin) in der er liegt mit den Ergebniswerten belegt. Wenn der kleinste Grauwert am echten Bildrand liegt, können die Ergebniswerte sofort aus der Differenz zwischen Punkt-Grauwert und diesem berechnet werden. Liegt der gefundene kleinste Grauwert jedoch innerhalb des Bildes, muss der bereits existierende Ergebniswert aus der Nachbarregion besorgt werden um mit seiner Hilfe die neuen Ergebniswerte zu berechnen. Die Suche nach der Nachbarregion erfolgt mit 8-er Nachbarschaft. Nach Abzug der gefundenen Region vom Suchbereich erfolgt ein neuer Durchgang u.s.w. So wird Basin für Basin das Bild von außen nach innen „geschält“.

Wie auch bei watersheds empfiehlt es sich, vor gray_inside eine Glättung (z.B. binomial_filter oder gauss_filter) des Bildes vorzunehmen, um die Entstehung zu vieler Einzelregionen im watersheds-Algorithmus (und dadurch sehr lange Laufzeiten) zu vermeiden.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Zu bearbeitendes Bild.
- ▷ **ImageDist** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* int2
  Ergebnisbild.

Beispiel

```
read_image(Image,'fabrik')
gauss_filter (Image,GaussImage,11)
gray_inside(GaussImage,ImageOut)
dev_display(ImageOut)
```

Ergebnis

gray_inside liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

binomial_filter, gauss_filter, smooth_image, mean_image, median_image

Nachfolger

select_shape, area_center, count_obj

Siehe auch

watersheds

Modul

Foundation

gray_skeleton (Image : GraySkeleton : :)

Verdünnung von Grauwertbildern.

gray_skeleton führt eine Grauwert-Verdünnung mittels lokalem Schnitt durch. Bildlich gesprochen wird dabei das Grauwertgebirge auf die Gratlinien reduziert, indem die Bergflanken mit dem Grauwert der jeweiligen Talsohle besetzt werden. Die resultierenden „Maximumslinien“ sind nur mehr maximal zwei Pixel dick. Die Prozedur ist insbesondere zur Verdünnung von Kantenbildern gedacht und insofern eine Alternative zu nonmax_suppression_amp. gray_skeleton erhält im Gegensatz zu nonmax_suppression_amp Konturen, ist allerdings auch ungleich zeitaufwendiger. Im Gegensatz zu skeleton verändert die Routine die Grauwerte der Bilder, lässt aber deren Regionen unverändert.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Bild, das verdünnt werden soll.
- ▷ **GraySkeleton** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Verdünntes Bild.

Beispiel

```
* Seeking leafs of a beech tree in an aerial picture:
read_image(Image,'forest')
gray_skeleton(Image,Skelett)
mean_image(Skelett,MeanSkelett,7,7)
dyn_threshold(Skelett,MeanSkelett,Leafs,3,'light')
```

Ergebnis

gray_skeleton liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Nachfolger

mean_image

Alternativen

nonmax_suppression_amp, nonmax_suppression_dir, local_max

Siehe auch

skeleton, gray_dilation_rect

Modul

Foundation

lut_trans (Image : ImageResult : Lut :)

Transformation eines Bildes mit einer Grauwerttabelle.

lut_trans transformiert ein Bild Image mittels einer Tabelle Lut. Die Tabelle funktioniert dabei wie eine Transformationsfunktion. Für byte-Bilder muss Lut ein Tupel der Länge 256 sein. Für int2-Bilder muss Lut ein Tupel der Länge 256 <= Länge <= 65536 sein. Falls die Länge von Lut <= 32768 ist, wird die Transformation nur auf den positiven Grauwerten durchgeführt, d.h., das erste Element von Lut spezifiziert den neuen Grauwert für den Eingabegrauwert 0. Falls Lut länger ist als 32768, so muss die Länge von Lut genau 65536 betragen. In diesem Fall werden sowohl die positiven als auch die negativen Werte transformiert, wobei das erste Element den neuen Grauwert für den Eingabegrauwert -32768 spezifiziert, während das letzte Element den neuen Grauwert für den Eingabegrauwert 32767 spezifiziert. In jedem Fall werden diejenigen Ausgabegrauwerte, deren Eingabegrauwerte außerhalb des gültigen Bereiches von Lut liegen, auf 0 gesetzt. Für uint2-Bilder muss Lut ein Tupel der Länge 256 <= Länge <= 65536 sein. Grauwerte außerhalb des gültigen Bereichs von Lut werden auf 0 gesetzt.

Achtung

lut_trans kann auf OpenCL-Geräten ausgeführt werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Bild, dessen Grauwerte transformiert werden sollen.
- ▷ **ImageResult** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Ergebnisbild.
- ▷ **Lut** (input_control) ..... integer-array ⇝ *integer*
  Tabelle, die die Transformation angibt.

Beispiel

```
* Apply a gamma correction to an image.
Gamma := 2.2
GammaLUT := []
for G := 0 to 255 by 1
    GammaLUT := [GammaLUT,round(255.0*pow(G/255.0,1.0/Gamma))]
endfor
read_image (Image, 'mreut')
lut_trans (Image, ImageGamma, GammaLUT)
```

Ergebnis

lut_trans liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Modul

Foundation

symmetry (Image : ImageSymmetry : MaskSize, Direction, Exponent :)

Grauwertsymmetrie entlang einer Zeile.

`symmetry` berechnet die Symmetrie entlang einer Zeile. Dabei werden für jeden Bildpunkt die Grauwerte auf beiden „Seiten“ entlang der vorgegebenen Suchrichtung verglichen: Es wird jeweils der Betrag der Differenz zwischen zwei gleich weit entfernten Punkten bestimmt. Jeder dieser Differenzen wird mit dem Exponenten gewichtet (nach einer Division durch 255) und dann wird die Summe all dieser Differenzen gebildet.

$$sym := 255 - \frac{255}{\texttt{MaskSize}} \sum_{i=1}^{\texttt{MaskSize}} \left(\frac{|g(i) - g(-i)|}{255} \right)^{\texttt{Exponent}}$$

Bildpunkte mit einer hohen Symmetrie erhalten einen hohen Grauwert.

Achtung

Gegenwärtig können nur horizontale Suchlinien verwendet werden. Beachten Sie, dass der Parameter `Direction` im Hinblick auf zukünftige Erweiterungen existiert und daher im Moment nur den Wert 0.0 haben kann.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Eingabebild.
- ▷ **`ImageSymmetry`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Symmetriebild.
- ▷ **`MaskSize`** (input_control) ..... number ⇝ *integer*
  Länge des Suchbereiches.
  Defaultwert: 40
  Wertevorschläge: `MaskSize` ∈ {3, 5, 7, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 100, 120, 140, 180}
  Typischer Wertebereich: 3 ≤ `MaskSize` ≤ 1000
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 2
- ▷ **`Direction`** (input_control) ..... number ⇝ *real*
  Winkel der Untersuchungsrichtung.
  Defaultwert: 0.0
  Wertevorschläge: `Direction` ∈ {0.0}
  Typischer Wertebereich: 0.0 ≤ `Direction` ≤ 0.0
- ▷ **`Exponent`** (input_control) ..... number ⇝ *real*
  Exponent für Gewichtung.
  Defaultwert: 0.5
  Wertevorschläge: `Exponent` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich: 0.05 ≤ `Exponent` ≤ 1.0
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
  Restriktion: 0 < Exponent && Exponent <= 1

Beispiel

```
read_image(Image,'monkey')
symmetry(Image,ImageSymmetry,70,0.0,0.5)
threshold(ImageSymmetry,SymmPoints,170,255)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `symmetry` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`threshold`

Modul

Foundation

`topographic_sketch ( Image : Sketch : : )`

Topographic primal sketch eines Grauwertbildes.

Die Funktion `topographic_sketch` berechnet aus dem Grauwertbild `Image` den „topographic primal sketch“. Dabei wird die Umgebung der einzelnen Bildpunkte mit einem bikubischen Polynom approximiert („facet model“). Aus diesen Daten werden die ersten und zweiten Richtungsableitungen an der Pixelposition berechnet, und daraus eine Einteilung des Pixels in eine von 11 Klassen hergeleitet. Die Klassen werden im Ausgabebild `Sketch` als Zahlen zwischen 1 und 11 zurückgegeben. Die einzelnen Klassen lauten wie folgt:

Peak	1
Pit	2
Ridge	3
Ravine	4
Saddle	5
Flat	6
Hillside Slope	7
Hillside Convex	8
Hillside Concave	9
Hillside Saddle	10
Hillside Inflection	11

Um die einzelnen Klassen als Regionen zu erhalten ist auf dem Labelbild Sketch noch eine Schwellenwertoperation mit entsprechendem Schwellenwert auszuführen.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
Das Bild, für das der topographic primal sketch berechnet werden soll.

▷ **`Sketch`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
Das Labelbild mit der Klasseneinteilung.

Beispiel

```
* To extract the Ridges from a Image
read_image(Image,'sinus')
topographic_sketch(Image,Sketch)
threshold(Sketch,Ridges,3,3)
```

Komplexität

Sei n die Anzahl der Bildpunkte im Eingabebild. Dann werden $O(n)$ Operationen benötigt.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `topographic_sketch` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system`

('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Nachfolger

threshold

Literatur

R. Haralick, L. Shapiro: „Computer and Robot Vision, Volume I“; Reading, Massachusetts, Addison-Wesley; 1992; Kapitel 8.13.

Modul

Foundation

10.15 Szenenfluss

```
scene_flow_calib ( ImageRect1T1, ImageRect2T1, ImageRect1T2,
    ImageRect2T2, Disparity : : SmoothingFlow, SmoothingDisparity,
    GenParamName, GenParamValue, CamParamRect1, CamParamRect2,
    RelPoseRect : ObjectModel3D )
```

Berechnung des kalibrierten Szenenflusses zwischen zwei Stereobildpaaren.

scene_flow_calib berechnet den kalibrierten Szenenfluss zwischen zwei rektifizierten aufeinanderfolgenden Stereobildpaaren. Der Szenenfluss ist die dreidimensionale Position und Bewegung von Oberflächenpunkten in einer dynamischen Szene. Die Bewegung in den Bildern kann dadurch verursacht werden, dass sich Objekte oder die Kamera (oder beides) in der Welt zwischen den Aufnahmen der beiden Stereobilder bewegen.

Der Szenenfluss wird in dem 3D-Objektmodell ObjectModel3D zurückgegeben. Das 3D-Objektmodell enthält die Koordinaten der rekonstruierten 3D-Punkte. Zusätzlich wird der 3D-Fluss in dem 3D-Objektmodell durch die Attribute *'&flow_x'*, *'&flow_y'* und *'&flow_z'* kodiert.

Die aufeinanderfolgenden Stereobildpaare werden in ImageRect1T1, ImageRect2T1, ImageRect1T2 und ImageRect2T2 übergeben. Jedes Stereobildpaar muss rektifiziert sein. Die Bildpaare können mit den Operatoren calibrate_cameras, gen_binocular_rectification_map und map_image rektifiziert werden.

Die Kamerageometrie des rektifizierten binokularen Kamerasystems wird durch die internen Kameraparameter CamParamRect1 der rektifizierten Kamera 1 und CamParamRect2 der rektifizierten Kamera 2 und die Pose RelPoseRect bestimmt, welche die Lage des rektifizierten Kamerasystems 2 bezüglich des rektifizierten Kamerasystems 1 beschreibt. Die Werte für diese Kameraparameter können mit den Operatoren calibrate_cameras und gen_binocular_rectification_map bestimmt werden. Die Brennweite und Pixelgröße der rektifizierten Kamerasysteme 1 und 2 müssen gleich sein.

Außerdem wird ein einkanaliges Disparitätsbild Disparity benötigt, welches für jedes Pixel $(r, c1)$ des Bildes ImageRect1T1 ein homologes Pixel $(r, c2)$ im Bild ImageRect2T1 festlegt. Hierbei gilt $c2 = c1 + d(r, c1)$, wobei $d(r, c)$ die Disparität in Disparity am Pixel (r, c) ist. Die Disparität kann mit binocular_disparity oder binocular_disparity_mg berechnet werden.

Um den kalibrierten Szenenfluss zu berechnen, wird intern zunächst scene_flow_uncalib ausgeführt. Die Ergebnisse werden dann unter Verwendung der oben beschriebenen Stereokamera-Geometrie-Parameter in 3D-Punkte und 3D-Flussvektoren umgerechnet.

Für eine Beschreibung der weiteren Parameter von scene_flow_calib wird auf scene_flow_uncalib verwiesen.

Parameter

- ▷ **ImageRect1T1** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 1 zum Zeitpunkt t_1.
- ▷ **ImageRect2T1** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 2 zum Zeitpunkt t_1.
- ▷ **ImageRect1T2** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 1 zum Zeitpunkt t_2.
- ▷ **ImageRect2T2** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 2 zum Zeitpunkt t_2.
- ▷ **Disparity** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* real
  Disparität zwischen Eingabebild 1 und 2 zum Zeitpunkt t_1.
- ▷ **SmoothingFlow** (input_control) . . . . . number ⇝ *real* / integer
  Gewicht des Regularisierungsterms relativ zum Datenterm (Ableitungen des optischen Flusses).
  Defaultwert: 40.0
  Wertevorschläge: `SmoothingFlow` ∈ {10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0}
  Restriktion: SmoothingFlow > 0.0
- ▷ **SmoothingDisparity** (input_control) . . . . . number ⇝ *real* / integer
  Gewicht des Regularisierungsterms relativ zum Datenterm (Ableitungen der Änderung der Disparität).
  Defaultwert: 40.0
  Wertevorschläge: `SmoothingDisparity` ∈ {10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0}
  Restriktion: SmoothingDisparity > 0.0
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Parametername(n) für den Algorithmus.
  Defaultwert: 'default_parameters'
  Wertevorschläge: `GenParamName` ∈ {'default_parameters', 'warp_levels', 'warp_zoom_factor', 'warp_last_level', 'outer_iter', 'inner_iter', 'sor_iter', 'omega'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
  Parameterwert(e) für den Algorithmus.
  Defaultwert: 'accurate'
  Wertevorschläge: `GenParamValue` ∈ {'very_accurate', 'accurate', 'fast', 'very_fast', 0, 1, 2, 3, 4, 5, 6, 0.5, 0.6, 0.7, 0.75, 3, 5, 7, 2, 3, 1.9}
- ▷ **CamParamRect1** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter der projektiven Kamera 1.
- ▷ **CamParamRect2** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter der projektiven Kamera 2.
- ▷ **RelPoseRect** (input_control) . . . . . pose ⇝ *real* / integer
  Lage der zweiten Kamera bzgl. der ersten.
  Parameteranzahl: 7
- ▷ **ObjectModel3D** (output_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle des erstellten 3D-Objektmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `scene_flow_calib` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`binocular_disparity`, `binocular_disparity_mg`

Alternativen

scene_flow_uncalib, optical_flow_mg

Literatur

A. Wedel, C. Rabe, T. Vaudrey, T. Brox, U. Franke and D. Cremers: „Efficient dense scene flow from sparse or dense stereo data“; In: Proceedings of the 10th European Conference on Computer Vision: Part I, pages 739-751. Springer-Verlag, 2008.

Modul

Foundation

scene_flow_uncalib (ImageRect1T1, ImageRect2T1, ImageRect1T2, ImageRect2T2, Disparity : OpticalFlow, DisparityChange : SmoothingFlow, SmoothingDisparity, GenParamName, GenParamValue :)

Berechnung des unkalibrierten Szenenflusses zwischen zwei Stereobildpaaren.

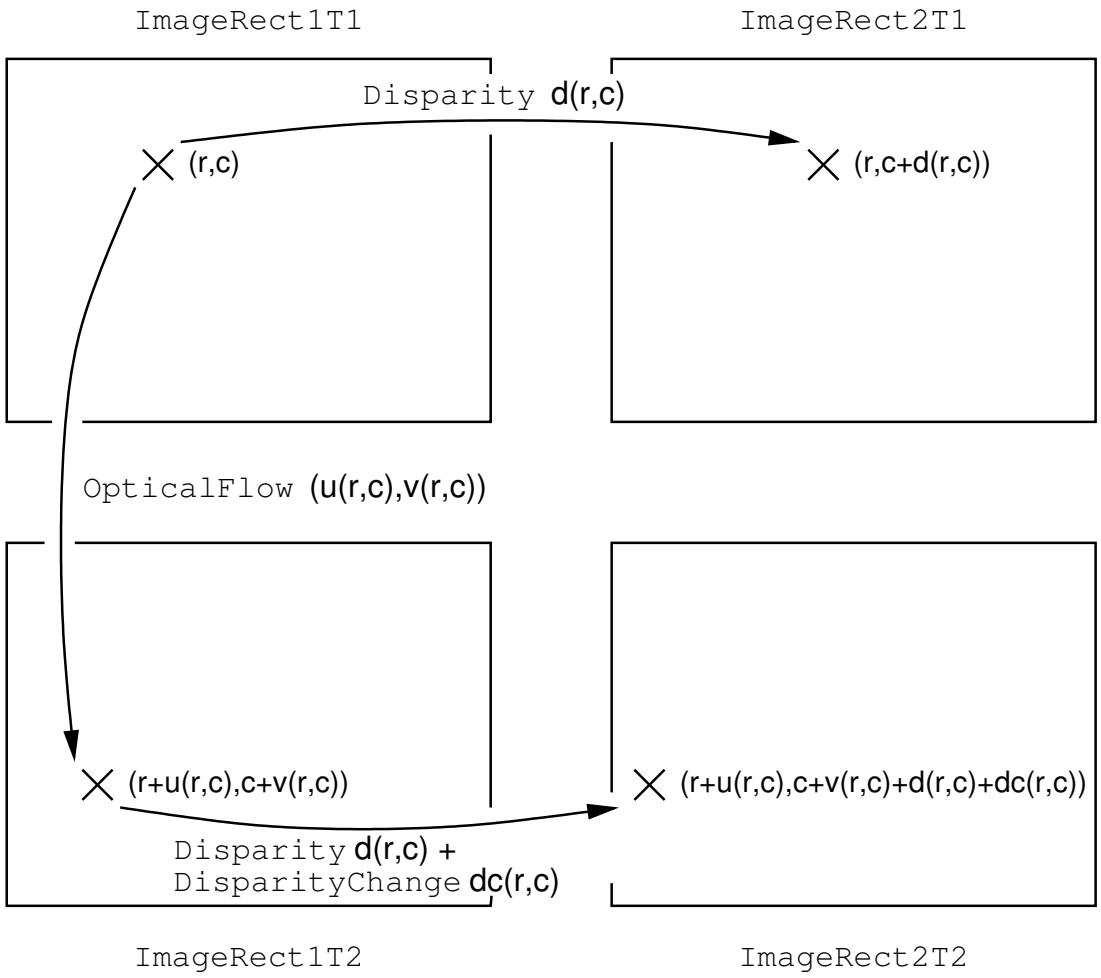
scene_flow_uncalib berechnet den unkalibrierten Szenenfluss zwischen zwei aufeinanderfolgenden rektifizierten Stereobildpaaren. Der Szenenfluss ist die dreidimensionale Position und Bewegung von Oberflächenpunkten in einer dynamischen Szene. Die Bewegung in den Bildern kann dadurch verursacht werden, dass sich Objekte oder die Kamera (oder beides) in der Welt zwischen den Aufnahmen der beiden Stereobilder bewegen. Um den kalibrierten Szenenflusses zu berechnen, kann scene_flow_calib verwendet werden.

Die aufeinanderfolgenden Stereobildpaare werden in ImageRect1T1, ImageRect2T1, ImageRect1T2 und ImageRect2T2 übergeben. Jedes Stereobildpaar muss rektifiziert sein. Die Bildpaare können mit den Operatoren calibrate_cameras, gen_binocular_rectification_map und map_image rektifiziert werden. Außerdem wird ein einkanaliges Disparitätsbild Disparity benötigt, welches für jedes Pixel $(r, c1)$ des Bildes ImageRect1T1 ein homologes Pixel $(r, c2)$ im Bild ImageRect2T1 festlegt. Hierbei gilt $c2 = c1 + d(r, c1)$, wobei $d(r, c)$ die Disparität in Disparity am Pixel (r, c) ist. Die Disparität kann mit binocular_disparity oder binocular_disparity_mg berechnet werden.

Der berechnete unkalibrierte Szenenfluss wird in OpticalFlow und DisparityChange zurückgegeben. Die Vektoren im Vektorfeld OpticalFlow beschreiben die Bewegung in der Bildebene zwischen ImageRect1T1 und ImageRect1T2. Das einkanalige Bild DisparityChange beschreibt die Änderung der Disparität zwischen ImageRect2T1 und ImageRect2T2. Ein Punkt in der Welt wird in das Bild ImageRect1T1 an der Position (r, c) projiziert. Derselbe Punkt wird projiziert in

- ImageRect2T1 an Position $(r, c + d(r, c))$,
- ImageRect1T2 an Position $(r + u(r, c), c + v(r, c))$,
- ImageRect2T2 an Position $(r + u(r, c), c + v(r, c) + d(r, c) + dc(r, c))$.

Hierbei bezeichnen $u(r, c)$ und $v(r, c)$ die Zeilen- und Spaltenkoordinaten des Vektorfeldes in OpticalFlow, $d(r, c)$ die Disparitäten in Disparity und $dc(r, c)$ die Disparitätsänderungen in DisparityChange, jeweils am Pixel (r, c).



Relationen zwischen den vier Bildern, dem optischen Fluss, sowie den Disparitäten in den Disparitätsbildern.

Parameterbeschreibung

Die rektifizierten Eingabebilder werden in `ImageRect1T1`, `ImageRect2T1`, `ImageRect1T2` und `ImageRect2T2` übergeben. Die Berechnung des Szenenflusses erfolgt auf der Region von `ImageRect1T1`. Dies ist auch die Region des Szenenflusses in `OpticalFlow` und `DisparityChange`. `Disparity` beinhaltet die Disparität zwischen den rektifizierten Bildern `ImageRect1T1` und `ImageRect2T1`.

`SmoothingFlow` und `SmoothingDisparity` spezifizieren die Regularisierungsgewichte λ und γ relativ zum Datenterm. Je größer diese Parameter sind, desto glatter ist der berechnete Szenenfluss. Für Byte-Bilder mit einem Wertebereich von $[0, 255]$ führen Werte in der Größenordnung von 40 typischerweise zu guten Ergebnissen.

Die Parameter für das Iterationsverfahren und für die Coarse-to-Fine-Warping-Strategie können mit den generischen Parametern `GenParamName` und `GenParamValue` festgelegt werden.

Im Normalfall ist es ausreichend, einen der Standardparametersätze für die Parameter zu verwenden, indem `GenParamName` = *'default_parameters'* und `GenParamValue` = *'very_accurate'*, *'accurate'*, *'fast'* oder *'very_fast'* übergeben werden. Falls notwendig, können einzelne Parameter modifiziert werden, nachdem der Standardparametersatz gewählt wurde, indem eine Untermenge der Parameter und entsprechenden Werte nach *'default_parameters'* in `GenParamName` und `GenParamValue` übergeben werden (z.B. `GenParamName` = *['default_parameters','warp_zoom_factor']* und `GenParamValue` = *['accurate',0.6]*). Die Bedeutung der einzelnen Parameter ist unten detailliert beschrieben. Die Standardparametersätze sind gegeben durch:

'default_parameters'	*'very_accurate'*	*'accurate'*	*'fast'*	*'very_fast'*
'warp_zoom_factor'	0.75	0.5	0.5	0.5
'warp_levels'	0	0	0	0
'warp_last_level'	1	1	1	2
'outer_iter'	10	7	5	4
'inner_iter'	2	2	2	2
'sor_iter'	3	3	3	3
'omega'	1.9	1.9	1.9	1.9

Falls die Parameter einzeln spezifiziert werden sollen, müssen `GenParamName` und `GenParamValue` Tupel derselben Länge enthalten. Die zu den Parametern in `GenParamName` gehörigen Werte müssen an der entsprechenden Position in `GenParamValue` spezifiziert werden. Um ein genaueres Verständnis der Parameter zu erlangen, wird auf den Abschnitt *Algorithmus* unten verwiesen.

- Mit `GenParamName` = *'warp_zoom_factor'* kann das Auflösungsverhältnis zwischen zwei aufeinanderfolgenden Warpingebenen in der Coarse-to-Fine-Warping-Hierarchie festgelegt werden. Dabei muss *'warp_zoom_factor'* im offenen Intervall $(0, 1)$ gewählt werden. Aus Performanzgründen wird üblicherweise *'warp_zoom_factor'* = 0.5 gewählt, d.h. die Pixelanzahl der Eingangsbilder wird von einer Warpingebene zur nächsten in jede Richtung halbiert. Werte für *'warp_zoom_factor'* näher bei 1 können zu qualitativ leicht besseren Ergebnissen führen, benötigen aber unverhältnismäßig mehr Rechenzeit.
- Mit `GenParamName` = *'warp_levels'* kann die Warpinghierarchie auf eine maximale Anzahl von Ebenen beschränkt werden. Für *'warp_levels'* = 0 wird die maximal mögliche Anzahl von Ebenen gewählt. Sollte die Bildgröße bei einem gegebenen Auflösungsverhältnis *'warp_zoom_factor'* die gewählte Anzahl an Ebenen nicht zulassen, wird die maximal mögliche Anzahl an Ebenen verwendet. Im Normalfall sollte *'warp_levels'* auf 0 gesetzt werden.
- Mit `GenParamName` = *'warp_last_level'* kann die Anzahl der Warpingebenen, für die *kein* Flussinkrement mehr berechnet werden soll, festgelegt werden. Normalerweise wird diese Anzahl entweder auf 1 oder 2 gesetzt, d.h. es wird entweder auf jeder Warpingebene ein Flussinkrement berechnet, oder die feinste Warpingebene wird bei der Berechnung ausgelassen. Im letzten Fall wird die Lösung nur auf halber Auflösung gerechnet wird und dann auf die volle Auflösung interpoliert.
- Mit `GenParamName` = *'outer_iter'* kann die Anzahl äußerer Iterationen des Minimierungsalgorithmus festgelegt werden. Typischerweise werden die Ergebnisse genauer, je größer *'outer_iter'* gewählt wird. Größere Werte dieses Parameters führen zu höheren Laufzeiten. Typischerweise wird *'outer_iter'* auf Werte zwischen 5 und 10 gesetzt.
- Mit `GenParamName` = *'inner_iter'* kann die Anzahl innerer Iterationen des Minimierungsalgorithmus festgelegt werden. Typischerweise werden die Ergebnisse genauer, je größer *'inner_iter'* gewählt wird. Größere Werte dieses Parameters führen zu höheren Laufzeiten. Typischerweise sind zwei innere Iterationen ausreichend.
- Mit `GenParamName` = *'sor_iter'* kann die Anzahl der SOR-Iterationen, die zur Lösung des linearen Gleichungssystems verwendet werden, festgelegt werden. Typischerweise werden die Ergebnisse genauer, je größer *'sor_iter'* gewählt wird. Größere Werte dieses Parameters führen zu höheren Laufzeiten. Typischerweise sind drei SOR-Iterationen ausreichend.
- Mit `GenParamName` = *'omega'* kann der Überrelaxationsfaktor ω des SOR-Algorithmus festgelegt werden. *'omega'* muss im offenen Intervall $(1, 2)$ gewählt werden. Typischerweise wird *'omega'* auf 1.9 gesetzt.

Algorithmus

Der Szenenfluss wird berechnet, indem ein geeignetes Energiefunktional minimiert wird:

$$E(f) = \int E_D(f) + E_S(f)\,\mathrm{d}r\mathrm{d}c.$$

Hierbei stellt $f = (u, v, dc)$ das optische Flussfeld und die Disparitätsänderung dar.$E_D(f)$ bezeichnet den Datenterm und $E_S(f)$ den Glattheitsterm (Regularisierungsterm). Der Algorithmus basiert auf folgenden Annahmen, die zu den Daten- und Glattheitstermen führen:

Helligkeitskonstanz Es wir angenommen, dass der Grauwert eines Punktes in allen vier Bildern konstant bleibt. Daraus folgen die folgenden vier Bedingungen:

$$\begin{aligned}
C_{D1} &= I_{1,t1}(r,c) - I_{2,t1}(r, c + d(r,c)) = 0 \\
C_{F1} &= I_{1,t2}(r + u(r,c), c + v(r,c)) - I_{1,t1}(r,c) = 0 \\
C_{F2} &= I_{2,t2}(r + u(r,c), c + v(r,c) + d(r,c) + dc(r,c)) - I_{2,t1}(r, c + d(r,c)) = 0 \\
C_{D2} &= I_{2,t2}(r + u(r,c), c + v(r,c) + d(r,c) + dc(r,c)) - I_{1,t2}(r + u(r,c), c + v(r,c)) = 0
\end{aligned}$$

Hierbei bezeichnen $I_{1,t1}$, $I_{2,t1}$, $I_{1,t2}$ und $I_{2,t2}$ die Bilder `ImageRect1T1`, `ImageRect2T1`, `ImageRect1T2` und `ImageRect2T2`.

Stückweise Glattheit des Szenenflusses Es wird angenommen, dass die Lösung stückweise glatt ist. Die Glattheit wird erreicht, indem die ersten Ableitungen des Flusses $\nabla(u)^2, \nabla(v)^2, \nabla(dc)^2$ bestraft werden. Die Verwendung einer statistisch robusten (linearen) Straffunktion $\Psi(s) = \sqrt{s^2 + \varepsilon}$ mit $\varepsilon = 0.001$ sorgt dafür, dass die Kanten in den Bewegungen im Szenenfluss erhalten bleiben.

Da das Disparitätsbild d gegeben ist, kann die erste Bedingung C_{D1} vernachlässigt werden. Unter Verwendung aller obigen Annahmen kann das Energiefunktional wie folgt geschrieben werden:

$$E(f) = \int \Psi(C_{F1}^2) + \Psi(C_{F2}^2) + \Psi(C_{D2}^2) + \Psi(\lambda(\nabla(u)^2 + \nabla(v)^2) + \gamma\nabla(dc)^2)\,\mathrm{d}r\mathrm{d}c$$

Hierbei sind λ und γ die Regularisierungsparameter, die in `SmoothingFlow` und `SmoothingDisparity` übergeben werden.

Um große Verschiebungen schätzen zu können, machen Coarse-to-Fine-Warping-Strategien von zwei Konzepten Gebrauch, die beide eng miteinander verzahnt sind: Die sukzessive Verfeinerung der Problemstellung (Coarse-to-Fine) und die fortlaufende Kompensation des aktuelles Bildpaares um bereits berechnete Verschiebungen (Warping). Algorithmisch lässt sich eine solche Coarse-to-Fine-Warping-Strategie wie folgt formulieren:

1. Zunächst werden beide Bilder des aktuellen Bildpaares auf eine sehr grobe Auflösungsstufe heruntergesampelt.
2. Dann wird der Szenenfluss auf dieser groben Auflösung berechnet.
3. Dieser Szenenfluss wird auf der nächstfeineren Auflösungsstufe benötigt: Es wird dort aus dem zweiten Bildpaar der Bildfolge herausgerechnet, d.h. die Problemstellung auf der feineren Auflösungsstufe wird um das bereits berechnete Flussfeld kompensiert. Dieser Schritt wird auch als Warping bezeichnet.
4. Das so modifizierte Problem (Differenzproblem) wird nun auf der feineren Auflösungsstufe gelöst, d.h. der Szenenfluss wird dort berechnet.
5. Die Schritte 3-4 werden wiederholt, bis die ursprüngliche Auflösung erreicht wird.
6. Das letztendliche Ergebnis wird durch Addition des Szenenflusses aller Auflösungsstufen gebildet.

Diese inkrementelle Berechnung des Szenenflusses bietet folgenden Vorteil: Während das Coarse-to-Fine-Konzept sicherstellt, dass die Verschiebungen auf der gröbsten Auflösungsstufe sehr klein sind, sorgt die Warping-Strategie dafür, dass das auch für die zu berechnenden Flussinkremente (Szenenfluss der Differenzprobleme) so bleibt. Da kleine Verschiebungen viel genauer berechnet werden können als größere, nimmt durch den Einsatz einer solchen Coarse-to-Fine-Warping-Strategie die Schätzqualität im allgemeinen deutlich zu. Jedoch muss anstelle eines einzelnen Korrespondenzproblems eine Hierarchie von ebensolchen Problemen gelöst werden.

Die Minimierung von Funktionalen ist aus mathematischer Sicht sehr eng mit der Minimierung von Funktionen verwandt: So wie eine Nullstelle der ersten Ableitung eine notwendige Bedingung für ein Minimum einer Funktion ist, ist die Erfüllung der sogenannten Euler-Lagrange-Gleichungen eine notwendige Bedingung für die minimierende Funktion eines Funktionals (die minimierende Funktion entspricht hier dem gesuchten Szenenfluss). Die Euler-Lagrange-Gleichungen sind partielle Differentialgleichungen. Durch die Diskretisierung dieser Euler-Lagrange-Gleichungen mit Hilfe finiter Differenzen entstehen große dünn besetzte nichtlineare Gleichungssysteme, die durch den Algorithmus gelöst werden müssen.

Auf jeder Warpingebene muss ein einzelnes Gleichungssystem gelöst werden. Der Algorithmus verwendet ein iteratives Verfahren, dass aus zwei verschachtelten Iterationen (den sog. äußeren und inneren Iterationen) sowie dem

SOR-Verfahren (Successive Over-Relaxation) besteht. Die äußere Schleife sorgt für die Linearisierung der nichtlinearen Terme, die aus den Datentermen folgen. Die Nichtlinearität von Ψ' wird durch die innere Fixpunktiteration beseitigt. Das resultierende lineare Gleichungssystem wird effizient mit dem SOR-Verfahren gelöst.

Parameter

- ▷ **ImageRect1T1** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 1 zum Zeitpunkt t_1.
- ▷ **ImageRect2T1** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 2 zum Zeitpunkt t_1.
- ▷ **ImageRect1T2** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 1 zum Zeitpunkt t_2.
- ▷ **ImageRect2T2** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild 2 zum Zeitpunkt t_2.
- ▷ **Disparity** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* real
  Disparität zwischen Eingabebild 1 und 2 zum Zeitpunkt t_1.
- ▷ **OpticalFlow** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* vector_field
  Berechneter optischer Fluss.
- ▷ **DisparityChange** (output_object) . . . . . singlechannelimage(-array) ⇝ *object :* real
  Berechnete Änderung der Disparität.
- ▷ **SmoothingFlow** (input_control) . . . . . number ⇝ *real* / integer
  Gewicht des Regularisierungsterms relativ zum Datenterm (Ableitungen des optischen Flusses).
  Defaultwert: 40.0
  Wertevorschläge: `SmoothingFlow` ∈ {10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0}
  Restriktion: SmoothingFlow > 0.0
- ▷ **SmoothingDisparity** (input_control) . . . . . number ⇝ *real* / integer
  Gewicht des Regularisierungsterms relativ zum Datenterm (Ableitungen der Änderung der Disparität).
  Defaultwert: 40.0
  Wertevorschläge: `SmoothingDisparity` ∈ {10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0}
  Restriktion: SmoothingDisparity > 0.0
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Parametername(n) für den Algorithmus.
  Defaultwert: 'default_parameters'
  Wertevorschläge: `GenParamName` ∈ {'default_parameters', 'warp_levels', 'warp_zoom_factor', 'warp_last_level', 'outer_iter', 'inner_iter', 'sor_iter', 'omega'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
  Parameterwert(e) für den Algorithmus.
  Defaultwert: 'accurate'
  Wertevorschläge: `GenParamValue` ∈ {'very_accurate', 'accurate', 'fast', 'very_fast', 0, 1, 2, 3, 4, 5, 6, 0.5, 0.6, 0.7, 0.75, 3, 5, 7, 2, 3, 1.9}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `scene_flow_uncalib` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`binocular_disparity`, `binocular_disparity_mg`

Nachfolger

`threshold`, `vector_field_length`

Alternativen

`scene_flow_calib`, `optical_flow_mg`

Literatur

A. Wedel, C. Rabe, T. Vaudrey, T. Brox, U. Franke and D. Cremers: „Efficient dense scene flow from sparse or dense stereo data“; In: Proceedings of the 10th European Conference on Computer Vision: Part I, pages 739-751. Springer-Verlag, 2008.

Modul

Foundation

10.16 Texturinspektion

deviation_image (Image : ImageDeviation : Width, Height :)

Standardabweichung der Grauwerte in Rechteckfenstern.

deviation_image transformiert die Grauwerte der Eingabebilder aus Image mit Hilfe einer Filtermaske (Height, Width), in der die Standardabweichung der Grauwerte berechnet wird. Um den Grauwertbereich des Ergebnisbildes besser auszunutzen, wird das Ergebnis mit 2 multipliziert. Das Ergebnis wird in ImageDeviation übergeben. Die Steuerparameter Height, Width werden, falls sie jeweils einen geraden Wert haben, zu dem nächstgrößeren ungeraden Wert gewandelt. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Achtung

deviation_image kann auf OpenCL-Geräten ausgeführt werden. Da der gleiche OpenCL Code verwendet wird, gelten die selben Einschränkungen wie für mean_image. Das Ergebnis kann geringfügig von dem der CPU abweichen, da intern mit einfach genauer Fließkommaarithmetik gerechnet wird.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int4 / real / int2 / uint2
  Bilder, für deren Grauwerte die Standardabweichung berechnet werden sollen.
- ▷ **ImageDeviation** (output_object) ..... image(-array) ⇝ *object :* byte / int4 / real / int2 / uint2
  Bilder, die die Standardabweichung enthalten.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert: 11
  Wertevorschläge: Width ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}
  Restriktion: 3 <= Width && odd(Width) && Width < width(Image) * 2
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert: 11
  Wertevorschläge: Height ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}
  Restriktion: 3 <= Height && odd(Height) && Height < height(Image) * 2

Beispiel

```
read_image(Image,'fabrik')
dev_display(Image)
deviation_image(Image,Deviation,9,9)
dev_display(Image)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deviation_image den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`disp_image`

Alternativen

`entropy_image`, `entropy_gray`

Siehe auch

`convol_image`, `texture_laws`, `intensity`

Modul

Foundation

`entropy_image` (Image : ImageEntropy : Width, Height :)

Entropie der Grauwerte in einem Rechteckfenster.

`entropy_image` transformiert die Grauwerte der Eingabebilder aus `Image` mit Hilfe einer Filtermaske (`Height`, `Width`), in der die Entropie der Grauwerte berechnet wird. Das Ergebnis wird in `ImageEntropy` abgelegt. Die Entropie wird dabei mit 32 multipliziert. Die Steuerparameter `Height` und `Width` werden, falls sie jeweils einen geraden Wert haben, zu dem nächstgrößeren ungeraden Wert gewandelt. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Bilder, in denen die Entropie berechnet werden soll.
- ▷ **`ImageEntropy`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Ergebnisbild.
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert: 9
  Wertevorschläge: `Width` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}
  Restriktion: 3 <= Width && odd(Width)
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert: 9
  Wertevorschläge: `Height` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}
  Restriktion: 3 <= Height && odd(Height)

Beispiel

```
read_image(Image,'fabrik')
dev_display(Image)
entropy_image(Image,Entropy,9,9)
dev_display(Entropy)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `entropy_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`disp_image`

Alternativen

`entropy_gray`

Siehe auch

`energy_gabor`, `entropy_gray`

Modul

Foundation

texture_laws (Image : ImageTexture : FilterTypes, Shift, FilterSize :)

Texturfilter nach Laws.

`texture_laws` führt eine Texturtransformationen (nach Laws) durch. Dazu wird das Bild mit einem speziellen Filter gefaltet. Bei den Filtern handelt es sich um:

9 verschiedene 3×3-Matrizen, die sich aus folgenden drei Vektoren berechnen lassen:

$$\begin{array}{lll} \mathrm{l} & = & [\ 1 \quad 2 \quad 1] \\ \mathrm{e} & = & [-1 \quad 0 \quad 1] \\ \mathrm{s} & = & [-1 \quad 2 \quad -1] \end{array}$$

25 verschiedene 5×5-Matrizen, die sich aus folgenden fünf Vektoren berechnen lassen:

$$\begin{array}{lll} \mathrm{l} & = & [\ 1 \quad 4 \quad 6 \quad 4 \quad 1] \\ \mathrm{e} & = & [-1 \quad -2 \quad 0 \quad 2 \quad 1] \\ \mathrm{s} & = & [-1 \quad 0 \quad 2 \quad 0 \quad -1] \\ \mathrm{w} & = & [-1 \quad 2 \quad 0 \quad -2 \quad 1] \\ \mathrm{r} & = & [\ 1 \quad -4 \quad 6 \quad -4 \quad 1] \end{array}$$

49 verschiedene 7×7-Matrizen, die sich aus folgenden sieben Vektoren berechnen lassen:

$$\begin{array}{lll} \mathrm{l} & = & [\ 1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1] \\ \mathrm{e} & = & [-1 \quad -4 \quad -5 \quad 0 \quad 5 \quad 4 \quad 1] \\ \mathrm{s} & = & [-1 \quad -2 \quad 1 \quad 4 \quad 1 \quad -2 \quad -1] \\ \mathrm{w} & = & [-1 \quad 0 \quad 3 \quad 0 \quad -3 \quad 0 \quad 1] \\ \mathrm{r} & = & [\ 1 \quad -2 \quad -1 \quad 4 \quad -1 \quad -2 \quad -1] \\ \mathrm{u} & = & [\ 1 \quad -4 \quad 5 \quad 0 \quad -5 \quad 4 \quad -1] \\ \mathrm{o} & = & [-1 \quad 6 \quad -15 \quad 20 \quad -15 \quad 6 \quad -1] \end{array}$$

Die Namen der Filter sind Abkürzungen für „Level“, „Edge“, „Spot“, „Wave“, „Ripple“, „Undulation“ und „Oscillation“.

Bei den meisten Filtern ist eine Reduktion der Grauwerte durch einen `Shift` durchzuführen. Auf diese Weise wird im Ausgabebild der Unterschied zwischen zwei Texturen des Eingabebildes vergleichbarer, sofern es sich um den geeigneten Filter handelt.

Der Name der Filter setzt sich aus den Buchstaben der beiden Vektoren zusammen (z.B. ’es’). Dabei gibt der erste Buchstabe die Faltung in Spaltenrichtung und der zweite die Faltung in Zeilenrichtung an.

Achtung

`texture_laws` kann auf OpenCL-Geräten ausgeführt werden.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Bilder, die texturtransformiert werden sollen.
- ▷ **ImageTexture** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Texturbilder.
- ▷ **FilterTypes** (input_control) ..... string ⇝ *string*
  Gewünschter Filter.
  Defaultwert: 'el'
  Wertevorschläge: `FilterTypes` ∈ {'ll', 'le', 'ls', 'lw', 'lr', 'lu', 'lo', 'el', 'ee', 'es', 'ew', 'er', 'eu', 'eo', 'sl', 'se', 'ss', 'sw', 'sr', 'su', 'so', 'wl', 'we', 'ws', 'ww', 'wr', 'wu', 'wo', 'rl', 're', 'rs', 'rw', 'rr', 'ru', 'ro', 'ul', 'ue', 'us', 'uw', 'ur', 'uu', 'uo', 'ol', 'oe', 'os', 'ow', 'or', 'ou', 'oo'}
- ▷ **Shift** (input_control) ..... integer ⇝ *integer*
  Shift der Grauwerte zur Dynamikverringerung.
  Defaultwert: 2
  Wertevorschläge: `Shift` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- ▷ **FilterSize** (input_control) ..... integer ⇝ *integer*
  Größe des Filterkerns.
  Defaultwert: 5
  Werteliste: `FilterSize` ∈ {3, 5, 7}

Beispiel

```
* Simple two-dimensional pixel classification
dev_get_window (WindowHandle)
read_image(Image,'combine')
texture_laws(Image,Texture1,'es',3,7)
texture_laws(Image,Texture2,'le',7,7)
MaskSize := 51
mean_image(Texture1,H1,MaskSize,MaskSize)
mean_image(Texture2,H2,MaskSize,MaskSize)
dev_clear_window ()
dev_display (Image)
dev_set_color ('green')
write_string (WindowHandle, 'Mark region within one texture area')
draw_region(Region,WindowHandle)
reduce_domain(H1,Region,Foreground1)
reduce_domain(H2,Region,Foreground2)
histo_2dim(Region,Foreground1,Foreground2,Histo)
get_image_size (Image, Width, Height)
threshold(Histo,Characteristic_area,1,Width*Height)
ShowIntermediateResult := 0
if (ShowIntermediateResult)
  histo_2dim(H1,H1,H2,HistoFull)
  dev_clear_window ()
  dev_set_lut ('sqrt')
  dev_display (HistoFull)
  dev_set_draw ('margin')
  dev_display (Characteristic_area)
  stop ()
  dev_set_lut ('default')
  dev_set_draw ('fill')
endif
class_2dim_sup(H1,H2,Characteristic_area,Seg)
dev_display (Image)
dev_set_color ('red')
dev_display (Seg)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `texture_laws` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`mean_image`, `binomial_filter`, `gauss_filter`, `median_image`, `histo_2dim`, `learn_ndim_norm`, `threshold`

Alternativen

`convol_image`

Siehe auch

`class_2dim_sup`, `class_ndim_norm`

Literatur

Laws, Kenneth Ivan. „Textured Image Segmentation“; Ph.D. Thesis, Department of Electrical Engineering, Image Processing Institute, University of Southern California, 1980

Modul

Foundation

10.17 Verbesserung

`coherence_enhancing_diff` `( Image : ImageCED : Sigma, Rho, Theta, Iterations : )`

Kohärenz verstärkende Diffusion eines Bildes.

Der Operator `coherence_enhancing_diff` führt einen anisotropen Diffusionsvorgang auf dem Eingabebild `Image` durch mit dem Ziel, die Kohärenz der in `Image` enthaltenen Bildstrukturen zu erhöhen, das heißt insbesondere, unterbrochene Bildkanten diffusiv miteinander zu verbinden ohne sie dabei senkrecht zur Kantenrichtung zu verschmieren. Dazu nutzt `coherence_enhancing_diff` eine Diskretisierung der anisotropen Diffusionsgleichung

$$u_t = div(G(u)\nabla u)$$

nach Weickert. Mit einer von den Grauwerten in `Image` abhängigen 2×2 Koeffizientenmatrix G ist diese eine Weiterentwicklung der Krümmungsfluss- oder intrinsischen Wärmeleitungsgleichung

$$u_t = div(\frac{\nabla u}{|\nabla u|})|\nabla u| = curv(u)|\nabla u|$$

auf der durch das Eingabebild `Image` zu einem Zeitpunkt $t_0 = 0$ definierten Grauwertfunktion u. Der Glättungsoperator `mean_curvature_flow` verwendet die Krümmungsflussgleichung direkt. Die diskrete Diffusionsgleichung wird in `Iterations` Zeitschritten der Länge `Theta` gelöst, so dass das Ausgabebild `ImageCED` die Grauwertfunktion zur Zeit `Iterations` $\cdot$ `Theta` enthält.

Um die Kantenrichtungen insbesondere bei verrauschten Eingabedaten stabiler zu bestimmen, kann der Berechnung der Grauwertgradienten ein zusätzlicher isotroper Glättungsschritt vorgeschaltet werden. Der Parameter

Sigma bestimmt die Stärke dieser Glättung als Standardabweichung des zugehörigen Gauß-Faltungskernes, wie er auch im Operator isotropic_diffusion zur isotropen Glättung eines Bildes verwendet wird.

Während die Matrix G im Fall des Krümmungsflusses mean_curvature_flow gegeben wird durch

$$G_{MCF}(u) = I - \frac{1}{|\nabla u|^2} \nabla u (\nabla u)^T,$$

wobei I die Einheitsmatrix bezeichnet, wird die Matrix G_{MCF} in coherence_enhancing_diff komponentenweise in Ortsrichtung durch einen Gauß-Filter mit Standardabweichung Rho geglättet und anschließend aus den Eigenwerten λ_1, λ_2 und Eigenvektoren w_1, w_2 der so entstandenen Matrix die endgültige Koeffizientenmatrix

$$G_{CED} = g_1\left((\lambda_1 - \lambda_2)^2\right) w_1 (w_1)^T + g_2\left((\lambda_1 - \lambda_2)^2\right) w_2 (w_2)^T$$

konstruiert, wobei die Funktionen

$$\begin{aligned} g_1(p) &= 0.001 \\ g_2(p) &= 0.001 + 0.999 \exp\left(\frac{-1}{p}\right) \end{aligned}$$

empirisch bestimmt und aus der Veröffentlichung von Weickert entnommen sind.

Die Diffusionsrichtung wird bei mean_curvature_flow also alleine durch die lokale Richtung des Grauwertgradienten bestimmt, wohingegen G_{CED} die makroskopische Struktur der Bildobjekte auf der Skala Rho wiedergibt und die Stärke der Diffusion in coherence_enhancing_diff davon abhängt, wie gut diese Struktur definiert ist.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **ImageCED** (output_object) ..... image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **Sigma** (input_control) ..... real ⇝ *real*
  Glättungsparameter für Ableitungsoperator.
  Defaultwert: 0.5
  Wertevorschläge: Sigma ∈ {0.0, 0.1, 0.5, 1.0}
  Restriktion: Sigma >= 0
- ▷ **Rho** (input_control) ..... real ⇝ *real*
  Glättungsparameter für Diffusionskoeffizienten.
  Defaultwert: 3.0
  Wertevorschläge: Rho ∈ {0.0, 1.0, 3.0, 5.0, 10.0, 30.0}
  Restriktion: Rho >= 0
- ▷ **Theta** (input_control) ..... real ⇝ *real*
  Zeitschritt.
  Defaultwert: 0.5
  Wertevorschläge: Theta ∈ {0.1, 0.2, 0.3, 0.4, 0.5}
  Restriktion: 0 < Theta <= 0.5
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl Iterationen.
  Defaultwert: 10
  Wertevorschläge: Iterations ∈ {1, 5, 10, 20, 50, 100, 500}
  Restriktion: Iterations >= 1

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Literatur

J. Weickert, V. Hlavac, R. Sara; „Multiscale texture enhancement“; Computer analysis of images and patterns, Lecture Notes in Computer Science, Vol. 970, pp. 230-237; Springer, Berlin; 1995.

J. Weickert, B. ter Haar Romeny, L. Florack, J. Koenderink, M. Viergever; „A review of nonlinear diffusion filtering“; Scale-Space Theory in Computer Vision, Lecture Notes in Comp. Science, Vol. 1252, pp. 3-28; Springer, Berlin; 1997.

Modul

Foundation

emphasize (Image : ImageEmphasize : MaskWidth, MaskHeight, Factor :)

Kontrast des Bildes verbessern.

`emphasize` hebt hochfrequente Anteile im Bild hervor (Kanten und Ecken). Das Ergebnisbild erscheint daher schärfer.

Die Prozedur führt zunächst eine Filterung mit dem Tiefpass (`mean_image`) durch. Aus den so gewonnenen Grauwerten ($mean$) und den Originalgrauwerten ($orig$) werden die Ergebnisgrauwerte (res) wie folgt berechnet:

$$res := round((orig - mean) * \texttt{Factor}) + orig$$

`Factor` dient somit als Maß für die Kontraststeigerung. Die Trennfrequenz wird über die Größe der Filtermatrix festgelegt. Je größer die Matrix ist, desto niedriger wird die Trennfrequenz.

Als Randbehandlungen werden die Grauwerte an den Bildrändern gespiegelt. Über- bzw. Unterlauf von Grauwerten wird beschnitten.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
Zu verbesserndes Bild.

▷ **ImageEmphasize** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
Kontrastverstärktes Bild.

▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
Breite der Tiefpassmaske.
Defaultwert: 7
Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9, 11, 15, 21, 25, 31, 39}
Typischer Wertebereich: 3 ≤ `MaskWidth` ≤ 201
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2

▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
Höhe der Tiefpassmaske.
Defaultwert: 7
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11, 15, 21, 25, 31, 39}
Typischer Wertebereich: 3 ≤ `MaskHeight` ≤ 201
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2

▷ **Factor** (input_control) ..... real ⇝ *real*
Stärke der Kontrastanhebung.
Defaultwert: 1.0
Wertevorschläge: `Factor` ∈ {0.3, 0.5, 0.7, 1.0, 1.4, 1.8, 2.0}
Typischer Wertebereich: 0.0 ≤ `Factor` ≤ 20.0 (sqrt)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.2
Restriktion: 0 < Factor && Factor < 20

Beispiel

```
read_image(Image,'mreut')
dev_display (Image)
draw_region(Region,WindowHandle)
reduce_domain(Image,Region,Mask)
emphasize(Mask,Sharp,7,7,2.0)
dev_display (Sharp)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `emphasize` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Nachfolger

`disp_image`

Alternativen

`mean_image`, `sub_image`, `laplace`, `add_image`

Siehe auch

`mean_image`, `highpass_image`

Modul

Foundation

`equu_histo_image` (Image : ImageEquHisto : :)

Histogrammlinearisierung von Bildern

`equ_histo_image` dient der Kontrastverbesserung. Ausgangspunkt dafür ist das Histogramm der Eingabebilder. Es wird folgende einfache Grauwerttransformation $f(g)$ für Bilder vom Typ byte durchgeführt:

$$f(g) = 255 \sum_{x=0\ldots g} h(x)$$

Dabei bezeichnet $h(x)$ die relative Häufigkeit des Auftretens des Grauwertes x. Für Bilder vom Typ uint2 ist der einzige Unterschied, dass statt 255 ein anderer Maximalwert verwendet wird. Dieser Wert wird aus der mit dem Eingabebild abgespeicherten Anzahl signifikanter Bits berechnet, falls dieser Wert gesetzt ist. Ansonsten wird der Wert des Systemparameters *'int2_bits'* verwendet (siehe `set_system`), falls er gesetzt (d.h. ungleich -1) ist. Falls keiner der beiden Werte gesetzt ist, wird die Anzahl signifikanter Bits auf 16 gesetzt.

Diese Transformation linearisiert das kumulative Histogramm. Maxima im ursprünglichen Histogramm werden „gespreizt“ und damit der Kontrast in Bildregionen mit diesen häufig auftretenden Grauwerten erhöht. Vermeintlich homogene Bereiche erhalten so besser sichtbare Strukturen. Andererseits wird natürlich auch das Rauschen im Bild entsprechend verstärkt. Minima im ursprünglichen Histogramm werden dual dazu „gestaucht“. Das transformierte Histogramm enthält dann zwar Lücken, die verbleibenden verwendeten Grauwerte treten aber etwa gleich häufig auf („histogram equalization“).

Achtung

`equ_histo_image` dient in erster Linie der optischen Aufbereitung von Bildern für einen menschlichen Betrachter. So kann beispielsweise die durchgeführte (lokale) Kontrastspreizung in der Folge zur Detektion von Scheinkanten führen.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Zu verbesserndes Bild.

▷ **ImageEquHisto** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Bild mit linearisierten Grauwerten.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Nachfolger

`disp_image`

Alternativen

`scale_image`, `scale_image_max`, `illuminate`

Siehe auch

`scale_image`

Literatur

R.C. Gonzales, P. Wintz: „Digital Image Processing“; Second edition; Addison Wesley; 1987.

Modul

Foundation

illuminate (Image : ImageIlluminate : MaskWidth, MaskHeight, Factor :)

Ausleuchten von Bildern.

`illuminate` dient der Kontrastverbesserung. Sehr dunkle Bildteile werden besser „ausgeleuchtet“, sehr helle „abgedunkelt“. Sei $orig$ der Original Grauwert und $mean$ der mittels `mean_image` und Filtergröße `MaskHeight` x `MaskWidth` ermittelte zugehörige Grauwert des tiefpassgefilterten Bildes. Bei byte-Bildern ist val gleich 127, bei int2-Bildern und uint2-Bildern ist val gleich dem Mittelwert. Dann ist der Ergebnisgrauwert new:

$$new := round((val - mean) * \texttt{Factor} + orig)$$

Der Tiefpass sollte dabei recht groß dimensioniert werden (30 x 30 bis 200 x 200). Vernünftige Parameterkombinationen sind etwa:

MaskHeight	MaskWidth	Factor
40	40	0.55
100	100	0.7
150	150	0.8

d.h. je größer die Tiefpassmaske gewählt wird, desto größer sollte auch `Factor` eingestellt werden.

Hingewiesen sei noch auf folgenden „Scheinwerfereffekt“: Befindet sich in einem Bild beispielsweise ein dunkles Objekt vor einer hellen Wand, so wird das Objekt, in unmittelbarer Nähe der Objektkontur aber auch die ohnehin schon helle Wand durch die Anwendung von `illuminate` aufgehellt. Dies entspricht in etwa dem Effekt, der sich durch Anstrahlen des Objekts mit einem starken Scheinwerfer ergeben. Analoges gilt auch für helle Objekte vor einem dunklen Hintergrund. Der fiktive „Scheinwerfer“ verdunkelt nun allerdings Objekte.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  Zu verbesserndes Bild.
- ▷ **`ImageIlluminate`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2
  „Ausgeleuchtetes“ Bild.
- ▷ **`MaskWidth`** (input_control) ..... extent.x ⇝ *integer*
  Breite der Tiefpassmaske.
  Defaultwert: 101
  Wertevorschläge: `MaskWidth` ∈ {31, 41, 51, 71, 101, 121, 151, 201}
  Typischer Wertebereich: 3 ≤ `MaskWidth` ≤ 299
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 10
  Restriktion: MaskWidth < width(Image) * 2
- ▷ **`MaskHeight`** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Tiefpassmaske.
  Defaultwert: 101
  Wertevorschläge: `MaskHeight` ∈ {31, 41, 51, 71, 101, 121, 151, 201}
  Typischer Wertebereich: 3 ≤ `MaskHeight` ≤ 299
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 10
  Restriktion: MaskHeight < height(Image) * 2
- ▷ **`Factor`** (input_control) ..... real ⇝ *real*
  Damit wird der „Korrekturgrauwert“, der zu den Original Grauwerten addiert wird, skaliert.
  Defaultwert: 0.7
  Wertevorschläge: `Factor` ∈ {0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0}
  Typischer Wertebereich: 0.0 ≤ `Factor` ≤ 5.0
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.2
  Restriktion: 0 < Factor && Factor < 5

Beispiel

```
read_image(Image,'fabrik')
dev_display (Image)
illuminate(Image,Better,40,40,0.55)
dev_display (Better)
```

Ergebnis

`illuminate` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`disp_image`

Alternativen

`scale_image_max`, `equ_histo_image`, `mean_image`, `sub_image`

Siehe auch

`emphasize`, `gray_histo`

Modul

Foundation

mean_curvature_flow (Image : ImageMCF : Sigma, Theta, Iterations :)

Anwendung der Krümmungsflussgleichung auf ein Bild.

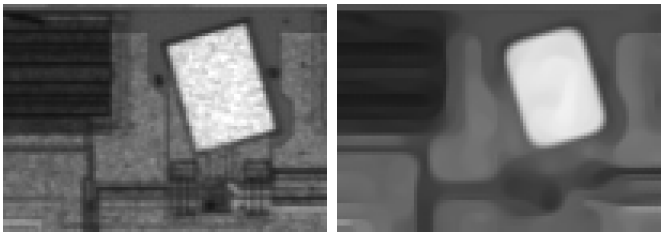
Der Operator `mean_curvature_flow` wendet die Krümmungsfluss- oder intrinsische Wärmeleitungsgleichung

$$u_t = div(\frac{\nabla u}{|\nabla u|})|\nabla u| = curv(u)|\nabla u|$$

auf die durch das Eingabebild `Image` zu einem Zeitpunkt $t_0 = 0$ definierte Grauwertfunktion u an. Die diskretisierte Gleichung wird in `Iterations` Zeitschritten der Länge `Theta` gelöst, so dass das Ausgabebild `ImageMCF` die Grauwertfunktion zur Zeit `Iterations` · `Theta` enthält.

Der Krümmungsfluss bewirkt eine Glättung von `Image` in Richtung der Bildkanten, also entlang der Niveaulinien von u, während senkrecht zur Kantenrichtung keine Glättung durchgeführt wird und die Ränder von Bildobjekten somit nicht verwischen. Um die Kantenrichtungen insbesondere bei verrauschten Eingabedaten stabiler zu bestimmen, kann der Berechnung der Grauwertgradienten ein zusätzlicher isotroper Glättungsschritt vorgeschaltet werden. Der Parameter `Sigma` bestimmt die Stärke dieser Glättung als Standardabweichung des zugehörigen Gauß-Faltungskernes, wie er auch im Operator `isotropic_diffusion` zur isotropen Glättung eines Bildes verwendet wird.

Die folgenden Abbildungen zeigen den Effekt der Parameter `Sigma`, `Theta`, und `Iterations`. Zuerst wird das Eingabebild zusammen mit dem Ergebnis dargestellt, das man mit den Defaultwerten erhält.

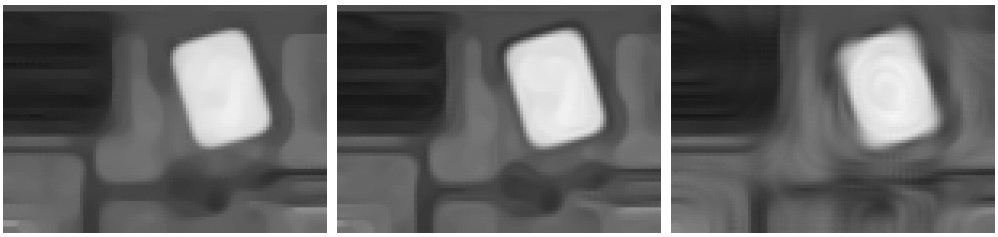


(1) (2)

(1) Eingabebild. (2) Ergebnis bei Verwendung der Defaultwerte.

In den folgenden Abbildungen werden die Ergebnisse dargestellt, die man erhält, wenn jeweils ein Parameterwert verändert wird, während die anderen beiden auf ihren jeweiligen Defaultwert gesetzt sind.

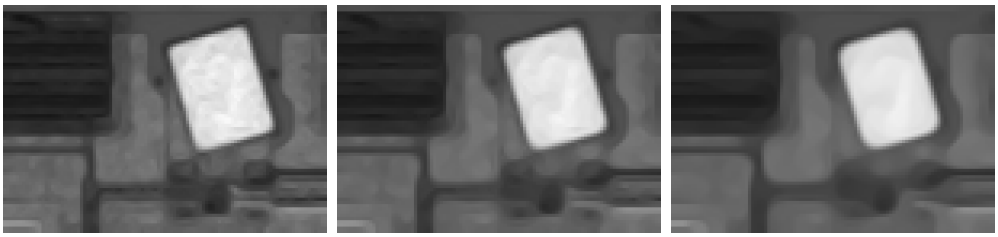
`Sigma` steuert die Stärke der Glättung, die vor der Berechnung der Grauwertgradienten durchgeführt wird. Es sollten keine zu großen Werte für `Sigma` verwendet werden, da dies zu unerwünschten Effekten führen kann.



(1) (2) (3)

(1) Sigma = 0.0. (2) Sigma = 1.0. (3) Sigma = 10.0.

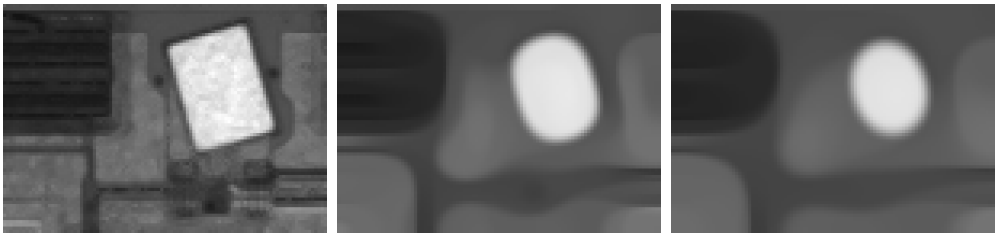
`Theta` steuert die Schrittweite innerhalb des iterativen Glättungsprozesses. Größere Werte führen zu einer stärkeren Glättung.



(1) (2) (3)

(1) Theta = 0.1. (2) Theta = 0.2. (3) Theta = 0.4.

`Iterations` legt die Anzahl von Iterationen fest, die durchgeführt werden. Mit steigender Anzahl von Iterationen nimmt auch die Laufzeit zu. Darüberhinaus kann eine hohe Anzahl von Iterationen zu einem Verlust der Bildstruktur im geglätteten Bild führen.



(1) (2) (3)

(1) Iterations = 1. (2) Iterations = 50. (3) Iterations = 100.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- **`ImageMCF`** (output_object) ..... image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- **`Sigma`** (input_control) ..... real ⇝ *real*
  Glättungsparameter für Ableitungsoperator.
  Defaultwert: 0.5
  Wertevorschläge: `Sigma` ∈ {0.0, 0.1, 0.5, 1.0}
  Restriktion: Sigma >= 0
- **`Theta`** (input_control) ..... real ⇝ *real*
  Zeitschritt.
  Defaultwert: 0.5
  Wertevorschläge: `Theta` ∈ {0.1, 0.2, 0.3, 0.4, 0.5}
  Restriktion: 0 < Theta <= 0.5

▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
Anzahl Iterationen.
Defaultwert: 10
Wertevorschläge: `Iterations` ∈ {1, 5, 10, 20, 50, 100, 500}
Restriktion: Iterations >= 1

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Literatur

M. G. Crandall, P. Lions; „Convergent Difference Schemes for Nonlinear Parabolic Equations and Mean Curvature Motion“; Numer. Math. 75 pp. 17-41; 1996.

G. Aubert, P. Kornprobst; „Mathematical Problems in Image Processing“; Applied Mathematical Sciences 147; Springer, New York; 2002.

Modul

Foundation

scale_image_max (Image : ImageScaleMax : :)

Maximale Grauwertspreizung auf den Wertebereich 0 bis 255.

`scale_image_max` berechnet Minimum und Maximum und skaliert das Bild hiermit auf den maximalen Wertebereich eines Byte-Bildes. Hierdurch wird die Dynamik (Wertebereich) voll ausgeschöpft. Die Anzahl der unterschiedlichen Graustufen ändert sich nicht. Es entsteht aber i.a. ein visuell besserer Eindruck. Bei Bildern vom Typ *'real'*, *'int2'*, *'uint2'*, *'int4'* und *'int8'* werden die Grauwerte auf den Bereich *0* bis *255* skaliert und als *'byte'*-Bild ausgegeben.

Achtung

Die Ausgabe ist immer ein Ergebnisbild vom Typ *'byte'*.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / int8 / real
Zu skalierendes Bild.

▷ **ImageScaleMax** (output_object) . . . . . image(-array) ⇝ *object :* byte
Kontrastverstärktes Bild.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Nachfolger

`disp_image`

Alternativen

`equ_histo_image`, `scale_image`, `illuminate`, `convert_image_type`

Siehe auch

`min_max_gray`, `gray_histo`

Modul

Foundation

shock_filter (Image : SharpenedImage : Theta, Iterations, Mode, Sigma :)

Anwendung eines Schockfilters auf ein Bild.

Der Operator shock_filter wendet einen Schockfilter zur Schärfung von Bildkanten auf das Eingabebild Image an. Das Prinzip eines Schockfilters beruht darauf, die Grauwerte eines Bildes durch Dilation und Erosion von beiden Seiten auf eine Kante zu zu transportieren und erfüllt somit die Differentialgleichung

$$u_t = s\,|\nabla u|$$

auf der zum Zeitpunkt $t_0 = 0$ durch die Grauwerte in Image definierten Funktion u. Die diskretisierte Gleichung wird in Iterations Zeitschritten der Länge Theta gelöst, so dass das Ausgabebild SharpenedImage die Grauwertfunktion zur Zeit Iterations · Theta enthält.

Die Entscheidung für Dilation oder Erosion wird über die Vorzeichenfunktion $s \in \{-1, 0, +1\}$ mit Hilfe eines üblichen Kantendetektors getroffen. Es stehen der Detektor von Canny

$$s = -sgn\left(D^2u(\frac{\nabla u}{|\nabla u|}, \frac{\nabla u}{|\nabla u|})\right)$$

mit Mode $= 'canny'$ und der Detektor von Marr/Hildreth (der Laplace-Operator)

$$s = -sgn(\Delta u)$$

mit Mode $= 'laplace'$ zur Verfügung.

Damit die Erkennung der Kanten, insbesondere bei verrauschten Bildern, robuster erfolgt, kann diese auf einer geglätteten Bildmatrix ausgeführt werden. Dazu kann im Parameter Sigma die Standardabweichung eines Gausskernes angegeben werden, mit dem die Bildmatrix gefaltet werden soll.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **SharpenedImage** (output_object) . . . . . image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild.
- ▷ **Theta** (input_control) . . . . . real ⇝ *real*
  Zeitschritt.
  Defaultwert: 0.5
  Wertevorschläge: Theta ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7}
  Restriktion: 0 < Theta <= 0.7
- ▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
  Anzahl Iterationen.
  Defaultwert: 10
  Wertevorschläge: Iterations ∈ {1, 3, 10, 100}
  Restriktion: Iterations >= 1
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Art des Kantendetektors.
  Defaultwert: 'canny'
  Werteliste: Mode ∈ {'laplace', 'canny'}
- ▷ **Sigma** (input_control) . . . . . real ⇝ *real*
  Glättung des Kantendetektors.
  Defaultwert: 1.0
  Wertevorschläge: Sigma ∈ {0.0, 0.5, 1.0, 2.0, 5.0}
  Restriktion: Theta >= 0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Literatur

F. Guichard, J. Morel; „A Note on Two Classical Shock Filters and Their Asymptotics“; Michael Kerckhove (Ed.): Scale-Space and Morphology in Computer Vision, LNCS 2106, pp. 75-84; Springer, New York; 2001.

G. Aubert, P. Kornprobst; „Mathematical Problems in Image Processing“; Applied Mathematical Sciences 147; Springer, New York; 2002.

Modul

Foundation

10.18 Wienerfilter

gen_psf_defocus (: Psf : PSFwidth, PSFheight, Blurring :)

Erzeugung der Impulsantwort einer gleichmäßigen Defokussierung.

gen_psf_defocus erzeugt die Impulsantwort (im Ortsraum) einer gleichmäßigen Defokussierung als Bildobjekt vom Bildtyp 'real'. Seine Breite bzw. Höhe lässt sich durch PSFwidth und PSFheight angeben. Die Bildbeeinträchtigung betrifft das gesamte Bild gleichermaßen. Ihr Ausmaß wird durch Blurring angegeben. Dieser Parameter beschreibt den sogenannten „Defokussierungsradius“. Durch die falsche Fokussierung wird jeder Bildpunkt (im Prinzip) auf eine kleine Kreisfläche abgebildet. Dadurch wird das Ergebnisbild unscharf. Der Radius dieses Kreises wird also durch Blurring (in Bildpunkten) festgelegt. Falls er kleiner als null eingegeben wird, so wird sein Betrag verwendet. Das erzeugte float-Bild enthält die gewünschte Impulsantwort im Ortsraum, wobei ihre Darstellung voraussetzt, dass sich der Nullpunkt in der Bildecke („links oben“) befindet. Das bedeutet folgende Einteilung der insgesamt NxM-großen Bildmatrix in vier rechteckige Regionen:

- Erstes Rechteck („links oben“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem vierten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 0..N/2$ und $y = 0.. - M/2$
- Zweites Rechteck („rechts oben“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem dritten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = -1.. - M/2$
- Drittes Rechteck („links unten“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = M/2..M - 1$)
  -entspricht dem ersten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 1..N/2$ und $y = M/2..0$
- Viertes Rechteck („rechts unten“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = M/2..M - 1$)
  -entspricht dem zweiten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = M/2..1$

Aufgrund dieser Darstellungsform ist es möglich, die erzeugte Impulsantwort direkt als Eingabe der Routine wiener_filter zu verwenden, um ein defokussiertes Bild mittels Wiener-Filterung zu restaurieren.

Parameter

▷ **Psf** (output_object) ..... image ⇝ *object :* real
Impulsantwort einer gleichmäßigen Defokussierung.

▷ **PSFwidth** (input_control) ..... integer ⇝ *integer*
Breite der Impulsantwort-Bildmatrix.
Defaultwert: 256
Wertevorschläge: PSFwidth ∈ {128, 256, 512, 1024}
Typischer Wertebereich: $1 \leq$ PSFwidth

▷ **PSFheight** (input_control) . . . . . integer ⇝ *integer*
Höhe der Impulsantwort-Bildmatrix.
Defaultwert: 256
Wertevorschläge: PSFheight ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ PSFheight

▷ **Blurring** (input_control) . . . . . real ⇝ *real*
Stärke der Beeinträchtigung (Defokussierungsradius).
Defaultwert: 5.0
Wertevorschläge: Blurring ∈ {1.0, 5.0, 10.0, 15.0, 18.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_psf_defocus den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

simulate_motion, gen_psf_motion

Nachfolger

simulate_defocus, wiener_filter, wiener_filter_ni

Siehe auch

simulate_defocus, gen_psf_motion, simulate_motion, wiener_filter, wiener_filter_ni

Literatur

Reginald L. Lagendijk, Jan Biemond: Iterative Identification and Restoration of Images, Kluwer Academic Publishers Boston/Dordrecht/London, 1991

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Modul

Foundation

gen_psf_motion (: Psf : PSFwidth, PSFheight, Blurring, Angle, Type :)

Erzeugung der Impulsantwort einer geradlinigen Bewegungsverschlechterung.

gen_psf_motion erzeugt die Impulsantwort (im Ortsraum) einer geradlinigen Bewegungsverschlechterung als Bildobjekt vom Bildtyp 'real'. Seine Breite bzw. Höhe lässt sich durch PSFwidth und PSFheight angeben. Die beschriebene Bewegung erfolgt entlang einer Linie. Ihre Richtung ist durch den Winkel zur x-Achse (gegen den Uhrzeigersinn) festgelegt. Dieser wird mit Angle im Gradmaß eingegeben. Zur Beschreibung von unterschiedlichen Bewegungsformen, stehen fünf Prototypen für die Impulsantwort zur Verfügung. Welcher Prototyp Verwendung findet, regelt Type. Folgende Werte sind möglich:

1. Umgekehrte Rampe (beschreibt in etwa eine Beschleunigung)
2. Umgekehrtes Trapezoid (beschreibt in etwa starke Beschleunigung)
3. Rechtecksfunktion (beschreibt (genau) konstante Geschwindigkeit)
4. Normales Trapezoid (beschreibt in etwa starkes Abbremsen)
5. Normale Rampe (beschreibt in etwa Abbremsen)

(Für alle anderen Werte wird automatisch die Rechtecksfunktion verwendet.) Die Bewegungsverschlechterung betrifft auf alle Fälle das gesamte eingegebene Bild. Ihr Ausmaß ist durch Blurring festgelegt. Dieser Parameter beschreibt, wie viele Bildpunkte, die auf der Geraden der Bewegungsrichtung hintereinander liegen, durch

die Bewegungsverschlechterung betroffen sind. Die Anzahl der betroffenen Punkte wird durch die Bewegungsgeschwindigkeit, -beschleunigung und -zeitdauer bestimmt. Falls `Blurring` kleiner als null angegeben ist, so wird die Impulsantwort einer entsprechend starken Bewegung in entgegengesetzter Richtung erzeugt. Falls `Angle` kleiner als null eingegeben wird, so wird der Winkel im Uhrzeigersinn interpretiert. Falls `Angle` größer als 360 bzw. kleiner als -360 angegeben ist, so wird er modulo(360) in das Intervall $[0..360]$ bzw. $[-360..0]$ zurückgerechnet. Das erzeugte float-Bild enthält die gewünschte Impulsantwort im Ortsraum, wobei ihre Darstellung voraussetzt, dass sich der Nullpunkt in der Bildecke („links oben“) befindet. Das bedeutet folgende Einteilung der insgesamt NxM-großen Bildmatrix in vier rechteckige Regionen:

- Erstes Rechteck („links oben“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem vierten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 0..N/2$ und $y = 0.. - M/2$
- Zweites Rechteck („rechts oben“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem dritten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = -1.. - M/2$
- Drittes Rechteck („links unten“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = M/2..M - 1$)
  -entspricht dem ersten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 1..N/2$ und $y = M/2..0$
- Viertes Rechteck („rechts unten“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = M/2..M - 1$)
  -entspricht dem zweiten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = M/2..1$

Aufgrund dieser Darstellungsform ist es möglich, die erzeugte Impulsantwort direkt als Eingabe der Routine `wiener_filter` zu verwenden, um ein bewegungsverschlechtertes Bild mittels Wiener-Filterung zu restaurieren.

Parameter

▷ **`Psf`** (output_object) ..... image ⇝ *object :* real
Impulsantwort einer Bewegungsverschlechterung.

▷ **`PSFwidth`** (input_control) ..... integer ⇝ *integer*
Breite der Impulsantwort-Bildmatrix.
Defaultwert: 256
Wertevorschläge: `PSFwidth` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: $1 \leq$ PSFwidth

▷ **`PSFheight`** (input_control) ..... integer ⇝ *integer*
Höhe der Impulsantwort-Bildmatrix.
Defaultwert: 256
Wertevorschläge: `PSFheight` ∈ {128, 256, 512, 1024}
Typischer Wertebereich: $1 \leq$ PSFheight

▷ **`Blurring`** (input_control) ..... real ⇝ *real*
Stärke der Bewegungsverschlechterung.
Defaultwert: 20.0
Wertevorschläge: `Blurring` ∈ {5.0, 10.0, 20.0, 30.0, 40.0}

▷ **`Angle`** (input_control) ..... integer ⇝ *integer*
Winkel zwischen der Bewegungsrichtung und der x-Achse (gegen den Uhrzeigersinn).
Defaultwert: 0
Wertevorschläge: `Angle` ∈ {0, 45, 90, 180, 270}

▷ **`Type`** (input_control) ..... integer ⇝ *integer*
Impulsantwort-Prototyp bzw. Form der Bewegung
Defaultwert: 3
Werteliste: `Type` ∈ {1, 2, 3, 4, 5}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_psf_motion` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

simulate_defocus, gen_psf_defocus

Nachfolger

simulate_motion, wiener_filter, wiener_filter_ni

Siehe auch

simulate_motion, simulate_defocus, gen_psf_defocus, wiener_filter, wiener_filter_ni

Literatur

Anil K. Jain:Fundamentals of Digital Image Processing, Prentice-Hall International Inc., Englewood Cliffs, New Jersey, 1989

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Kha-Chye Tan, Hock Lim, B. T. G. Tan:„Restoration of Real-World Motion-Blurred Images“;S. 291-299 in: CVGIP Graphical Models and Image Processing, Vol. 53, No. 3, May 1991

Modul

Foundation

simulate_defocus (Image : DefocusedImage : Blurring :)

Simulation einer gleichmäßigen Defokussierung.

simulate_defocus simuliert eine Bildbeeinträchtigung. Ihre Ursache liegt in einer falschen Fokussierung des aufnehmenden Linsensystems. Die daraus resultierende Bildverschlechterung betrifft das gesamte Eingabebild gleichermaßen. Ihr Ausmaß wird durch Blurring angegeben. Dieser Parameter beschreibt den sogenannten „Defokussierungsradius“. Durch die falsche Fokussierung wird jeder Bildpunkt (im Prinzip) auf eine kleine Kreisfläche abgebildet. Dadurch wird das Ergebnisbild unscharf. Der Radius dieses Kreises wird also durch Blurring (in Bildpunkten) festgelegt. Falls er kleiner als null eingegeben wird, so wird sein Betrag verwendet. Die Simulation der Bildbeeinträchtigung erfolgt durch eine Faltung vom Eingabebild mit einer Impulsantwort, die eine gleichmäßige Defokussierung beschreibt. simulate_defocus erzeugt die benötigte Impulsantwort und multipliziert ihre (diskrete) Fouriertransformierte mit der des Eingabebildes. Das in den Ortsraum zurück transformierte Produkt stellt das defokussierte Bild dar.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Bild, das beeinträchtigt werden soll.

▷ **DefocusedImage** (output_object) ..... image(-array) ⇝ *object :* real
Defokussiertes Bild.

▷ **Blurring** (input_control) ..... real ⇝ *real*
Stärke der Beeinträchtigung (Defokussierungsradius).
Defaultwert: 5.0
Wertevorschläge: Blurring ∈ {1.0, 5.0, 10.0, 15.0, 18.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert simulate_defocus den Wert TRUE. Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

gen_psf_defocus, simulate_motion, gen_psf_motion

Nachfolger

wiener_filter, wiener_filter_ni

Siehe auch

gen_psf_defocus, simulate_motion, gen_psf_motion

Literatur

Reginald L. Lagendijk, Jan Biemond: Iterative Identification and Restoration of Images, Kluwer Academic Publishers Boston/Dordrecht/London, 1991

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Modul

Foundation

simulate_motion (Image : MovedImage : Blurring, Angle, Type :)

Simulation einer geradlinigen Bewegungsverschlechterung.

simulate_motion simuliert eine Bildbeeinträchtigung. Ihre Ursache liegt in einer relativen Bewegung zwischen dem aufzunehmenden Objekt und dem Aufnahmesystem während der Aufnahme. Die Bewegung erfolgte dabei entlang einer Linie. Ihre Richtung ist durch den Winkel zur x-Achse (gegen den Uhrzeigersinn) festgelegt. Dieser wird mit Angle im Gradmaß eingegeben. Die Simulation erfolgt durch eine Faltung vom Eingabebild mit einer Impulsantwort, die die jeweilige Bewegungsart beschreibt. simulate_motion erzeugt die benötigte Impulsantwort und multipliziert ihre (diskrete) Fouriertransformierte mit der des Eingabebildes. Das in den Ortsraum zurück transformierte Produkt stellt das bewegungsverschlechterte Bild dar. Zur Beschreibung von unterschiedlichen Bewegungsformen, stehen fünf Prototypen für die Impulsantwort zur Verfügung. Welcher Prototyp Verwendung findet, regelt Type. Folgende Werte sind möglich:

1. Umgekehrte Rampe (beschreibt in etwa eine Beschleunigung)
2. Umgekehrtes Trapezoid (beschreibt in etwa starke Beschleunigung)
3. Rechtecksfunktion (beschreibt (genau) konstante Geschwindigkeit)
4. Normales Trapezoid (beschreibt in etwa starkes Abbremsen)
5. Normale Rampe (beschreibt in etwa Abbremsen)

(Für alle anderen Werte wird automatisch die Rechtecksfunktion verwendet.) Die Bewegungsverschlechterung betrifft auf alle Fälle das gesamte eingegebene Bild. Ihr Ausmaß ist durch Blurring festgelegt. Dieser Parameter beschreibt, wie viele Bildpunkte, die auf einer Geraden in Bewegungsrichtung hintereinander liegen, durch die Bewegungsverschlechterung betroffen sind. Die Anzahl der betroffenen Punkte wird durch die Bewegungsgeschwindigkeit, -beschleunigung und -zeitdauer bestimmt. Falls Blurring kleiner als null angegeben ist, so wird eine entsprechend starke Bewegung in entgegengesetzter Richtung simuliert. Falls Angle kleiner als null eingegeben wird, so wird der Winkel im Uhrzeigersinn interpretiert. Falls Angle größer als 360 bzw. kleiner als -360 angegeben ist, so wird er modulo(360) in das Intervall $[0..360]$ bzw. $[-360..0]$ zurückgerechnet.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Bild, das beeinträchtigt werden soll.

▷ **MovedImage** (output_object) ..... image ⇝ *object :* real
Bewegungsverschlechtertes Bild.

▷ **Blurring** (input_control) ..... real ⇝ *real*
Stärke der Bewegungsverschlechterung.
Defaultwert: 20.0
Wertevorschläge: Blurring ∈ {5.0, 10.0, 20.0, 30.0, 40.0}

▷ **Angle** (input_control) . . . . . integer ⇝ *integer*
Winkel zwischen der Bewegungsrichtung und der x-Achse (gegen den Uhrzeigersinn).
Defaultwert: 0
Wertevorschläge: `Angle` ∈ {0, 45, 90, 180, 270}

▷ **Type** (input_control) . . . . . integer ⇝ *integer*
Impulsantwort-Prototyp bzw. Form der Bewegung.
Defaultwert: 3
Werteliste: `Type` ∈ {1, 2, 3, 4, 5}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `simulate_motion` den Wert TRUE. Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

`gen_psf_motion`, `gen_psf_motion`

Nachfolger

`simulate_defocus`, `wiener_filter`, `wiener_filter_ni`

Siehe auch

`gen_psf_motion`, `simulate_defocus`, `gen_psf_defocus`

Literatur

Anil K. Jain:Fundamentals of Digital Image Processing, Prentice-Hall International Inc., Englewood Cliffs, New Jersey, 1989

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Kha-Chye Tan, Hock Lim, B. T. G. Tan:„Restoration of Real-World Motion-Blurred Images“;S. 291-299 in: CVGIP Graphical Models and Image Processing, Vol. 53, No. 3, May 1991

Modul

Foundation

`wiener_filter ( Image, Psf, FilteredImage : RestoredImage : : )`

Bildrestauration mittels Wiener-Filterung.

`wiener_filter` liefert eine Abschätzung des unbeeinträchtigten Originalbildes. Die Abschätzung ist im Sinne eines minimalen, quadratischen Fehlers zwischen dem Originalbild und seiner Abschätzung optimal. Diese Abschätzung stellt eine restaurierte Version eines durch verschiedene Einflüsse (z.B. relative Bewegung zwischen aufzunehmendem Objekt und Aufnahmesystem, falsch fokussiertes Linsensystem, atmosphärische Störungen usw.) beeinträchtigten Bildes dar. Ausgangspunkt ist das beeinträchtigte Bild, das als Ausgabe eines gestörten, linearen Systems interpretiert wird. Das Verhalten eines linearen Systems, ist durch seine Impulsantwort bestimmt. Folgerichtig wird davon ausgegangen, dass das beeinträchtigte Bild das Ergebnis einer Faltung von Originalbild und Impulsantwort des linearen Systems darstellt. Diese Impulsantwort beschreibt die Bildaufnahme und die dabei aufgetretenen Störungen. Um auch Störungsursachen stochastischer Natur (Rauschen) einbeziehen zu können, wird zu dem Faltungsergebnis ein Rauschterm addiert. Insgesamt ergibt sich also das beeinträchtigte Bild aus

$$[Faltung(ImpulsantwortundOriginalbild)] + Rauschterm$$

Der Rauschterm beinhaltet zwei Komponenten, von denen jeweils eine das bildabhängige und bildunabhängige Rauschen beschreibt. Die Routine `wiener_filter` benötigt eine Bildversion, die (in der Vorverarbeitung durch den Anwender) mit einem Rauschfilter bearbeitet wurde. Diese rauschreduzierte Bildversion wird von `wiener_filter` dafür verwendet, die spektralen Leistungsdichten vom Rauschen und dem Originalbild abzuschätzen. Weiterhin ist durch den Anwender eine Impulsantwort zu erzeugen, die die spezielle Beeinträchtigung

im Bild beschreibt. Die Impulsantwort wird als Bildmatrix eingegeben (Bildtyp: 'real'). Sie stellt die Impulsantwort im Ortsraum dar und muss dahingehend in die Bildmatrix eingetragen werden, dass der Nullpunkt in der Bildecke „links oben" liegt. Das bedeutet folgende Einteilung der insgesamt NxM-großen Bildmatrix in vier rechteckige Regionen:

- Erstes Rechteck („links oben"): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem vierten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 0..N/2$ und $y = 0.. - M/2$
- Zweites Rechteck („rechts oben"): (Bildkoordinaten $xb = N/2..N - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem dritten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = -1.. - M/2$
- Drittes Rechteck („links unten"): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = M/2..M - 1$)
  -entspricht dem ersten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 1..N/2$ und $y = M/2..0$
- Viertes Rechteck („rechts unten"): (Bildkoordinaten $xb = N/2..N - 1$, $yb = M/2..M - 1$)
  -entspricht dem zweiten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = M/2..1$

`wiener_filter` verwendet folgende Strategie zur Erzeugung des restaurierten Bildes:

- Abschätzen der spektralen Leistungsdichte des Originalbildes aus der des rauschgefilterten Eingabebildes.
- Subtrahieren der Bildwerte des rauschgefilterten Eingabebildes von denen des belassenen Eingabebildes ergibt eine Abschätzung der Rauschstärke an jedem Bildpunkt. Damit lässt sich die spektrale Leistungsdichte des Rauschens für jeden Bildpunkt abschätzen.
- Mit Hilfe des Quotienten der Leistungsdichten vom Rauschen und Originalbild und der Fouriertransformierten der Impulsantwort wird das Wiener-Filter gebildet.
- Durchführung der Wiener-Filterung durch Faltung von dem Eingabebild und der Frequenzantwort des Filters (= Multiplikation von deren diskreten Fouriertransformierten).
- Rücktransformiertes Ergebnis stellt das restaurierte Bild dar.

Das Restaurationsergebnis ist vom Bildtyp 'real'.

Achtung

Die Impulsantwort `Psf` muss vom Bildtyp 'real' sein und dieselbe Bildbreite und -länge wie `Image` und `FilteredImage` besitzen.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Beeinträchtigtes Bild, das gefiltert werden soll.

▷ **`Psf`** (input_object) ..... (multichannel-)image ⇝ *object :* real
Impulsantwort (Point Spread Function) der Bildbeeinträchtigung (im Ortsraum mit dem Nullpunkt in der Bildecke angegeben).

▷ **`FilteredImage`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Beeinträchtigtes, durch ein Rauschfilter vorverarbeitetes Bild, das gefiltert werden soll.

▷ **`RestoredImage`** (output_object) ..... image ⇝ *object :* real
Restauriertes Bild.

Beispiel

```
/* Restoration of a noisy image (size=256x256), that was blurred by motion*/
Hobject object;
Hobject restored;
Hobject psf;
Hobject noisefiltered;
/* 1. Generate a Point-Spread-Function for a motion-blur with       */
/*    parameter a=10 and direction along the x-axis                 */
```

```
gen_psf_motion(&psf,256,256,10,0,3);
/* 2. Noisefiltering of the image                                    */
median_image(object,&noisefiltered,"circle",2,0);
/* 3. Wiener-filtering                                               */
wiener_filter(object,psf,noisefiltered,&restored);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `wiener_filter` den Wert TRUE. Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

`gen_psf_motion`, `simulate_motion`, `simulate_defocus`, `gen_psf_defocus`, `optimize_fft_speed`

Alternativen

`wiener_filter_ni`

Siehe auch

`simulate_motion`, `gen_psf_motion`, `simulate_defocus`, `gen_psf_defocus`

Literatur

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995

Azriel Rosenfeld, Avinash C. Kak: Digital Picture Processing, Computer Science and Aplied Mathematics, Academic Press New York/San Francisco/London 1982

Modul

Foundation

wiener_filter_ni (Image, Psf, NoiseRegion : RestoredImage : MaskWidth, MaskHeight :)

Bildrestauration mittels Wiener-Filterung.

`wiener_filter_ni` (ni = noise-estimation integrated) liefert eine Abschätzung des unbeeinträchtigten Originalbildes. Die Abschätzung ist im Sinne eines minimalen, quadratischen Fehlers zwischen dem Originalbild und seiner Abschätzung optimal. Diese Abschätzung stellt eine restaurierte Version eines durch verschiedene Einflüsse (z.B. relative Bewegung zwischen aufzunehmendem Objekt und Aufnahmesystem, falsch fokussiertes Linsensystem, atmosphärische Störungen usw.) beeinträchtigten Bildes dar. Ausgangspunkt ist das beeinträchtigte Bild, das als Ausgabe eines gestörten, linearen Systems interpretiert wird. Das Verhalten eines linearen Systems, ist durch seine Impulsantwort bestimmt. Folgerichtig wird davon ausgegangen, dass das beeinträchtigte Bild das Ergebnis einer Faltung von Originalbild und Impulsantwort des linearen Systems darstellt. Diese Impulsantwort beschreibt die Bildaufnahme und die dabei aufgetretenen Störungen. Um auch Störungsursachen stochastischer Natur (Rauschen) einbeziehen zu können, wird zu dem Faltungsergebnis ein Rauschterm addiert. Insgesamt ergibt sich also das beeinträchtigte Bild aus

$$[Faltung(ImpulsantwortundOriginalbild)] + Rauschterm$$

Der Rauschterm beinhaltet zwei Komponenten, von denen jeweils eine das bildabhängige und bildunabhängige Rauschen beschreibt. Die Routine `wiener_filter_ni` schätzt die Stärke des Rauschens innerhalb einer Bildregion ab. Dies geschieht, indem mittels einer (ungewichteten) Median-Filterung (mit einer rechteckigen Filtermaske) eine rauschreduzierte Version der Bildregion erzeugt wird. Die Stärke des Rauschens ergibt sich punktweise durch Subtraktion des Median-Wertes vom dortigen Grauwert. Daraus wird die durchschnittliche Rauschamplitude innerhalb der Bildregion berechnet. Als zweites wird der durchschnittliche Grauwert des gesamten zu filternden Bildes gebildet. Hiermit lässt sich das durchschnittliche Verhältnis vom Rauschen und Bildwert abschätzen. Das Quadrat dieses Quotienten ergibt wiederum eine Abschätzung des Verhältnisses der spektralen Leistungsdichten

vom Rauschen und Bild. Bei der Wiener-Filterung mit `wiener_filter_ni` wird also (im Gegensatz zu der Version von `wiener_filter`) angenommen, dass dieses Verhältnis innerhalb des gesamten Bildraums näherungsweise konstant bleibt. Die Bildregion, die die Grundlage der Rauschabschätzung bildet, muss durch den Anwender in einer Vorverarbeitung bestimmt werden. Dabei ist zu beachten, dass eine möglichst homogene Region auszuwählen ist, da starke Texturen oder Kanten große Fehler bei der Rauschabschätzung verursachen können. Um dem Anwender einen (begrenzten) Einfluss auf die Abschätzung zu ermöglichen, sind die Breite bzw. Höhe der rechteckigen Filtermaske als Eingabe-Steuerparameter anzugeben. Weiterhin ist durch den Anwender eine Impulsantwort zu erzeugen, die die spezielle Beeinträchtigung im Bild beschreibt. Die Impulsantwort wird als Bildmatrix eingegeben (Bildtyp: 'real'). Sie stellt die Impulsantwort im Ortsraum dar und muss dahingehend in die Bildmatrix eingetragen werden, dass der Nullpunkt in der Bildecke „links oben“ liegt. Das bedeutet folgende Einteilung der insgesamt NxM-großen Bildmatrix in vier rechteckige Regionen:

- Erstes Rechteck („links oben“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem vierten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 0..N/2$ und $y = 0.. - M/2$
- Zweites Rechteck („rechts oben“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem dritten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = -1.. - M/2$
- Drittes Rechteck („links unten“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = M/2..M - 1$)
  -entspricht dem ersten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 1..N/2$ und $y = M/2..0$
- Viertes Rechteck („rechts unten“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = M/2..M - 1$)
  -entspricht dem zweiten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = M/2..1$

`wiener_filter_ni` verwendet folgende Strategie zur Erzeugung des restaurierten Bildes:

- Abschätzen des durchschnittlichen Verhältnisses der spektralen Leistungsdichten vom Rauschen und Originalbild (siehe oben). - Mit Hilfe dieses Verhältnisses und der Fouriertransformierten der Impulsantwort wird das Wiener-Filter gebildet. - Durchführung der Wiener-Filterung durch Faltung von dem Eingabebild und der Frequenzantwort des Filters (= Multiplikation von deren diskreten Fouriertransformierten). - Rücktransformiertes Produkt stellt das restaurierte Bild dar.

Das Restaurationsergebnis ist vom Bildtyp 'real'.

Achtung

Die Impulsantwort `Psf` muss vom Bildtyp 'real' sein und dieselbe Bildbreite und -länge wie `Image` besitzen. Die Bildregion zur Rauschabschätzung darf keine Bereiche außerhalb des Bildes umfassen. Die Filtermaskengröße `MaskWidth` und `MaskHeight` muss größer oder gleich 0 und kleiner oder gleich der Bildgröße sein. Werden für `MaskWidth` und `MaskHeight` gerade statt ungerader Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Beeinträchtigtes Bild, das gefiltert werden soll.

▷ **`Psf`** (input_object) ..... (multichannel-)image ⇝ *object :* real
Impulsantwort (Point Spread Function) der Bildbeeinträchtigung (im Ortsraum mit dem Nullpunkt in der Bildecke angegeben).

▷ **`NoiseRegion`** (input_object) ..... region(-array) ⇝ *object*
Bildregion, innerhalb der die durchschnittliche Stärke des Rauschens abgeschätzt werden soll.

▷ **`RestoredImage`** (output_object) ..... image ⇝ *object :* real
Restauriertes Bild

▷ **`MaskWidth`** (input_control) ..... integer ⇝ *integer*
Breite der rechteckigen Filtermaske, die bei der Median-Filterung zur internen Rauschabschätzung verwendet wird.
Defaultwert: 3
Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9}
Typischer Wertebereich: $0 \leq$ `MaskWidth` $\leq$ width(Image)

▷ **MaskHeight** (input_control) ..... integer ⇝ *integer*
Höhe der rechteckigen Filtermaske, die bei der Median-Filterung zur internen Rauschabschätzung verwendet wird.
Defaultwert: 3
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9}
Typischer Wertebereich: 0 ≤ `MaskHeight` ≤ height(Image)

Beispiel

```
/* Restoration of a noisy image (size=256x256), that was blurred by motion*/
Hobject object;
Hobject restored;
Hobject psf;
Hobject noise_region;
/* 1. Generate a Point-Spread-Function for a motion-blur with       */
/*    parameter a=10 and direction of the x-axis                    */
gen_psf_motion(&psf,256,256,10,0,3);
/* 2. Segmentation of a region for the noise-estimation             */
open_window(0,0,256,256,0,"visible",&WindowHandle);
disp_image(object,WindowHandle);
draw_region(&noise_region,draw_region);
/* 3. Wiener-filtering                                              */
wiener_filter_ni(object,psf,noise_region,&restored,3,3);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `wiener_filter_ni` den Wert TRUE. Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

`gen_psf_motion`, `simulate_motion`, `simulate_defocus`, `gen_psf_defocus`, `optimize_fft_speed`

Alternativen

`wiener_filter`

Siehe auch

`simulate_motion`, `gen_psf_motion`, `simulate_defocus`, `gen_psf_defocus`

Literatur

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995

Azriel Rosenfeld, Avinash C. Kak: Digital Picture Processing, Computer Science and Aplied Mathematics, Academic Press New York/San Francisco/London 1982

Modul

Foundation

Kapitel 11

Graphik

11.1 3D-Szene

add_scene_3d_camera (: : Scene3D, CameraParam : CameraIndex)

Hinzufügen einer Kamera zu einer 3D-Szene.

add_scene_3d_camera fügt eine Kamera zur 3D-Szene Scene3D hinzu und gibt den Index der Kamera in CameraIndex zurück. Die Kameraparameter der Kamera werden in CameraParam übergeben.

Zunächst befindet sich die neue Kamera am Ursprung des Weltkoordinatensystems. Ihre Pose kann mit set_scene_3d_camera_pose gesetzt werden.

Achtung

Kameras mit hyperzentrischen Objektiven werden nicht unterstützt. Falls CameraParam Verzeichnungskoeffizienten ungleich null enthält ist bei der Projektion großer Dreiecke oder Polygone (oder 3D-Primitive) darauf zu achten, dass die Verzeichnung nur auf die Punkte des Objektmodells angewandt werden. In der Projektion werden diese wiederum mit geraden Linien verbunden. Für eine gute Approximation der verzeichneten Linien sollte eine Triangulierung mit ausreichend kleinen Dreiecken verwendet werden.

Parameter

- ▷ **Scene3D** (input_control) . . . . . scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
  Parameter der neuen Kamera.
- ▷ **CameraIndex** (output_control) . . . . . integer ⇝ *integer*
  Index der neuen Kamera in der 3D-Szene.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_scene_3d_camera den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_scene_3d

Nachfolger

set_scene_3d_camera_pose, display_scene_3d

Modul

3D Metrology

add_scene_3d_instance (: : Scene3D, ObjectModel3D, Pose : InstanceIndex)

Hinzufügen einer Instanz eines 3D-Objektmodells zu einer 3D-Szene.

add_scene_3d_instance fügt eine Instanz des 3D-Objektmodells ObjectModel3D zur 3D-Szene Scene3D hinzu und gibt den Index der Instanz in InstanceIndex zurück. Werden mehrere Instanzen auf einmal hinzugefügt, kann entweder eine Pose für alle Instanzen oder für jede Instanz eine Pose übergeben werden.

Die Pose der Instanz im Koordinatensystem der Szene wird in Pose übergeben. Mit dem Operator set_scene_3d_instance_pose wird diese Pose geändert. Solange keine Pose für die Szene mit set_scene_3d_to_world_pose gesetzt wurde, bezieht sich diese Pose auf das Weltkoordinatensystem.

Mit set_scene_3d_instance_param können Parameter (z.B. Farbe) der Instanz verändert werden.

Parameter

- ▷ **Scene3D** (input_control) . . . . . scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **ObjectModel3D** (input_control) . . . . . object_model_3d(-array) ⇝ *handle*
  Handle des 3D-Objektmodells.
- ▷ **Pose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
  Pose des 3D-Objektmodells.
  Parameteranzahl: 7
- ▷ **InstanceIndex** (output_control) . . . . . integer ⇝ *integer*
  Index der neuen Instanz in der 3D-Szene.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_scene_3d_instance den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_scene_3d

Nachfolger

set_scene_3d_instance_param, set_scene_3d_instance_pose

Siehe auch

remove_scene_3d_instance

Modul

3D Metrology

add_scene_3d_label (: : Scene3D, Text, ReferencePoint, Position, RelatesTo : LabelIndex)

Hinzufügen eines Labels zu einer 3D-Szene.

add_scene_3d_label fügt ein Label mit dem Inhalt Text zur 3D-Szene Scene3D hinzu und gibt den Index des Labels in LabelIndex zurück.

Der Bezugspunkt des Labels wird als *[X,Y,Z]*-Punkt im Koordinatensystem der 3D-Szene in ReferencePoint festgelegt. Es ist möglich, ein Label ohne Bezugspunkt zu erzeugen, indem ein leeres Tupel übergeben wird. Dies ist allerdings nur möglich, wenn RelatesTo auf *'point'* gesetzt wird (siehe die Beschreibung von RelatesTo weiter unten).

Werden mehrere Texte in Text als Tupel übergeben, werden mehrere Labels erzeugt. Es ist möglich einen Bezugspunkt für alle Labels oder spezifische Bezugspunkte für jedes Label festzulegen. Das gilt auch für Position und RelatesTo.

Labels können fest in Fensterkoordinaten oder relativ zur Projektion des Bezugspunktes positioniert werden. Dafür werden die beiden Parameter `Position` und `RelatesTo` verwendet:

Feste Positionierung

Wird `RelatesTo` auf *'window'* gesetzt, wird das Label an einer festen Position im Fenster (bzw. gerenderten Bild) dargestellt, auch wenn sich die Pose der 3D-Szene ändert. Es gibt zwei Möglichkeiten, die Position festzulegen. Es können entweder die *[Zeile,Spalte]*-Koordinaten der linken oberen Ecke des Labels in `Position` übergeben werden oder eine der folgenden vordefinierten Positionen:

'top_left'	*'top'*	*'top_right'*
'left'	*'center'*	*'right'*
'bottom_left'	*'bottom'*	*'bottom_right'*

Relative Positionierung

Wird `RelatesTo` auf *'point'* gesetzt, wird das Label an einer Position relativ zur Projektion des Bezugspunkts dargestellt. Auf diese Weise bewegt sich das Label mit dem Bezugspunkt, wenn die Pose der 3D-Szene geändert wird. Es gibt wiederum zwei Möglichkeiten die Position festzulegen. Es kann entweder der Offset (in *[Zeile,Spalte]*-Koordinaten) der linken oberen Ecke des Labels oder eine der folgenden vordefinierten Positionen in `Position` übergeben werden:

'top_left'	*'top'*	*'top_right'*
'left'	*'center'*	*'right'*
'bottom_left'	*'bottom'*	*'bottom_right'*

Mit `set_scene_3d_label_param` können diese und andere Parameter wie z.B. Farbe und Schriftart des Labels verändert werden.

Parameter

▷ **`Scene3D`** (input_control) . . . . . scene_3d ⇝ *handle*
Handle der 3D-Szene.

▷ **`Text`** (input_control) . . . . . string(-array) ⇝ *string*
Text des Labels.
Defaultwert: 'label'

▷ **`ReferencePoint`** (input_control) . . . . . point-array ⇝ *real* / integer
Bezugspunkt des Labels.
Parameteranzahl: 3

▷ **`Position`** (input_control) . . . . . point-array ⇝ *string* / real / integer
Position im Fenster
Defaultwert: 'top'
Werteliste: `Position` ∈ {'top_left', 'top', 'top_right', 'left', 'center', 'right', 'bottom_left', 'bottom', 'bottom_right'}

▷ **`RelatesTo`** (input_control) . . . . . string-array ⇝ *string*
Feste oder relative Positionierung
Parameteranzahl: 1
Defaultwert: 'point'
Werteliste: `RelatesTo` ∈ {'point', 'window'}

▷ **`LabelIndex`** (output_control) . . . . . integer ⇝ *integer*
Index des neuen Labels in der 3D-Szene.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_scene_3d_label` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_scene_3d`

Nachfolger

`set_scene_3d_label_param`

Siehe auch

`remove_scene_3d_instance`

Modul

3D Metrology

add_scene_3d_light (: : Scene3D, LightPosition, LightKind : LightIndex)

Hinzufügen einer Lichtquelle zu einer 3D-Szene.

`add_scene_3d_light` fügt ein neue Lichtquelle zur 3D-Szene `Scene3D` hinzu und gibt ihren Index in `LightIndex` zurück. Der Typ der Lichtquelle wird in `LightKind` festgelegt. Für `LightKind` = *'point_light'* entspricht `LightPosition` der Position der Lichtquelle. Für `LightKind` = *'directional_light'* entspricht `LightPosition` dem Vektor der gerichteten Lichtquelle.

Momentan wird nur eine Lichtquelle unterstützt. Deshalb überschreibt `add_scene_3d_light` die bestehende Lichtquelle. Dieses Verhalten kann sich in zukünftigen Versionen ändern.

Wird keine Lichtquelle gesetzt, wird eine Punktlichtquelle an der Position [-100.0, -100.0, 0.0] verwendet.

Parameter

- ▷ **Scene3D** (input_control) . . . . . scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **LightPosition** (input_control) . . . . . real-array ⇝ *real* / integer
  Position der neuen Lichtquelle.
  Defaultwert: [-100.0,-100.0,0.0]
- ▷ **LightKind** (input_control) . . . . . string ⇝ *string*
  Typ der neuen Lichtquelle.
  Defaultwert: 'point_light'
  Werteliste: `LightKind` ∈ {'point_light', 'directional_light'}
- ▷ **LightIndex** (output_control) . . . . . integer ⇝ *integer*
  Index der neuen Lichtquelle in der 3D-Szene.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_scene_3d_light` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_scene_3d`

Nachfolger

`set_scene_3d_light_param`

Siehe auch

`remove_scene_3d_light`

Modul

3D Metrology

clear_scene_3d (: : Scene3D :)

Freigeben einer 3D-Szene und des zugehörigen Speichers.

clear_scene_3d gibt die 3D-Szene Scene3D frei.

Parameter

▷ **Scene3D** (input_control) . . . . . scene_3d(-array) ⇝ *handle*
Handle der 3D-Szene.

Ergebnis

Konnte die 3D-Szene freigegeben werden, liefert clear_scene_3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- Scene3D

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

display_scene_3d

Modul

3D Metrology

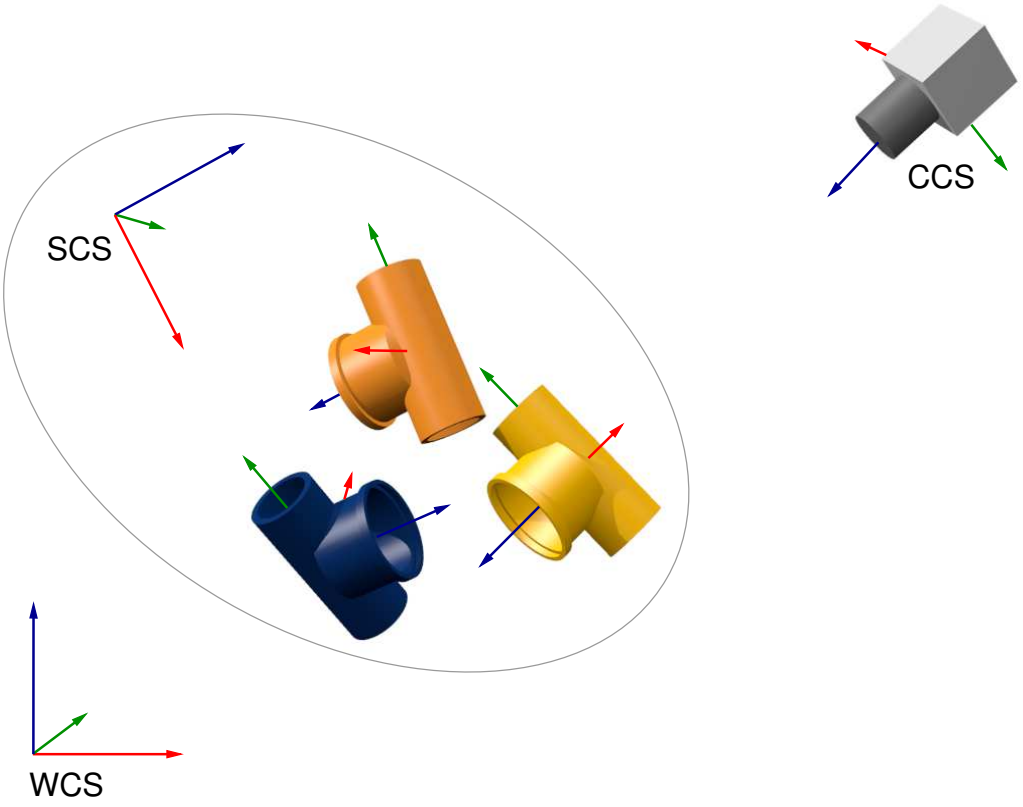
create_scene_3d (: : : Scene3D)

Erzeugen der Datenstruktur um eine Sammlung von 3D-Objekten zu visualisieren.

create_scene_3d erzeugt eine neue 3D-Szene und gibt sie in Scene3D zurück.

Eine 3D-Szene ist eine Sammlung von 3D-Objektmodellen, Kameras und Lichtquellen. Diese Objekte können mit den Operatoren add_scene_3d_instance, add_scene_3d_camera und add_scene_3d_light zur 3D-Szene Scene3D hinzugefügt werden. Mit display_scene_3d wird eine 3D-Szene in einem Fenster dargestellt.

Jeder Instanz ist eine Pose zugeordnet (mit add_scene_3d_instance oder set_scene_3d_instance_pose), die die Position der Instanz im Koordinatensystem der Szene (SCS) darstellt. Die Pose der Szene im Weltkoordinatensystem WCS kann mit set_scene_3d_to_world_pose gesetzt werden. Die Pose der Kamera im Weltkoordinatensystem wird mit set_scene_3d_camera_pose gesetzt und definiert das Kamerakoordinatensystem CCS.



Die Koordinatensysteme einer 3D-Szene

Parameter, welche die gesamte Szene beeinflussen (z.B. die Qualität des Renderings), werden mit `set_scene_3d_param` gesetzt.

Parameter

▷ **`Scene3D`** (output_control) . . . . . scene_3d ⇝ *handle*
Handle der 3D-Szene.

Beispiel

```
* Create a new scene
create_scene_3d (Scene)
* A scene needs at least one camera. The default pose
* of a camera is located at the origin. The pose can be
* changed with set_scene_3d_camera_pose.
add_scene_3d_camera (Scene, CameraParam, CameraIndex)
* Further a scene needs at least one light.
add_scene_3d_light (Scene, [42.0, 42.0, 42.0], 'point_light', LightIndex)
*
* To add an object, add_scene_3d_instance is called with a
* 3d object model and a pose. add_scene_3d returns an InstanceIndex
* which must be used to reference this instance in subsequent calls.
add_scene_3d_instance (Scene, ObjectModel3D, Pose, InstanceIndex)
* Set its color.
set_scene_3d_instance_param (Scene, InstanceIndex, 'color', 'green')
* Display the scene.
display_scene_3d (WindowHandle, Scene, CameraIndex)
clear_scene_3d (Scene)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_scene_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`add_scene_3d_instance`, `add_scene_3d_light`, `add_scene_3d_camera`, `set_scene_3d_param`

Siehe auch

`clear_scene_3d`

Modul

3D Metrology

`display_scene_3d` (: : `WindowHandle`, `Scene3D`, `CameraIndex` :)

Anzeigen einer 3D-Szene.

`display_scene_3d` zeigt die 3D-Szene `Scene3D` im Fenster `WindowHandle` aus der Sicht der Kamera mit dem Index `CameraIndex` an. Ein Beispiel in der Dokumentation von `create_scene_3d` zeigt einen kurzen Ablauf.

`display_scene_3d` benötigt OpenGL 2.1, GLSL 1.2 und die OpenGL Extensions GL_EXT_framebuffer_object und GL_EXT_framebuffer_blit. Andernfalls wird automatisch der Kompatibilitätsmodus mit geringeren Anforderungen an OpenGL benutzt.

Bei Grafikkarten mit geringem Speicher können bei großer Auflösung des Renderfensters folgende Fehlermeldungen auftreten:

Low-Level Fehler: 'Incomplete attachment'

Unbehandelte Ausnahme: 'Benötigtes Framebuffer Objekt wird nicht unterstützt'

Lösungen:

Um die Anforderungen an den Speicher der Grafikkarte zu verringern, kann mit `set_scene_3d_param` der Parameter *'quality'* auf *'low'* gesetzt oder der Kompatibilitätsmodus verwendet werden.

Die Systemvariable (siehe `set_system`) *'opengl_compatibility_mode_enable'* kann auf *'true'* gesetzt werden, um den Kompatibilitätsmodus permanent einzuschalten. Dieser Modus benötigt OpenGL 1.1.

Im Kompatibilitätsmodus werden die Parameter *'object_index_persistence'*, *'depth_persistence'* und *'quality'* nicht unterstützt.

Auf Linux Remote-Desktop wird *'disp_background'* nicht unterstützt.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fensteridentifikator.
- ▷ **`Scene3D`** (input_control) ..... scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **`CameraIndex`** (input_control) ..... integer ⇝ *string* / integer
  Index der Kamera zum Anzeigen der 3D-Szene.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `display_scene_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`create_scene_3d`, `render_scene_3d`

Modul

3D Metrology

`get_display_scene_3d_info` `( : : WindowHandle, Scene3D, Row, Column, Information : Value )`

Zurückgeben der Tiefe oder des Indexes eines dargestellten 3D-Objektmodells.

`get_display_scene_3d_info` gibt Informationen über die 3D-Objektmodelle in der 3D-Szene `Scene3D`, die im Fenster `WindowHandle` an den Positionen (`Row`, `Column`) mit `display_scene_3d` angezeigt wurden in `Value` zurück.

Mit dem Parameter `Information` wird ausgewählt, welche Werte zurückgegeben werden:

'object_index' Die Indices der 3D-Objektmodelle, die an den Positionen (`Row`, `Column`) dargestellt wurden. Befindet sich kein 3D-Objektmodell an einer dieser Stellen, wird -1 zurückgegeben. Um diese Information ermitteln zu können, muss der Parameter *'object_index_persistence'* mit `set_scene_3d_param` auf *'true'* gesetzt worden sein.

'depth' Die Tiefe (d.h. die Z-Koordinate im Kamerakoordinatensystem) an den Positionen (`Row`, `Column`). Befindet sich kein 3D-Objektmodell an einer dieser Stellen, wird -1.0 zurückgegeben. Um diese Information ermitteln zu können, muss der Parameter *'depth_persistence'* mit `set_scene_3d_param` auf *'true'* gesetzt worden sein.

Die Fensterkoordinaten `Row` und `Column` müssen so angegeben werden, dass sie dem gesetzten Bildausschnitt entsprechen. Sie können daher auch subpixelgenau sein. Ist der Bildausschnitt durch (Zeile1,Spalte1,Zeile2, Spalte2) gegeben, entspricht die linke obere Ecke den Koordinaten (Zeile1 - 0.5, Spalte1 - 0.5), die rechte untere Ecke den Koordinaten (Zeile2 - 0.5, Spalte2 - 0.5). Die Operatoren `get_mposition_sub_pix` oder `get_mbutton_sub_pix` können verwendet werden, um diese Koordinaten direkt zu erhalten.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fensteridentifikator.
- ▷ **`Scene3D`** (input_control) ..... scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **`Row`** (input_control) ..... integer(-array) ⇝ *real* / integer
  Zeilenkoordinaten.
- ▷ **`Column`** (input_control) ..... integer(-array) ⇝ *real* / integer
  Spaltenkoordinaten.
- ▷ **`Information`** (input_control) ..... string(-array) ⇝ *string*
  Abzufragende Information.
  Defaultwert: 'depth'
  Werteliste: `Information` ∈ {'depth', 'object_index'}
- ▷ **`Value`** (output_control) ..... integer(-array) ⇝ *integer* / real
  Werte an (Row,Column).

Ergebnis

`get_display_scene_3d_info` liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

display_scene_3d, get_mbutton, get_mbutton_sub_pix, get_mposition, get_mposition_sub_pix

Siehe auch

display_scene_3d

Modul

3D Metrology

remove_scene_3d_camera (: : Scene3D, CameraIndex :)

Entfernen einer Kamera aus einer 3D-Szene.

remove_scene_3d_camera entfernt die Kamera CameraIndex aus der 3D-Szene Scene3D.

Parameter

- ▷ **Scene3D** (input_control) ..... scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **CameraIndex** (input_control) ..... integer ⇝ *integer*
  Index der zu entfernenden Kamera.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert remove_scene_3d_camera den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

add_scene_3d_camera

Modul

3D Metrology

remove_scene_3d_instance (: : Scene3D, InstanceIndex :)

Entfernen einer Objektinstanz aus einer 3D-Szene.

remove_scene_3d_instance entfernt die Instanz InstanceIndex aus der 3D-Szene Scene3D.

Parameter

- ▷ **Scene3D** (input_control) ..... scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **InstanceIndex** (input_control) ..... integer(-array) ⇝ *integer*
  Index der zu entfernenden Instanz.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert remove_scene_3d_instance den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

add_scene_3d_instance

Modul

3D Metrology

remove_scene_3d_label (: : Scene3D, LabelIndex :)

Entfernen eines Labels aus einer 3D-Szene.

remove_scene_3d_label entfernt das Label LabelIndex aus der 3D-Szene Scene3D.

Parameter

- ▷ **Scene3D** (input_control) ..... scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **LabelIndex** (input_control) ..... integer(-array) ⇝ *integer*
  Index des zu entfernenden Textlabels.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert remove_scene_3d_label den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

add_scene_3d_label

Modul

3D Metrology

remove_scene_3d_light (: : Scene3D, LightIndex :)

Entfernen einer Lichtquelle aus einer 3D-Szene.

remove_scene_3d_light entfernt in zukünftigen Versionen die Lichtquelle LightIndex aus der 3D-Szene Scene3D. Momentan wird nur eine Lichtquelle unterstützt

Parameter

- ▷ **Scene3D** (input_control) ..... scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **LightIndex** (input_control) ..... integer ⇝ *integer*
  Index der zu entfernenden Lichtquelle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert remove_scene_3d_light den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

add_scene_3d_light

Modul

3D Metrology

render_scene_3d (: Image : Scene3D, CameraIndex :)

Rendern eines Bildes einer 3D-Szene.

render_scene_3d rendert ein Bild der 3D-Szene Scene3D aus der Sicht der Kamera mit dem Index CameraIndex und gibt es in Image zurück.

Anders als in display_scene_3d werden die Parameter *'object_index_persistence'* und *'disp_background'* nicht berücksichtigt. Der Hintergrund von Image ist schwarz.

render_scene_3d benötigt OpenGL 2.1, GLSL 1.2 und die OpenGL Extensions GL_EXT_framebuffer_object und GL_EXT_framebuffer_blit. Andernfalls wird automatisch der Parameter *'opengl_compatibility_mode_enable'* der Systemvariable (siehe set_system) auf *'true'* gesetzt und die Visualisierung mit geringeren Anforderungen durchgeführt. Hierfür wird OpenGL 1.1 benötigt.

Parameter

▷ **Image** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte
Bild der 3D-Szene.

▷ **Scene3D** (input_control) . . . . . scene_3d ⇝ *handle*
Handle der 3D-Szene.

▷ **CameraIndex** (input_control) . . . . . integer ⇝ *integer*
Index der Kamera zum rendern der 3D-Szene.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert render_scene_3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

display_scene_3d

Siehe auch

create_scene_3d

Modul

3D Metrology

set_scene_3d_camera_pose (: : Scene3D, CameraIndex, Pose :)

Setzen der Pose einer Kamera in einer 3D-Szene.

set_scene_3d_camera_pose setzt die Pose der Kamera mit dem Index CameraIndex in der 3D-Szene Scene3D.

Die Koordinatensysteme der 3D-Szene sind in der Dokumentation von create_scene_3d beschrieben.

Parameter

▷ **Scene3D** (input_control) . . . . . scene_3d ⇝ *handle*
Handle der 3D-Szene.

▷ **CameraIndex** (input_control) . . . . . integer ⇝ *integer*
Index der Kamera.

▷ **Pose** (input_control) . . . . . pose ⇝ *real* / integer
Neue Pose der Kamera.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_scene_3d_camera_pose` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_scene_3d_camera`

Siehe auch

`set_scene_3d_to_world_pose`, `set_scene_3d_instance_pose`

Modul

3D Metrology

set_scene_3d_instance_param (: : Scene3D, InstanceIndex, GenParamName, GenParamValue :)

Setzen von Parameter einer Instanz in einer 3D-Szene.

`set_scene_3d_instance_param` setzt die Parameter der Instanz mit dem Index `InstanceIndex` in der 3D-Szene `Scene3D`. Name und Wert eines Parameters müssen in `GenParamName` und `GenParamValue` übergeben werden. Alle Parameter werden auf alle Instanzen angewandt.

Die folgenden Werte können gesetzt werden:

'visible' Sichtbarkeit des 3D-Objektmodells. Wird *'visible'* auf *'false'* gesetzt, wird diese Instanz nicht angezeigt.
Werteliste: 'true', 'false'.
Standardwert: 'true'.

'attribute' Wählt explizit die Art und Weise aus, wie ein 3D-Objektmodell visualisiert wird.
Werteliste: 'auto', 'faces', 'primitive', 'points', 'lines'.
Standardwert: 'auto'.

'color': Farbe des 3D-Objektmodells. Die verfügbaren Farben können mit dem Operator `query_color` erfragt werden. Darüberhinaus kann die Farbe auch über ein RGB-Tripel in der Form *'#rrggbb'* gesetzt werden, wobei 'rr', 'gg' und 'bb' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind.
Wertevorschläge: 'red', 'green'.
Standardwert: 'white'.

'alpha': Transluzenz des 3D-Objektmodells. Das Darstellen von 3D-Objektmodellen mit Transluzenz kleiner *1.0* kann zu signifikant höherer Laufzeit führen.
Mögliche Werte: Fließkommawerte zwischen *0.0* (transparent) und *1.0* (opak).
Standardwert: 1.0.

'disp_pose': Flag, ob die Pose des 3D-Objektmodells visualisiert werden soll.
Werteliste: 'true', 'false'.
Standardwert: 'false'.

'disp_lines': Flag, ob die Konturen der Polygone des 3D-Objektmodells angezeigt werden sollen.

Werteliste: 'true', 'false'.

Standardwert: 'false'.

'disp_normals': Flag, ob die Oberflächennormalen der Polygone des 3D-Objektmodells angezeigt werden sollen.

Werteliste: 'true', 'false'.

Standardwert: 'false'.

'line_color': Farbe der Konturen, falls *'disp_lines'* auf *'true'* gesetzt ist. Die verfügbaren Farben können mit dem Operator `query_color` erfragt werden. Darüberhinaus kann die Farbe auch über ein RGB-Tripel in der Form *'#rrggbb'* gesetzt werden, wobei 'rr', 'gg' und 'bb' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind.

Wertevorschläge: 'red', 'green'.

Standardwert: Der Wert von *'color'*.

'line_width': Setzt die Breite von Linien in Pixel.

Standardwert: 1.0.

'normal_color': Farbe der Oberflächennormalen, falls *'disp_normals'* auf *'true'* gesetzt ist. Die verfügbaren Farben können mit dem Operator `query_color` erfragt werden. Darüberhinaus kann die Farbe auch über ein RGB-Tripel in der Form *'#rrggbb'* gesetzt werden, wobei 'rr', 'gg' und 'bb' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind.

Wertevorschläge: 'red', 'green'.

Standardwert: Der Wert von *'color'*.

'point_size': Setzt den Durchmesser der Punkte.

Standardwert: 3.5.

'lut': Setzt die LUT, die die Werte des mit *'color_attrib'* gesetzten Attributes in Farben umwandelt. In der Dokumentation von `set_lut` befindet sich eine Liste der verfügbaren LUTs. Ist *'lut'* auf etwas anderes als *'default'* gesetzt, wird *'color'* ignoriert.

Standardwert: 'default'.

'color_attrib': Name eines Punktattributes, das für die Falschfarbendarstellung verwendet wird.

Falls ein Attribut gesetzt ist, wird die Farbe eines dargestellten 3D-Punktes durch den Attributwert des Punktes und die aktuell gesetzte LUT bestimmt (siehe *'lut'*). Auf diese Weise ist es möglich, Attribute in Falschfarben darzustellen.

Beispiel: Wenn *'color_attrib'* auf *'coord_z'* und *'lut'* auf *'color1'* gesetzt ist, werden die Z-Koordinaten von rot nach blau farbkodiert.

Falls *'lut'* auf *'default'* gesetzt ist, werden die Attributwerte verwendet, um die mit *'color'* gesetzte Farbe zu skalieren.

Falls *'lut'* auf etwas anderes gesetzt, werden die Attributwerte intern auf den Bereich [0,255] skaliert und als Eingabe für die LUT-Funktion verwendet.

Die Abbildung wird außerdem von den Parametern *'color_attrib_start'* und *'color_attrib_end'* bestimmt (siehe unten).

Falls Flächen dargestellt werden, wird ihre Farbe zwischen den Eckpunktfarben interpoliert.

Werteliste: 'none', '&distance', 'coord_x', 'coord_y', 'coord_z', benutzerdefinierte Punktattribute, oder beliebige andere verfügbare Punktattribute.

Standardwert: 'none'.

'color_attrib_start','color_attrib_end': Der maßgebliche Wertebereich für das mit *'color_attrib'* eingestellte Attribut.

Die Attributwerte zwischen *'color_attrib_start'* und *'color_attrib_end'* werden auf den Anfang und das Ende der ausgewählten LUT skaliert. Attributwerte außerhalb des Bereiches werden beschnitten. Damit ist es möglich, einen festen Farbbereich darzustellen, der nicht durch Ausreißer verzerrt wird.

Bei *'auto'* wird der kleinste Attributwert auf den Anfang der LUT abgebildet, der größte Wert auf das Ende, es sei denn, *'color_attrib'* ist *'normal_x'*, *'normal_y'* oder *'normal_z'*. In diesem Fall, werden Start und Ende automatisch auf -1 und 1 gesetzt.

Der Startwert kann höher als der Endwert sein. In dem Fall wird die LUT umgedreht.

Wertevorschläge: 0, 0.1, 1, 100, 255.

Standardwert: 'auto'.

'red_channel_attrib', *'green_channel_attrib'*, *'blue_channel_attrib'*: Name eines Punktattributes, das für den Rot-, Grün-, oder Blaukanal verwendet wird.

Das ist besonders nützlich, wenn drei zusammengehörige Attribute verwendet werden, z.B. RGB-Farben oder Normalenvektoren. Auf diese Weise können Punkte mit farbigen Texturen, z.B. RGB-Sensor-Daten, dargestellt oder Punktnormalen in Falschfarben kodiert werden.

Um ein einzelnes Attribut in Falschfarben darzustellen, verwenden Sie bitte *'color_attrib'* (siehe oben).

Per Default wird angenommen, dass die Attributwerte zwischen 0 und 255 liegen. Falls die Attribute andere Wertebereiche haben, müssen zusätzlich die Parameter *'rgb_channel_attrib_start'* und *'rgb_channel_attrib_end'* gesetzt werden (siehe unten).

Falls nur 1 oder 2 Kanäle gesetzt sind, verwenden die übrigen Kanäle den RGB-Wert der mit *'color'* eingestellten Farbe.

Falls Flächen dargestellt werden, wird ihre Farbe zwischen den Eckpunktfarben interpoliert.

Mögliche Werte: 'none', *'&red'*, *'&green'*,*'&blue'*, *'normal_x'*, *'normal_y'*, *'normal_z'*, benutzerdefinierte Punktattribute, oder beliebige andere verfügbare Punktattribute.

Standardwert: 'none'.

'rgb_channel_attrib_start', *'rgb_channel_attrib_end'*: Der maßgebliche Wertebereich für die mit *'red_channel_attrib'*, *'green_channel_attrib'* und *'blue_channel_attrib'* eingestellten Attribute.

Diese Parameter definieren den Wertebereich, der auf die gesamten RGB-Kanäle skaliert wird. Das ist nützlich, wenn die Werte der Eingabeattribute nicht im Intervall [0,255] liegen.

Bei *'auto'* wird der kleinste Attributwert auf 0 abgebildet, der größte Wert auf 255, es sei denn, das Attribut ist *'normal_x'*, *'normal_y'* oder *'normal_z'*. In diesem Fall, werden Start und Ende automatisch auf -1 und 1 gesetzt.

Der Startwert kann höher als der Endwert sein. In dem Fall werden die RGB-Farben invertiert.

Der Bereich kann für jeden Kanal einzeln eingestellt werden, indem im Parameternamen das *'rgb'* durch den jeweiligen Kanal ersetzt wird, z.B. *'green_channel_attrib_start'*.

Wertevorschläge: 'auto', *0*, *0.1*, *1*, *100*, *255*.

Standardwert: 0, *255*.

Parameter

▷ **`Scene3D`** (input_control) ..... scene_3d ⇝ *handle*
Handle der 3D-Szene.

▷ **`InstanceIndex`** (input_control) ..... integer(-array) ⇝ *integer*
Index der Instanz.

▷ **`GenParamName`** (input_control) ..... string-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'color'
Werteliste: `GenParamName` ∈ {'alpha', 'attribute', 'color', 'disp_lines', 'disp_pose', 'disp_normals', 'line_color', 'line_width', 'normal_color', 'color_attrib', 'red_channel_attrib', 'green_channel_attrib', 'blue_channel_attrib', 'rgb_channel_attrib_start', 'rgb_channel_attrib_end', 'color_attrib_start', 'color_attrib_end', 'lut', 'visible', 'point_size'}

▷ **`GenParamValue`** (input_control) ..... string-array ⇝ *string* / integer / real
Werte der generischen Parameter.
Defaultwert: 'green'
Wertevorschläge: `GenParamValue` ∈ {'true', 'false', 'coord_x', 'coord_y', 'coord_z', 'normal_x', 'normal_y', 'normal_z', 'red', 'green', 'blue', 'auto', 'faces', 'primitive', 'points', 'lines'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_scene_3d_instance_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_scene_3d_instance

Siehe auch

set_scene_3d_param

Modul

3D Metrology

set_scene_3d_instance_pose (: : Scene3D, InstanceIndex, Pose :)

Setzen der Pose einer Instanz in einer 3D-Szene.

set_scene_3d_instance_pose setzt die Pose der Instanz mit dem Index InstanceIndex in der 3D-Szene Scene3D auf Pose. Es kann eine Pose für mehrere Instanzen gesetzt werden, oder für jede Instance eine Pose.

Die Koordinatensysteme der 3D-Szene sind in der Dokumentation von create_scene_3d beschrieben.

Parameter

- ▷ **Scene3D** (input_control) ..... scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **InstanceIndex** (input_control) ..... integer(-array) ⇝ *integer*
  Index der Instanz.
- ▷ **Pose** (input_control) ..... pose(-array) ⇝ *real* / integer
  Neue Pose der Instanz.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_scene_3d_instance_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_scene_3d_instance

Siehe auch

set_scene_3d_to_world_pose, set_scene_3d_camera_pose

Modul

3D Metrology

set_scene_3d_label_param (: : Scene3D, LabelIndex, GenParamName, GenParamValue :)

Setzen von Parametern eines Labels in einer 3D-Szene.

set_scene_3d_label_param setzt Parameter des Labels mit dem Index LabelIndex in der 3D-Szene Scene3D. Name und Wert eines Parameters werden in GenParamName und GenParamValue übergeben. Es ist möglich, ein Paar aus GenParamName und GenParamValue für einzelne oder mehrere Labels zu setzen.

Die folgenden Werte können gesetzt werden:

'reference_point': Koordinaten des Bezugspunkts als *[X, Y, Z]*-Koordinatentupel. Um einen Bezugspunkt zu entfernen, kann auch ein leeres Tupel übergeben werden. Das ist nur möglich wenn *'relates_to'* auf *'window'* gesetzt ist.

'position': Position (fest oder relativ, abhängig von *'relates_to'*) des Labels. Entweder als *[Row, Column]*-Koordinaten oder eine der folgenden vordefinierten Positionen: *'top_left'*, *'top'*, *'top_right'*, *'left'*, *'center'*, *'right'*, *'bottom_left'*, *'bottom'* oder *'bottom_right'*.

'relates_to': Art der Positionierung. Ist entweder *'window'* (feste Positionierung in Fensterkoordinaten) oder *'point'* (Positionierung relativ zum Bezugspunkt). *'relates_to'* kann nicht auf *'point'* gesetzt werden, wenn der Bezugspunkt auf einen leeren Tupel gesetzt wurde.

Werteliste: 'window', *'point'*.

Standardwert: 'window'.

'text': Text des Labels. Mehrere Zeilen werden mit *'\n'* getrennt.

'font': Schriftart des Textlabels.

Mögliche Werte: Verfügbare Schriftarten können mit `query_font` abgefragt werden.

Standardwert: Font des ersten Fensters, in dem die Szene dargestellt wird. Ansonsten die Default-Systemschriftart.

'text_color': Farbe des Texts. Die verfügbaren Farben können mit dem Operator `query_color` erfragt werden. Darüberhinaus kann die Farbe auch über ein RGB-Tripel in der Form *'#rrggbb'* gesetzt werden, wobei 'rr', 'gg' und 'bb' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind.

Wertevorschläge: 'red', *'green'*.

Standardwert: 'white'.

'alpha': Transparenz des Labelhintergrunds.

Mögliche Werte: Zahl zwischen *0.0* (voll transparent) und *1.0* (voll opak).

Standardwert: 1.0.

'disp_background': Flag, ob der Hintergrund des Labels dargestellt wird.

Werteliste: 'true', *'false'*.

Standardwert: 'false'.

'disp_connecting_line': Flag, ob die Verbindungslinie zwischen dem Label und dem entsprechenden 3D-Punkt angezeigt werden soll.

Werteliste: 'true', *'false'*.

Standardwert: 'false'.

'visible': Setzt die Sichtbarkeit des Labels. Wird *'visible'* auf *'if_point_is_visible'* gesetzt, ist das Label nur sichtbar, wenn der Bezugspunkt sichtbar ist. Im Kompatibilitätsmodus oder im Falle, dass kein Bezugspunkt gesetzt ist, verhält sich *'if_point_is_visible'* wie *'true'*.

Werteliste: 'true', *'false'*, *'if_point_is_visible'*.

Standardwert: 'false'.

Parameter

▷ **Scene3D** (input_control) ..... scene_3d ⇝ *handle*
Handle der 3D-Szene.

▷ **LabelIndex** (input_control) ..... integer(-array) ⇝ *integer*
Index des Textlabels.

▷ **GenParamName** (input_control) ..... string ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'color'
Werteliste: `GenParamName` ∈ {'reference_point', 'position', 'relates_to', 'color', 'text_color', 'font', 'text', 'disp_connecting_line', 'disp_background', 'visible'}

▷ **GenParamValue** (input_control) ..... string-array ⇝ *string* / integer / real
Werte der generischen Parameter.
Defaultwert: 'red'
Werteliste: `GenParamValue` ∈ {'red', 'green', 'blue', 'window', 'point', 'top_left', 'top', 'top_right', 'left', 'center', 'right', 'bottom_left', 'bottom', 'bottom_right'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_scene_3d_label_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_scene_3d_label`

Modul

3D Metrology

set_scene_3d_light_param (: : Scene3D, LightIndex, GenParamName, GenParamValue :)

Setzen von Parametern einer Lichtquelle in einer 3D-Szene.

`set_scene_3d_light_param` setzt die Parameter der Lichtquelle mit dem Index `LightIndex` in der 3D-Szene `Scene3D`. Name und Wert eines Parameters müssen in `GenParamName` und `GenParamValue` übergeben werden. Die folgenden Werte können gesetzt werden:

'light_ambient': Ambienter Anteil der Lichtquelle. Dieser Parameter muss als Tupel der Länge 3 übergeben werden.
Standardwert: [0.2, 0.2, 0.2]

'light_diffuse': Diffuser Anteil der Lichtquelle. Dieser Parameter muss als Tupel der Länge 3 übergeben werden.
Standardwert: [0.8 0.8 0.8]

Parameter

▷ **Scene3D** (input_control) . . . . . scene_3d ⇝ *handle*
Handle der 3D-Szene.

▷ **LightIndex** (input_control) . . . . . integer ⇝ *integer*
Index der Lichtquelle.

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'ambient'
Werteliste: `GenParamName` ∈ {'ambient', 'diffuse'}

▷ **GenParamValue** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
Werte der generischen Parameter.
Defaultwert: [0.2,0.2,0.2]
Wertevorschläge: `GenParamValue` ∈ {[0.2,0.2,0.2]}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_scene_3d_light_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_scene_3d_instance`

Modul

3D Metrology

set_scene_3d_param (: : Scene3D, GenParamName, GenParamValue :)

Setzen von Parametern einer 3D-Szene.

set_scene_3d_param setzt Parameter der 3D-Szene Scene3D. Name und Wert eines Parameters müssen in GenParamName und GenParamValue übergeben werden.

Alle Parameter, die mit set_scene_3d_instance_param gesetzt werden können, können auch mit set_scene_3d_param für alle Instanzen einer Szene gesetzt werden.

Zusätzlich können die folgenden Werte gesetzt werden:

'disp_background': Flag, ob der gegenwärtige Fensterinhalt als Hintergrund verwendet werden soll.
Werteliste: 'true', 'false'.
Standardwert: 'false'.

'colored': Setzt die Farbe aller gerade hinzugefügten Instanzen auf unterschiedlichen Farben. Der Wert dieses Parameters bestimmt die Anzahl der verwendeten Farben.
Werteliste: 3, 6, 12.
Standardwert: Alle 3D-Objektmodelle werden weiß dargestellt.

'object_index_persistence': Muss auf *'true'* gesetzt werden, wenn der Objektindex mit get_disp_object_model_3d_info abgefragt werden soll.
Werteliste: 'true', 'false'.
Standardwert: 'false'.

'depth_persistence': Muss auf *'true'* gesetzt werden, wenn die Tiefe mit get_disp_object_model_3d_info abgefragt werden soll.
Werteliste: 'true', 'false'.
Standardwert: 'false'.

'quality': Kann auf *'low'* gesetzt werden um schnelleres Rendering ohne Antialiasing zu erzielen.
Werteliste: 'low', 'high'.
Standardwert: 'high'.

'compatibility_mode_enable': Erzwingt die Nutzung des Fallbacks auf OpenGL 1.1.
Werteliste: 'true', 'false'.
Standardwert: 'false'.

Parameter

▷ **Scene3D** (input_control) ..... scene_3d ⇝ *handle*
Handle der 3D-Szene.

▷ **GenParamName** (input_control) ..... string ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'quality'
Werteliste: GenParamName ∈ {'alpha', 'attribute', 'color', 'colored', 'depth_persistence', 'disp_background', 'disp_lines', 'disp_pose', 'disp_normals', 'line_color', 'line_width', 'normal_color', 'object_index_persistence', 'point_size', 'quality', 'compatibility_mode_enable', 'color_attrib', 'color_attrib_start', 'color_attrib_end', 'red_channel_attrib', 'green_channel_attrib', 'blue_channel_attrib', 'rgb_channel_attrib_start', 'rgb_channel_attrib_end', 'lut', 'visible'}

▷ **GenParamValue** (input_control) ..... string ⇝ *string* / integer / real
Werte der generischen Parameter.
Defaultwert: 'high'
Wertevorschläge: GenParamValue ∈ {'true', 'false', 'high', 'low', 'coord_x', 'coord_y', 'coord_z', 'normal_x', 'normal_y', 'normal_z', 'red', 'green', 'blue', 'auto', 'faces', 'primitive', 'points', 'lines'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_scene_3d_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_scene_3d

Siehe auch

set_scene_3d_instance_param

Modul

3D Metrology

set_scene_3d_to_world_pose (: : Scene3D, ToWorldPose :)

Setzen der Pose einer 3D-Szene.

set_scene_3d_to_world_pose setzt die Pose der 3D-Szene Scene3D im Weltkoordinatensystem auf ToWorldPose.

Die Koordinatensysteme der 3D-Szene sind in der Dokumentation von create_scene_3d beschrieben.

Parameter

- ▷ **Scene3D** (input_control) ..... scene_3d ⇝ *handle*
  Handle der 3D-Szene.
- ▷ **ToWorldPose** (input_control) ..... pose ⇝ *real* / integer
  Neue Pose der 3D-Szene

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_scene_3d_to_world_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_scene_3d

Siehe auch

set_scene_3d_instance_pose

Modul

3D Metrology

11.2 Ausgabe

disp_arc (: : WindowHandle, CenterRow, CenterCol, Angle, BeginRow, BeginCol :)

Ausgeben von Bögen in ein Fenster.

disp_arc trägt einen oder mehrere Bögen, die durch den Schwerpunkt (CenterRow,CenterCol), den Winkel zwischen Anfang und Ende des Bogens (Angle in Bogenmaß) und den ersten Punkt (BeginRow,BeginCol) beschrieben werden, in das Ausgabefenster ein. Die Ausgabe des Bogens geschieht im Uhrzeigersinn. Die Parameter für die Ausgabe können - wie bei der Darstellung einer Region (disp_region) - mit den Operatoren set_color, set_gray, set_draw, etc., bestimmt werden. Es können mehrere Bögen bei einem Aufruf gezeichnet werden, indem jeweils Tupel übergeben werden. Für die Behandlung der Farben bei mehr als einem Bogen, siehe set_color.

Achtung

Der Schwerpunkt muss innerhalb des Fensters liegen. Der Bogen muss mindestens einen Radius von 2 Pixel haben.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **CenterRow** (input_control) . . . . . arc.center.y ⇝ *real* / integer
Zeilenindex des Schwerpunktes.
Defaultwert: 64
Wertevorschläge: `CenterRow` ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ `CenterRow` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **CenterCol** (input_control) . . . . . arc.center.x ⇝ *real* / integer
Spaltenindex des Schwerpunktes.
Defaultwert: 64
Wertevorschläge: `CenterCol` ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ `CenterCol` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Angle** (input_control) . . . . . arc.angle.rad ⇝ *real* / integer
Winkel zwischen Anfang und Ende des Bogens in Bogenmaß.
Defaultwert: 3.1415926
Wertevorschläge: `Angle` ∈ {0.0, 0.785398, 1.570796, 3.1415926, 6.283185}
Typischer Wertebereich: 0.0 ≤ `Angle` ≤ 6.283185 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: Angle > 0.0

▷ **BeginRow** (input_control) . . . . . arc.begin.y(-array) ⇝ *integer* / real
Zeilenindex des Anfangs des Bogens.
Defaultwert: 32
Wertevorschläge: `BeginRow` ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ `BeginRow` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **BeginCol** (input_control) . . . . . arc.begin.x(-array) ⇝ *integer* / real
Spaltenindex des Anfangs des Bogens.
Defaultwert: 32
Wertevorschläge: `BeginCol` ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ `BeginCol` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Beispiel

```
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
set_draw(WindowHandle,'fill')
set_color(WindowHandle,'white')
Row := 100
Column := 100
disp_arc(WindowHandle,Row,Column,3.14,Row+10,Column+10)
close_window(WindowHandle)
```

Ergebnis

`disp_arc` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi

Alternativen

disp_circle, disp_ellipse, disp_region, gen_circle, gen_ellipse

Siehe auch

open_window, set_color, set_draw, set_rgb, set_hsi

Modul

Foundation

disp_arrow (: : WindowHandle, Row1, Column1, Row2, Column2, Size :)

Ausgeben von Pfeilen in ein Fenster.

disp_arrow trägt einen oder mehrere Pfeile mit Anfangskoordinaten (Row1,Column1) und Endkoordinaten (Row2,Column2) in das Ausgabefenster ein. Die Pfeilspitzen befinden sich jeweils an den Endkoordinaten. Der Parameter Size bestimmt die Größe der Pfeilspitze. Besteht der Pfeil nur aus einem Punkt (Anfangspunkt = Endpunkt), wird nichts gezeichnet. Die Operatoren, die zur Steuerung der Regionenausgabe verwendet werden, legen auch hier die Ausgabeparameter fest (z.B. set_draw, set_color,set_line_width).

Es können mehrere Pfeile bei einem Aufruf gezeichnet werden, indem Tupel von Koordinaten übergeben werden. Für die Behandlung der Farben bei mehr als einem Pfeil, siehe set_color.

Achtung

Die Anfangs und Endpunkte der Pfeile müssen innerhalb des Fensters liegen.

Parameter

- **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- **Row1** (input_control) . . . . . line.begin.y(-array) ⇝ *real* / integer
  Zeilenindex des Anfangspunktes.
  Defaultwert: 10.0
  Wertevorschläge: Row1 ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich: 0.0 ≤ Row1 ≤ 511.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0
- **Column1** (input_control) . . . . . line.begin.x(-array) ⇝ *real* / integer
  Spaltenindex des Anfangspunktes.
  Defaultwert: 10.0
  Wertevorschläge: Column1 ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich: 0.0 ≤ Column1 ≤ 511.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0
- **Row2** (input_control) . . . . . line.end.y(-array) ⇝ *real* / integer
  Zeilenindex des Endpunktes.
  Defaultwert: 118.0
  Wertevorschläge: Row2 ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich: 0.0 ≤ Row2 ≤ 511.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0
- **Column2** (input_control) . . . . . line.end.x(-array) ⇝ *real* / integer
  Spaltenindex des Endpunktes.
  Defaultwert: 118.0
  Wertevorschläge: Column2 ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich: 0.0 ≤ Column2 ≤ 511.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0

▷ **Size** (input_control) ..... number ⇝ *real* / integer
Größe der Pfeilspitze.
Defaultwert: 1.0
Wertevorschläge: Size ∈ {1.0, 2.0, 3.0, 5.0}
Typischer Wertebereich: 0.0 ≤ Size ≤ 20.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 1.0
Restriktion: Size > 0.0

Beispiel

```
set_color(WindowHandle,['red','green'])
disp_arrow(WindowHandle,[10,10],[10,10],[118,110],[118,118],1.0)
```

Ergebnis

disp_arrow liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi

Alternativen

disp_line, gen_region_polygon, disp_region

Siehe auch

open_window, set_color, set_draw, set_line_width

Modul

Foundation

disp_channel (MultichannelImage : : WindowHandle, Channel :)

Ausgeben von mehrkanaligen Bildern.

disp_channel gibt ein Bild in das Ausgabefenster aus. Es können auch mehrere Bilder gleichzeitig ausgegeben werden. Hierbei werden diese nacheinander gezeichnet. Falls die Definitionsbereiche der Bilder sich überlappen, ist nur das zuletzt gezeichnete Bild sichtbar. Der Parameter Channel gibt die Nummer des Kanals an, der ausgegeben werden soll. Bei RGB-Bildern sind die drei Farbkanäle als Tupel anzugeben. Weitere Informationen sind bei disp_image zu finden.

Parameter

▷ **MultichannelImage** (input_object) ..... (multichannel-)image(-array) ⇝ *object* : byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Auszugebende Mehrkanalbilder.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Channel** (input_control) ..... integer(-array) ⇝ *integer*
Nummer des Kanals oder die Nummern der RGB-Kanäle.
Defaultwert: 1
Wertevorschläge: Channel ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Beispiel

```
read_image(Image,'patras')
count_channels (Image, Channels)
for I := 1 to Channels by 1
    disp_channel (Image, WindowHandle, I)
endfor
```

Ergebnis

Falls die übergebenen Bilder definierte Werte enthalten und ein gültiger Ausgabemodus eingestellt ist, liefert `disp_channel` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_rgb`, `set_lut`, `set_hsi`

Alternativen

`disp_image`, `disp_color`

Siehe auch

`open_window`, `reset_obj_db`, `set_lut`, `dump_window`

Modul

Foundation

disp_circle (: : WindowHandle, Row, Column, Radius :)

Ausgeben von Kreisen in ein Fenster.

`disp_circle` trägt einen oder mehrere Kreise, die durch den Schwerpunkt (`Row`, `Column`) und den `Radius` beschrieben werden, in das Ausgabefenster ein. Liegen die angegebenen Koordinaten ganz oder teilweise außerhalb der Fensterkoordinaten, dann wird der Kreis entsprechend beschnitten.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (`disp_region`) mit Operatoren wie `set_color`, `set_gray`, `set_draw`, etc. bestimmt werden.

Es können mehrere Kreise bei einem Aufruf gezeichnet werden, indem Tupel von Koordinaten übergeben werden. Für die Behandlung der Farben bei mehr als einem Kreis, siehe `set_color`.

Achtung

Der Schwerpunkt des Kreises muss innerhalb des Fensters liegen.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Row** (input_control) . . . . . circle.center.y(-array) ⇝ *real* / integer
Zeilenindex des Schwerpunktes.
Defaultwert: 64
Wertevorschläge: Row ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ Row ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Column** (input_control) . . . . . circle.center.x(-array) ⇝ *real* / integer
Spaltenindex des Schwerpunktes.
Defaultwert: 64
Wertevorschläge: Column ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ Column ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Radius** (input_control) . . . . . circle.radius(-array) ⇝ *real* / integer
Radius des Kreises.
Defaultwert: 64
Wertevorschläge: Radius ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ Radius ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Radius > 0.0

Beispiel

```
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
set_draw(WindowHandle,'fill')
set_color(WindowHandle,'white')
repeat
  get_mbutton(WindowHandle,Row,Column,Button)
  disp_circle(WindowHandle,Row,Column,(Row + Column) % 50)
until(Button == 1)
close_window(WindowHandle)
```

Ergebnis

disp_circle liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi

Alternativen

disp_ellipse, disp_region, gen_circle, gen_ellipse

Siehe auch

open_window, set_color, set_draw, set_rgb, set_hsi

Modul

Foundation

disp_color (ColorImage : : WindowHandle :)

Ausgeben eines RGB-Bildes.

disp_color trägt die drei Kanäle eines Farbbildes in das Ausgabefenster ein. Der erste Kanal ist der Rotkanal, der zweite der Grünkanal und der dritte der Blaukanal. disp_color kann mit disp_channel simuliert werden.

Achtung

Wegen der Beschränkung der Anzahl von Bildschirmfarben ist die Darstellung von Farbbildern i.a. nicht originalgetreu.

Parameter

▷ **ColorImage** (input_object) ..... multichannel-image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Auszugebendes Farbbild.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

Beispiel

```
/* disp_color(ColorImage) is identical to: */
Herror my_disp_color(Hobject ColorImage, Htuple *WindowHandle) {
  Htuple  Tupel;
  create_tuple(&Tupel,3);
  set_i(Tupel,1,0);
  set_i(Tupel,2,1);
  set_i(Tupel,3,2);
  T_disp_channel(ColorImage,*WindowHandle,Tupel);
  destroy_tuple(Tupel);
}
```

Ergebnis

Falls das übergebene Bild definierte Werte enthält und ein gültiger Ausgabemodus eingestellt ist, dann liefert `disp_color` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_rgb`, `set_lut`, `set_hsi`

Alternativen

`disp_channel`, `disp_obj`

Siehe auch

`disp_image`, `open_window`, `reset_obj_db`, `set_lut`, `dump_window`

Modul

Foundation

disp_cross (: : WindowHandle, Row, Column, Size, Angle :)

Zeichnen von Kreuzen in ein Fenster.

`disp_cross` trägt ein oder mehrere Kreuze mit Mittelpunkt (`Row`,`Column`), Schenkellänge `Size` und Orientierung `Angle` in das Ausgabefenster ein. Bei der Darstellung von mehr als einem Kreuz müssen die Koordinaten dabei jeweils in Form eines Tupels übergeben werden.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (`disp_region`) mit den Operatoren `set_color`, `set_gray`, `set_draw`, `set_line_width`, etc. bestimmt werden. Für die Behandlung der Farben bei mehr als einem Kreuz siehe `set_color`.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Row** (input_control) ..... coordinates.y(-array) ⇝ *real*
Zeilenkoordinate des Mittelpunktes.
Defaultwert: 32.0
Wertevorschläge: `Row` ∈ {0.0, 64.0, 128.0, 256.0, 511.0}

▷ **Column** (input_control) ..... coordinates.x(-array) ⇝ *real*
Spaltenkoordinate des Mittelpunktes.
Defaultwert: 32.0
Wertevorschläge: Column ∈ {0.0, 64.0, 128.0, 256.0, 511.0}

▷ **Size** (input_control) ..... number ⇝ *real*
Länge der Schenkel.
Defaultwert: 6.0
Wertevorschläge: Size ∈ {4.0, 6.0, 8.0, 10.0}
Typischer Wertebereich: 0.0 ≤ Size

▷ **Angle** (input_control) ..... angle.rad ⇝ *real*
Orientierung.
Defaultwert: 0.0
Wertevorschläge: Angle ∈ {0.0, 0.785398163397448309615660845819 88}

Ergebnis

disp_cross liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_rgb, set_lut, set_hsi, set_draw, set_color, set_colored, set_line_width

Alternativen

disp_arrow, disp_rectangle1, disp_rectangle2, disp_circle

Siehe auch

open_window, set_color, set_rgb, set_hsi, set_insert, set_line_width

Modul

Foundation

disp_ellipse (: : WindowHandle, CenterRow, CenterCol, Phi, Radius1, Radius2 :)

Ausgeben von Ellipsen.

disp_ellipse trägt eine oder mehrere Ellipsen mit dem Schwerpunkt (CenterRow,CenterCol), der Orientierung Phi (in Bogenmaß) und den Halbradien Radius1 und Radius2 in das Ausgabefenster ein.

Es kann mehr als eine Ellipse ausgegeben werden, indem Tupel von Punkten, Winkeln und Halbradien übergeben werden. Die Parameter für die Ausgabe können wie bei der Darstellung von Regionen (disp_region) mit den Operatoren set_color, set_draw, set_line_width, etc. bestimmt werden. Für die Behandlung der Farben bei mehr als einer Ellipse, siehe set_color.

Achtung

Der Schwerpunkt muss innerhalb der Fensterkoordinaten liegen.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **CenterRow** (input_control) ..... ellipse.center.y(-array) ⇝ *integer*
Zeilenindex des Schwerpunktes.
Defaultwert: 64
Wertevorschläge: CenterRow ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ CenterRow ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **CenterCol** (input_control) . . . . . ellipse.center.x(-array) ⇝ *integer*
Spaltenindex des Schwerpunktes.
Defaultwert: 64
Wertevorschläge: CenterCol ∈ {0, 64, 128, 256}
Typischer Wertebereich: 0 ≤ CenterCol ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Phi** (input_control) . . . . . ellipse.angle.rad(-array) ⇝ *real* / integer
Orientierung der Ellipse in Bogenmaß.
Defaultwert: 0.0
Wertevorschläge: Phi ∈ {0.0, 0.785398, 1.570796, 3.1415926, 6.283185}
Typischer Wertebereich: 0.0 ≤ Phi ≤ 6.283185 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1

▷ **Radius1** (input_control) . . . . . ellipse.radius1(-array) ⇝ *real* / integer
Lange Halbachse der Ellipse.
Defaultwert: 24.0
Wertevorschläge: Radius1 ∈ {0.0, 64.0, 128.0, 256.0}
Typischer Wertebereich: 0.0 ≤ Radius1 ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Radius2** (input_control) . . . . . ellipse.radius2(-array) ⇝ *real* / integer
Kurze Halbachse der Ellipse.
Defaultwert: 14.0
Wertevorschläge: Radius2 ∈ {0.0, 64.0, 128.0, 256.0}
Typischer Wertebereich: 0.0 ≤ Radius2 ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

Beispiel

```
set_color(WindowHandle,'red')
draw_region(MyRegion,WindowHandle)
elliptic_axis(MyRegion,Ra,Rb,Phi)
area_center(MyRegion,_,Row,Column)
disp_ellipse(WindowHandle,Row,Column,Phi,Ra,Rb)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert disp_ellipse den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi, elliptic_axis, area_center

Alternativen

disp_circle, disp_region, gen_ellipse, gen_circle

Siehe auch

open_window, set_color, set_rgb, set_hsi, set_draw, set_line_width

Modul

Foundation

disp_image (Image : : WindowHandle :)

Ausgeben von Grauwertdaten.

disp_image gibt die Grauwerte von einem Bild auf dem Ausgabefenster aus. Es werden dabei diejenigen Grauwerte verwendet, die innerhalb des Definitionsbereiches (set_comprise(::WindowHandle,'object':)) oder des gesamten Bildes (set_comprise(::WindowHandle,'image':)) liegen. Voreingestellt ist, dass nur die definierten Grauwerte verwendet werden.

Bei der Ausgabe von Graubildern wird die Anzahl der Graustufen i.a. reduziert. Dies ist durch die Tatsache bedingt, dass noch Farben für die Darstellung von Grafiken (z.B. set_color) und den Window-Manager reserviert werden müssen. Zudem stehen abhängig von den auf der Ausgabemaschine verfügbaren Bitplanes häufig weniger als 256 Farben (acht Bitplanes) zur Verfügung. Die Anzahl der tatsächlich für die Ausgabe von Graustufen reservierten „Farben“ kann mittels get_system abgefragt und vor dem Öffnen des ersten Fensters mittels set_system auch verändert werden. Für 8 Bitplanes sind beispielsweise 200 reale Graustufen voreingestellt.

Die Reduktion der Anzahl der Graustufen ist unproblematisch, solange nur Grauwertinformation ausgegeben wird, da der Mensch 256 verschiedenen Graustufen ohnehin nicht unterscheiden kann. Sind jedoch bestimmte Grauwerte für die Speicherung von Regioneninformation verwendet worden (was nicht der Vorgehensweise von HALCON entspricht), so kann dies bei der Darstellung zu Verwechslungen führen, da somit unterschiedliche Zahlenwerte mit dem gleichen Grauwert auf dem Bildschirm abgebildet werden. Sollten derartige Bilder vorliegen, empfiehlt es sich, den Operator label_to_region zu verwenden, um das Bild mit den Markierungen in HALCON-Objekte umzuwandeln.

Werden Bilder vom Typ ’int2’, ’int4’, ’int8’, ’real’ oder ’complex’ ausgegeben, so wird zunächst der kleinste und der größte Grauwert bestimmt. Für Bilder vom Typ ’complex’ basiert diese Berechnung auf dem entsprechenden Leistungsspektrum. Danach werden die Bildpunkte auf die zur Verfügung stehende Anzahl von Graustufen (abhängig von der Ausgabemaschine, ca. 200) skaliert. Hierbei kann es vorkommen, dass einige Pixelwerte extrem andere Werte als alle übrigen haben. Dies führt dazu, dass entweder ein ganz weißes oder schwarzes Bild auf dem Fenster erscheint. Um festzustellen, ob es sich um ein binäres Bild handelt, kann man z.B. min_max_gray verwenden. Gegebenenfalls kann das Bild vor der Ausgabe mit scale_image und convert_image_type in der gewünschten Weise angepasst werden.

Achtung

Wurde bei set_paint ein falscher Ausgabemodus gesetzt, dann erfolgt die Fehlermeldung erst bei disp_image.

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Auszugebendes Grauwertbild.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*

Fenster-Handle.

Beispiel

```
* Output of a gray image:
read_image(Image,'monkey')
disp_image(Image,WindowHandle)
```

Ergebnis

Falls das übergebene Bild definierte Werte enthält und ein gültiger Ausgabemodus eingestellt ist, dann liefert disp_image den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_rgb, set_lut, set_hsi, scale_image, convert_image_type, min_max_gray

Alternativen

disp_obj, disp_color

Siehe auch

open_window, reset_obj_db, set_paint, set_lut, paint_gray, scale_image, convert_image_type, dump_window

Modul

Foundation

disp_line (: : WindowHandle, Row1, Column1, Row2, Column2 :)

Zeichnen von Linien in ein Fenster.

disp_line trägt eine oder mehrere Linien mit Anfangskoordinaten (Row1,Column1) und Endkoordinaten (Row2,Column2) in das Ausgabefenster ein. Bei der Darstellung von mehr als einer Linie müssen die Koordinaten dabei jeweils in Form eines Tupels übergeben werden. Für die Behandlung der Farben bei mehr als einer Linie, siehe set_color.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (disp_region) mit den Operatoren set_color, set_gray, set_draw, set_line_width, etc. bestimmt werden.

Achtung

Die Anfangs- und Endpunkte der Linien müssen innerhalb des Fensters liegen.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Row1** (input_control) ..... line.begin.y(-array) ⇝ *real*
Zeilenindex des Anfangspunktes.
Defaultwert: 32.0
Wertevorschläge: Row1 ∈ {0.0, 64.0, 128.0, 256.0, 511.0}
Typischer Wertebereich: 0 ≤ Row1 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Column1** (input_control) ..... line.begin.x(-array) ⇝ *real*
Spaltenindex des Anfangspunktes.
Defaultwert: 32.0
Wertevorschläge: Column1 ∈ {0.0, 64.0, 128.0, 256.0, 511.0}
Typischer Wertebereich: 0 ≤ Column1 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Row2** (input_control) ..... line.end.y(-array) ⇝ *real*
Zeilenindex des Endpunktes.
Defaultwert: 64.0
Wertevorschläge: Row2 ∈ {0.0, 64.0, 128.0, 256.0, 511.0}
Typischer Wertebereich: 0 ≤ Row2 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Column2** (input_control) ..... line.end.x(-array) ⇝ *real*
Spaltenindex des Endpunktes.
Defaultwert: 64.0
Wertevorschläge: Column2 ∈ {0.0, 64.0, 128.0, 256.0, 511.0}
Typischer Wertebereich: 0 ≤ Column2 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

Beispiel

```
* Display contour of a rectangle
disp_line(WindowHandle,Row1,Column1,Row1,Column2)
disp_line(WindowHandle,Row1,Column2,Row2,Column2)
disp_line(WindowHandle,Row2,Column2,Row2,Column1)
disp_line(WindowHandle,Row2,Column1,Row1,Column1)
```

Ergebnis

`disp_line` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `set_draw`, `set_color`, `set_colored`, `set_line_width`

Alternativen

`disp_arrow`, `disp_rectangle1`, `disp_rectangle2`, `disp_region`, `gen_region_polygon`, `gen_region_points`

Siehe auch

`open_window`, `set_color`, `set_rgb`, `set_hsi`, `set_insert`, `set_line_width`

Modul

Foundation

disp_obj (Object : : WindowHandle :)

Ausgeben von Bildobjekten (Bilder, Regionen, XLD).

`disp_obj` gibt die übergebenen Objekte entsprechend ihrer Art (Bilder, Regionen, XLD) aus. `disp_obj` entspricht `disp_image` bei einem einkanaligen Bild, `disp_color` bei einem dreikanaligen Bild, `disp_region` bei Regionen und `disp_xld` bei XLD.

Parameter

▷ **Object** (input_object) ..... object(-array) ⇝ *object*
Auszugebendes Bildobjekt.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

Beispiel

```
* Output of a gray image:
read_image(Image1,'monkey')
disp_obj(Image1,WindowHandle)
threshold(Image,Region,0,128)
disp_obj(Region,WindowHandle)
```

Ergebnis

Falls das übergebene Objekt definierte Werte enthält und ein gültiger Ausgabemodus eingestellt ist, dann liefert `disp_obj` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `scale_image`, `convert_image_type`, `min_max_gray`

Alternativen

`disp_color`, `disp_image`, `disp_xld`, `disp_region`

Siehe auch

`open_window`, `reset_obj_db`, `set_paint`, `set_lut`, `paint_gray`, `scale_image`, `convert_image_type`, `dump_window`

Modul

Foundation

`disp_object_model_3d` `( : : WindowHandle, ObjectModel3D, CamParam, Pose, GenParamName, GenParamValue : )`

Ausgeben von 3D Objektmodellen.

`disp_object_model_3d` zeigt die 3D Objektmodelle `ObjectModel3D` im Fenster `WindowHandle` an. Die Szene wird über die Kameraparameter `CamParam` und die Posen (in `Pose`) der 3D Objektmodelle definiert. Wird ein leeres Tupel für `CamParam` übergeben, verwendet `disp_object_model_3d` Default-Kameraparameter, die auf der Fenstergröße beruhen. `Pose` kann entweder eine Pose enthalten, die für alle 3D Objektmodelle gilt, oder eine individuelle Pose für jedes 3D Objektmodell. Wird ein leeres Tupel für `Pose` übergeben, schätzt `disp_object_model_3d` diese Parameter so, dass alle Objekte sichtbar sind. Die Posen werden in der Form ${}^{ccs}\mathbf{P}_{mcs}$, erwartet. Dabei steht *ccs* für das Kamerakoordinaten und *mcs* für das Modellkoordinatensystem (ein 3D Weltkoordinatensystem), siehe auch Transformationen / Posen und „`Solution Guide III-C - 3D Vision`".

3D Objektmodelle, die Dreiecke oder Polygone beinhalten, werden als geschlossene Körper dargestellt. 3D Objektmodelle, die nur Koordinaten enthalten, werden dagegen als Punktwolken visualisiert. 3D-Primitive (wie sie zum Beispiel von `fit_primitives_object_model_3d` erzeugt werden), werden auch als geschlossene Körper dargestellt. Mit dem generischen Parameter *'attribute'* kann explizit die Art der Darstellung ausgewählt werden.

Um 3D Objektmodelle in ein Bild zu rendern, kann der Operator `render_object_model_3d` benutzt werden.

Mit `GenParamName` und `GenParamValue` können weitere Eigenschaften der Szene konfiguriert werden. Die folgenden Werte beeinflussen die gesamte Szene:

'disp_background': Flag, ob der gegenwärtige Fensterinhalt als Hintergrund verwendet werden soll.
Werteliste: 'true', 'false'.
Standardwert: 'false'.

'light_position': Position der Lichtquelle. Dieser Parameter muss als String mit vier Fließkommawerten (getrennt durch Leerzeichen) übergeben werden. Ist der vierte Wert 0.0, wird eine gerichtete Lichtquelle verwendet, wobei die ersten drei Werte die Richtung bestimmen. Ansonsten wird eine Punktlichtquelle verwendet, deren Position durch die ersten drei Werte bestimmt ist.
Standardwert: '-100.0 -100.0 0.0 1.0'.

'light_ambient': Ambienter Anteil der Lichtquelle. Dieser Parameter muss als String mit drei Fließkommawerten (getrennt durch Leerzeichen) übergeben werden.
Standardwert: '0.2 0.2 0.2'.

'light_diffuse': Diffuser Anteil der Lichtquelle. Dieser Parameter muss als String mit drei Fließkommawerten (getrennt durch Leerzeichen) übergeben werden.
Standardwert: '0.8 0.8 0.8'.

'colored': Darstellung von 3D Objektmodellen mit unterschiedlichen Farben. Der Wert dieses Parameters bestimmt die Anzahl der verwendeten Farben.

Werteliste: 3, *6*, *12*.
Standardwert: alle 3D Objektmodelle werden weiß dargestellt.

'object_index_persistence': Muss auf *'true'* gesetzt werden, wenn der Objektindex mit `get_disp_object_model_3d_info` abgefragt werden soll.
Werteliste: 'true', *'false'*.
Standardwert: 'false'.

'depth_persistence': Muss auf *'true'* gesetzt werden, wenn die Tiefe mit `get_disp_object_model_3d_info` abgefragt werden soll.
Werteliste: 'true', *'false'*.
Standardwert: 'false'.

'quality': Kann auf *'low'* gesetzt werden um schnelleres Rendering ohne Antialiasing zu erzielen.
Werteliste: 'low', *'high'*.
Standardwert: 'high'.

Die folgenden Parameter können entweder für alle 3D Objektmodelle der Szene oder für ein einzelnes 3D Objektmodell gesetzt werden (indem der Index des 3D Objektmodells an den Parameternamen angehängt wird, z.B. *'color_0'*).

'attribute' Wählt explizit die Art und Weise aus, wie ein 3D Objektmodell visualisiert wird.
Werteliste: 'auto', *'faces'*, *'primitive'*, *'points'*, *'lines'*.
Standardwert: 'auto'.

'color': Farbe des 3D Objektmodells. Die verfügbaren Farben können mit dem Operator `query_color` erfragt werden. Darüberhinaus kann die Farbe auch über ein RGB-Tripel in der Form *'#rrggbb'* gesetzt werden, wobei 'rr', 'gg' und 'bb' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind.
Wertevorschläge: 'red', *'green'*.
Standardwert: 'white'.

'alpha': Transluzenz des 3D Objektmodells. Das Darstellen von 3D Objektmodellen mit Transluzenz kleiner 1.0 kann zu signifikant höherer Laufzeit führen.
Mögliche Werte: Fließkommawerte zwischen *0.0* (transparent) und *1.0* (opak).
Standardwert: 1.0.

'disp_pose': Flag, ob die Pose des 3D Objektmodells visualisiert werden soll.
Werteliste: 'true', *'false'*.
Standardwert: 'false'.

'disp_lines': Flag, ob die Konturen der Polygone des 3D Objektmodells angezeigt werden sollen.
Werteliste: 'true', *'false'*
Standardwert: 'false'.

'disp_normals': Flag, ob die Oberflächennormalen der Polygone des 3D Objektmodells angezeigt werden sollen.
Werteliste: 'true', *'false'*.
Standardwert: 'false'.

'line_color': Farbe der Konturen, falls *'disp_lines'* auf *'true'* gesetzt ist. Die verfügbaren Farben können mit dem Operator `query_color` erfragt werden. Darüberhinaus kann die Farbe auch über ein RGB-Tripel in der Form *'#rrggbb'* gesetzt werden, wobei 'rr', 'gg' und 'bb' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind.
Wertevorschläge: 'red', *'green'*.
Standardwert: Der Wert von *'color'*.

'line_width': Setzt die Breite von Linien in Pixel.
Standardwert: 1.0.

'normal_color': Farbe der Oberflächennormalen, falls *'disp_normals'* auf *'true'* gesetzt ist. Die verfügbaren Farben können mit dem Operator `query_color` erfragt werden. Darüberhinaus kann die Farbe auch über ein RGB-Tripel in der Form *'#rrggbb'* gesetzt werden, wobei 'rr', 'gg' und 'bb' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind.
Wertevorschläge: 'red', *'green'*.
Standardwert: Der Wert von *'color'*.

'point_size': Setzt den Durchmesser der Punkte.

Standardwert: 3.5.

'lut': Setzt die LUT, die die Werte des mit *'color_attrib'* gesetzten Attributes in Farben umwandelt. In der Dokumentation von `set_lut` befindet sich eine Liste der verfügbaren LUTs. Ist *'lut'* auf etwas anderes als *'default'* gesetzt, wird *'color'* ignoriert.

Standardwert: 'default'.

'color_attrib': Name eines Punktattributes, das für die Falschfarbendarstellung verwendet wird.

Falls ein Attribut gesetzt ist, wird die Farbe eines dargestellten 3D-Punktes durch den Attributwert des Punktes und die aktuell gesetzte LUT bestimmt (siehe *'lut'*). Auf diese Weise ist es möglich, Attribute in Falschfarben darzustellen.

Beispiel: Wenn *'color_attrib'* auf *'coord_z'* und *'lut'* auf *'color1'* gesetzt ist, werden die Z-Koordinaten von rot nach blau farbkodiert.

Falls *'lut'* auf *'default'* gesetzt ist, werden die Attributwerte verwendet, um die mit *'color'* gesetzte Farbe zu skalieren.

Falls *'lut'* auf etwas anderes gesetzt, werden die Attributwerte intern auf den Bereich [0,255] skaliert und als Eingabe für die LUT-Funktion verwendet.

Die Abbildung wird außerdem von den Parametern *'color_attrib_start'* und *'color_attrib_end'* bestimmt (siehe unten).

Falls Flächen dargestellt werden, wird ihre Farbe zwischen den Eckpunktfarben interpoliert.

Mögliche Werte: 'none', *'&distance'*, *'coord_x'*, *'coord_y'*, *'coord_z'*, benutzerdefinierte Punktattribute, oder beliebige andere verfügbare Punktattribute.

Standardwert: 'none'.

'color_attrib_start',*'color_attrib_end'*: Der maßgebliche Wertebereich für das mit *'color_attrib'* eingestellte Attribut.

Die Attributwerte zwischen *'color_attrib_start'* und *'color_attrib_end'* werden auf den Anfang und das Ende der ausgewählten LUT skaliert. Attributwerte außerhalb des Bereiches werden beschnitten. Damit ist es möglich, einen festen Farbbereich darzustellen, der nicht durch Ausreißer verzerrt wird.

Bei *'auto'* wird der kleinste Attributwert auf den Anfang der LUT abgebildet, der größte Wert auf das Ende, es sei denn, *'color_attrib'* ist *'normal_x'*, *'normal_y'* oder *'normal_z'*. In diesem Fall, werden Start und Ende automatisch auf -1 und 1 gesetzt.

Der Startwert kann höher als der Endwert sein. In dem Fall wird die LUT umgedreht.

Wertevorschläge: 0, *0.1*, *1*, *100*, *255*.

Standardwert: 'auto'.

'red_channel_attrib', *'green_channel_attrib'*, *'blue_channel_attrib'*: Name eines Punktattributes, das für den Rot-, Grün-, oder Blaukanal verwendet wird.

Das ist besonders nützlich, wenn drei zusammengehörige Attribute verwendet werden, z.B. RGB-Farben oder Normalenvektoren. Auf diese Weise können Punkte mit farbigen Texturen, z.B. RGB-Sensor-Daten, dargestellt oder Punktnormalen in Falschfarben kodiert werden.

Um ein einzelnes Attribut in Falschfarben darzustellen, verwenden Sie bitte *'color_attrib'* (siehe oben).

Per Default wird angenommen, dass die Attributwerte zwischen 0 und 255 liegen. Falls die Attribute andere Wertebereiche haben, müssen zusätzlich die Parameter *'rgb_channel_attrib_start'* und *'rgb_channel_attrib_end'* gesetzt werden (siehe unten).

Falls nur 1 oder 2 Kanäle gesetzt sind, verwenden die übrigen Kanäle den RGB-Wert der mit *'color'* eingestellten Farbe.

Falls Flächen dargestellt werden, wird ihre Farbe zwischen den Eckpunktfarben interpoliert.

Mögliche Werte: 'none', *'&red'*, *'&green'*, *'&blue'*, *'normal_x'*, *'normal_y'*, *'normal_z'*, benutzerdefinierte Punktattribute, oder beliebige andere verfügbare Punktattribute.

Standardwert: 'none'.

'rgb_channel_attrib_start', *'rgb_channel_attrib_end'*: Der maßgebliche Wertebereich für die mit *'red_channel_attrib'*, *'green_channel_attrib'* und *'blue_channel_attrib'* eingestellten Attribute.

Diese Parameter definieren den Wertebereich, der auf die gesamten RGB-Kanäle skaliert wird. Das ist nützlich, wenn die Werte der Eingabeattribute nicht im Intervall [0,255] liegen.

Bei *'auto'* wird der kleinste Attributwert auf 0 abgebildet, der größte Wert auf 255, es sei denn, das Attribut ist *'normal_x'*, *'normal_y'* oder *'normal_z'*. In diesem Fall, werden Start und Ende automatisch auf -1 und 1 gesetzt.

Der Startwert kann höher als der Endwert sein. In dem Fall werden die RGB-Farben invertiert.

Der Bereich kann für jeden Kanal einzeln eingestellt werden, indem im Parameternamen das *'rgb'* durch den jeweiligen Kanal ersetzt wird, z.B. *'green_channel_attrib_start'*.

Wertevorschläge: 'auto', *0*, *0.1*, *1*, *100*, *255*.

Standardwerte: 0, *255*.

Alle generischen Parameter werden von links nach rechts ausgewertet. Um beispielsweise die Farbe des ersten Objekts auf rot und die Farbe aller anderen Objekte auf grün zu setzen, muss `GenParamName` auf *['color','color_0']* und `GenParamValue` auf *['green','red']* gesetzt werden.

`disp_object_model_3d` benötigt OpenGL 2.1, GLSL 1.2 und die OpenGL Extensions GL_EXT_framebuffer_object und GL_EXT_framebuffer_blit. Andernfalls wird automatisch der Kompatibilitätsmodus mit geringeren Anforderungen an OpenGL benutzt.

Bei Grafikkarten mit geringem Speicher können bei großer Auflösung des Renderfensters folgende Fehlermeldungen auftreten:

Low-Level Fehler: 'Incomplete attachment'

Unbehandelte Ausnahme: 'Benötigtes Framebuffer Objekt wird nicht unterstützt'

Lösungen:

Um die Anforderungen an den Speicher der Grafikkarte zu verringern, kann der Parameter *'quality'* auf *'low'* gesetzt oder der Kompatibilitätsmodus verwendet werden.

Die Systemvariable (siehe `set_system`) *'opengl_compatibility_mode_enable'* kann auf *'true'* gesetzt werden, um den Kompatibilitätsmodus permanent einzuschalten. Dieser Modus benötigt OpenGL 1.1.

Im Kompabilitätsmodus werden die Parameter *'object_index_persistence'*, *'depth_persistence'* und *'quality'* nicht unterstützt.

Auf Linux Remote-Desktop wird *'disp_background'* nicht unterstützt.

Achtung

Kameras mit hyperzentrischen Objektiven werden nicht unterstützt. Falls `CamParam` Verzeichnungskoeffizienten ungleich null enthält ist bei der Projektion großer Dreiecke oder Polygone (oder 3D-Primitive) darauf zu achten, dass die Verzeichnung nur auf die Punkte des Objektmodells angewandt werden. In der Projektion werden diese wiederum mit geraden Linien verbunden. Für eine gute Approximation der verzeichneten Linien sollte eine Triangulierung mit ausreichend kleinen Dreiecken verwendet werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fensteridentifikator.
- ▷ **`ObjectModel3D`** (input_control) ..... object_model_3d(-array) ⇝ *handle*
  Handle der 3D Objektmodelle.
- ▷ **`CamParam`** (input_control) ..... campar ⇝ *real* / integer / string
  Kameraparameter.
  Defaultwert: []
- ▷ **`Pose`** (input_control) ..... pose(-array) ⇝ *real* / integer
  3D Posen der Objekte im Kamerakoordinatensystem.
  Defaultwert: []
- ▷ **`GenParamName`** (input_control) ..... string-array ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'alpha', 'attribute', 'color', 'colored', 'depth_persistence', 'disp_background', 'disp_lines', 'disp_pose', 'disp_normals', 'light_position', 'light_ambient', 'light_diffuse', 'line_color', 'line_width', 'normal_color', 'object_index_persistence', 'point_size', 'quality', 'compatibility_mode_enable', 'color_attrib', 'color_attrib_start', 'color_attrib_end', 'red_channel_attrib', 'green_channel_attrib', 'blue_channel_attrib', 'rgb_channel_attrib_start', 'rgb_channel_attrib_end', 'lut'}
- ▷ **`GenParamValue`** (input_control) ..... string-array ⇝ *string* / integer / real
  Werte der generischen Parameter.
  Defaultwert: []
  Werteliste: `GenParamValue` ∈ {'true', 'false', 'coord_x', 'coord_y', 'coord_z', 'normal_x', 'normal_y', 'normal_z', 'red', 'green', 'blue', 'auto', 'faces', 'primitive', 'points', 'lines'}

Ergebnis

`disp_object_model_3d` liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Sollte das Rendering den verfügbaren Speicher der Grafikkarte überschreiten, wird die Fehlermeldung 5188 zurückgegeben.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_surface_model`, `fit_primitives_object_model_3d`, `read_object_model_3d`, `segment_object_model_3d`

Nachfolger

`get_disp_object_model_3d_info`

Siehe auch

`render_object_model_3d`, `project_object_model_3d`, `project_shape_model_3d`, `object_model_3d_to_xyz`

Modul

3D Metrology

disp_polygon (: : WindowHandle, Row, Column :)

Zeichnen eines Polygonzuges.

`disp_polygon` trägt ein Polygon mit den Zeilenkoordinaten `Row` und Spaltenkoordinaten `Column` in das Ausgabefenster ein. Die Parameter `Row` und `Column` werden jeweils in Form von Tupeln übergeben. Zwischen den angegebenen Punkten werden Geraden gezeichnet. Der Anfangs- und Endpunkt des Polygons wird nicht verbunden.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (`disp_region`) mit Operatoren wie `set_color`, `set_rgb`, etc. bestimmt werden.

Achtung

Die angegebenen Koordinaten müssen innerhalb des Fensters liegen.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Row** (input_control) . . . . . polygon.y-array ⇝ *integer* / real
Zeilenindex
Defaultwert: [16,80,80]
Wertevorschläge: `Row` ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich: 0 ≤ `Row` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Column** (input_control) . . . . . polygon.x-array ⇝ *integer* / real
Spaltenindex
Defaultwert: [48,16,80]
Wertevorschläge: `Column` ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich: 0 ≤ `Column` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

Beispiel

```
/* display a rectangle */

disp_rectangle1_margin1(Htuple WindowHandle,
                        Hlong Row1, long Column1,
                        Hlong Row2, long Column2)
{
  Htuple Row, Col;
  create_tuple(&Row,4);
  create_tuple(&Col,4);

  set_i(Row,Row1,0);
  set_i(Col,Column1,0);

  set_i(Row,Row1,1);
  set_i(Col,Column2,1);

  set_i(Row,Row2,2);
  set_i(Col,Column2,2);

  set_i(Row,Row2,3);
  set_i(Col,Column1,3);

  set_i(Row,Row1,4);
  set_i(Col,Column1,4);

  T_disp_polygon(WindowHandle,Row,Col);

}
```

Ergebnis

`disp_polygon` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `set_draw`, `set_color`, `set_colored`, `set_line_width`

Alternativen

`disp_line`, `gen_region_polygon`, `disp_region`

Siehe auch

`open_window`, `set_color`, `set_rgb`, `set_hsi`, `set_insert`, `set_line_width`

Modul

Foundation

`disp_rectangle1` `( : : WindowHandle, Row1, Column1, Row2, Column2 : )`

Ausgeben von achsenparallelen Rechtecken.

`disp_rectangle1` trägt ein oder mehrere Rechtecke, die jeweils durch die linke obere Ecke (`Row1`,`Column1`) und die rechte untere Ecke (`Row2`,`Column2`) beschrieben werden, in das Ausgabefenster ein. Liegen die angegebenen Koordinaten ganz oder teilweise außerhalb der Fensterkoordinaten, dann wird das Rechteck entsprechend beschnitten. Sollen mehrere Rechtecke dargestellt werden, sind die Koordinaten als Tupel zu übergeben.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (disp_region) mit den Operatoren set_color, set_gray, set_draw, set_line_width, etc. bestimmt werden.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Row1** (input_control) . . . . . rectangle.origin.y(-array) ⇝ *real* / integer
Zeilenindex der linken oberen Ecke.
Defaultwert: 16
Wertevorschläge: Row1 ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich: 0 ≤ Row1 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Column1** (input_control) . . . . . rectangle.origin.x(-array) ⇝ *real* / integer
Spaltenindex der linken oberen Ecke.
Defaultwert: 16
Wertevorschläge: Column1 ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich: 0 ≤ Column1 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Row2** (input_control) . . . . . rectangle.corner.y(-array) ⇝ *real* / integer
Zeilenindex der rechten unteren Ecke.
Defaultwert: 48
Wertevorschläge: Row2 ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich: 0 ≤ Row2 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Row2 >= Row1

▷ **Column2** (input_control) . . . . . rectangle.corner.x(-array) ⇝ *real* / integer
Spaltenindex der rechten unteren Ecke.
Defaultwert: 80
Wertevorschläge: Column2 ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich: 0 ≤ Column2 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Column2 >= Column1

Beispiel

```
set_color(WindowHandle,'green')
draw_region(MyRegion,WindowHandle)
smallest_rectangle1(MyRegion,R1,C1,R2,C2)
disp_rectangle1(WindowHandle,R1,C1,R2,C2)
```

Ergebnis

disp_rectangle1 liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_rgb, set_lut, set_hsi, set_draw, set_color, set_colored, set_line_width

Alternativen

disp_rectangle2, gen_rectangle1, disp_region, disp_line, set_shape

Siehe auch

`open_window`, `set_color`, `set_draw`, `set_line_width`

Modul

Foundation

disp_rectangle2 (: : WindowHandle, CenterRow, CenterCol, Phi, Length1, Length2 :)

Ausgeben von beliebig orientierten Rechtecken.

`disp_rectangle2` trägt ein oder mehrere Rechtecke mit dem Schwerpunkt (`CenterRow`,`CenterCol`), der Orientierung `Phi` (in Bogenmaß) und den halben Kantenlängen `Length1` und `Length2` in das Ausgabefenster ein. Es kann mehr als ein Rechteck gezeichnet werden, indem jeweils Tupel von Koordinaten übergeben werden. Für die Behandlung der Farben bei mehr als einem Rechteck, siehe `set_color`.

Die Parameter für die Ausgabe können wie bei der Darstellung von Regionen (`disp_region`) mit den Operatoren `set_color`, `set_draw`, `set_line_width`, etc. bestimmt werden.

Achtung

Der Schwerpunkt muss innerhalb der Fensterkoordinaten liegen.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **CenterRow** (input_control) . . . . . rectangle2.center.y(-array) ⇝ *real* / integer
  Zeilenindex des Schwerpunktes.
  Defaultwert: 48
  Wertevorschläge: `CenterRow` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich: 0 ≤ `CenterRow` ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
- ▷ **CenterCol** (input_control) . . . . . rectangle2.center.x(-array) ⇝ *real* / integer
  Spaltenindex des Schwerpunktes.
  Defaultwert: 64
  Wertevorschläge: `CenterCol` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich: 0 ≤ `CenterCol` ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
- ▷ **Phi** (input_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real* / integer
  Orientierung des Rechtecks in Bogenmaß.
  Defaultwert: 0.0
  Wertevorschläge: `Phi` ∈ {0.0, 0.785398, 1.570796, 3.1415926, 6.283185}
  Typischer Wertebereich: 0.0 ≤ `Phi` ≤ 6.283185 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1
- ▷ **Length1** (input_control) . . . . . rectangle2.hwidth(-array) ⇝ *real* / integer
  Hälfte der größeren Seitenlänge.
  Defaultwert: 48
  Wertevorschläge: `Length1` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich: 0 ≤ `Length1` ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10

▷ **Length2** (input_control) . . . . . rectangle2.hheight(-array) ⇝ *real / integer*
Hälfte der kürzeren Seitenlänge.
Defaultwert: 32
Wertevorschläge: Length2 ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich: 0 ≤ Length2 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Length2 < Length1

Beispiel

```
set_color(WindowHandle,'green')
draw_region(MyRegion,WindowHandle)
elliptic_axis(MyRegion,Ra,Rb,Phi)
area_center(MyRegion,_,Row,Column)
disp_rectangle2(WindowHandle,Row,Column,Phi,Ra,Rb)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert disp_rectangle2 den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_rgb, set_lut, set_hsi, set_draw, set_color, set_colored, set_line_width

Alternativen

disp_region, gen_rectangle2, disp_rectangle1, set_shape

Siehe auch

open_window, disp_region, set_color, set_draw, set_line_width

Modul

Foundation

disp_region (DispRegions : : WindowHandle :)

Ausgeben von Regionen in ein Fenster.

disp_region trägt die Regionen aus DispRegions in das Ausgabefenster ein. Die Parameter für die Ausgabe können mit den Operatoren set_color, set_gray, set_draw, set_line_width, etc. bestimmt werden.

Die Farbe(n) für die Darstellung der Regionen wird mit set_color, set_rgb, set_gray oder set_colored festgelegt. Wird mehr als eine Region ausgegeben und ist mehr als eine Farbe gesetzt (siehe set_color, set_colored), dann werden die Regionen zyklisch mit den angegebenen Farben ausgegeben.

Die Form der Region kann für die Ausgabe mit set_paint verändert werden (z.B. umschließender Kreis, konvex Hülle, etc.). Mit set_draw wird festgelegt, ob die Region ausgefüllt oder nur der Rand gezeichnet werden soll. Falls nur der Rand gezeichnet wird, kann mit set_line_width dessen Dicke in Pixeln und mit set_line_style das Darstellungsmuster angegeben werden.

Parameter

▷ **DispRegions** (input_object) . . . . . region(-array) ⇝ *object*
Auszugebende Regionen.

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

Beispiel

```
* Output with 12 colors:
set_colored(WindowHandle,12)
disp_region(SomeSegments,WindowHandle)

* Symbolic representation:
set_draw(WindowHandle,'margin')
set_color(WindowHandle,'red')
set_shape(WindowHandle,'ellipse')
disp_region(SomeSegments,WindowHandle)

* Representation of a margin with pattern:
set_draw(WindowHandle,'margin')
set_color(WindowHandle,'blue')
set_line_style(WindowHandle,[12,3])
disp_region(Segments,WindowHandle)
```

Ergebnis

`disp_region` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `set_shape`, `set_line_style`, `set_insert`, `set_draw`, `set_color`, `set_colored`, `set_line_width`

Alternativen

`disp_obj`, `disp_arrow`, `disp_line`, `disp_circle`, `disp_rectangle1`, `disp_rectangle2`, `disp_ellipse`

Siehe auch

`open_window`, `set_color`, `set_colored`, `set_draw`, `set_shape`, `set_paint`, `set_gray`, `set_rgb`, `set_hsi`, `set_line_width`, `set_line_style`, `set_insert`, `paint_region`, `dump_window`

Modul

Foundation

disp_xld (XLDObject : : WindowHandle :)

Ausgeben der Daten eines XLD-Objekts.

Mit `disp_xld` können XLD-Objekte eines beliebigen Typs ausgegeben werden.

Parameter

▷ **XLDObject** (input_object) ..... xld-array ⇝ *object*
Auszugebendes XLD-Objekt.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster Id.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Siehe auch

disp_image, disp_region, disp_channel, disp_color, disp_line, disp_arc

Modul

Foundation

11.3 Farbtabelle

get_lut (: : WindowHandle : LookUpTable)

Abfragen der aktuellen Farbtabelle.

get_lut gibt den Namen oder die Werte der Farbtabelle des gültigen Ausgabefensters aus, die von dem Operator disp_image (indirekt auch von disp_region etc.) verwendet wird. Gesetzt wird die Farbtabelle mit dem Operator set_lut. Handelt es sich bei der aktuellen Tabelle um eine Systemtabelle (die nicht durch set_fix modifiziert wurde), so wird der Name der Tabelle angegeben. Handelt es sich um eine modifizierte Tabelle, um eine von Datei eingelesene oder um eine Tabelle für Pseudo-Echtfarbenausgabe, so wird die Tabelle in Form von RGB-Werten ausgegeben.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fensteridentifikator.
- ▷ **LookUpTable** (output_control) . . . . . string-array ⇝ *string* / integer
  Name der Farbtabelle oder Tupel der RGB-Werte.

Ergebnis

get_lut liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_lut

Siehe auch

set_lut

Modul

Foundation

query_lut (: : WindowHandle : LookUpTable)

Abfragen aller verfügbaren Farbtabellen.

query_lut gibt die Namen aller Farbtabellen des aktuellen Gerätes an. Diese Farbtabellen können mit set_lut gesetzt werden. Immer vorhanden ist die Farbtabelle 'default'.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fensteridentifikator.
- ▷ **LookUpTable** (output_control) . . . . . string-array ⇝ *string*
  Namen der Farbtabellen.

Ergebnis

`query_lut` liefert den Wert TRUE, falls ein gültiges Fenster geöffnet ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`set_lut`, `disp_lut`

Siehe auch

`set_lut`

Modul

Foundation

set_lut (: : WindowHandle, LookUpTable :)

Setzen einer Farbtabelle.

`set_lut` setzt die Farbtabelle des Gerätes (Bildschirm), auf dem das Ausgabefenster dargestellt wird. Eine Farbtabelle ist die Umsetzung der „Grauwerte“ der Bildmatrix in einen Grauton oder eine Farbe auf dem Bildschirm. Für jeden Wert (0..255) wird festgelegt, in welcher Kombination aus rot grün und blau er dargestellt werden soll. Hierbei werden jedem Grauwert drei Intensitäten für die Grundfarben zugeordnet; deshalb spricht man von einer Farbtabelle. Die Umsetzung erfolgt in Echtzeit und bei jedem Bildschirmaufbau (typisch 60 bis 70 mal pro Sekunde) neu. Dies ermöglicht es, die Farbtabelle zu wechseln, um ein anderes Erscheinungsbild zu erhalten. Es ist zu beachten, dass die Farbtabelle nicht bei jedem Rechner gewechselt werden kann (z.B.: Schwarz/Weiß bzw. Echtfarbe).

Die Farbtabellen in HALCON sind (bei einem Rechner mit 256 Farben) in drei Bereiche aufgeteilt:

S: Systembereich, bzw. Benutzerbereich

G: Grafikfarben

B: Bilddaten

Die Farben in S stammen von den Anwendungen, die vor dem Start von HALCON aktiv waren, und weiterhin erhalten bleiben sollen. Die Grafikfarben G werden für Operatoren wie `disp_region`, `disp_circle`, etc. verwendet und sind in allen Farbtabellen gleich besetzt, d.h. die Ausgaben, die in einer Grafikfarbe erfolgen bleiben bei allen Farbtabellen erhalten. Grafikfarben werden mit `set_color` und `set_rgb` gesetzt. Die Grautöne bzw. Farben der Bilddaten B werden von dem Operator `disp_image` verwendet. Diese können sich in Abhängigkeit von der aktuellen Farbtabelle ändern. Eine Ausnahme vom oben Beschriebenen machen die Operatoren `set_gray` (setzen einer Farbe aus dem Bereich B für Operatoren wie `disp_region`) und `set_fix`, die eine Modifikation der Grafikfarben ermöglicht.

Bei den heute üblichen Bildschirmen kann nur eine Farbtabelle pro Bildschirm geladen werden. `set_lut` kann jedoch für jedes Fenster getrennt aufgerufen werden. Für dieses Problem gibt es folgende Lösung: Es wird immer diejenige Tabelle aktiviert, die dem gerade aktiven Fenster zugeordnet ist (dabei hat hier aktiv nichts mit `set_lut`, sondern vielmehr mit dem „Window-Manager“ zu tun).

Farb- und Grauwerttabellen können auch zusammen mit Echtfarben Grafikkarten verwendet werden. In diesem Fall wird die Tabelle durch Software simuliert, das heißt, dass das Bild bei der Ausgabe über die Tabelle bearbeitet wird.

Die Namen aller Farbtabellen können mit `query_lut` abgefragt werden. Die Farbtabellen unterscheiden sich in dem für die Grauwerte vorgesehenen Bereich. In diesem Bereich zeigen sie folgendes Verhalten:

Grauwerttabellen (1-7 Bildebenen)

Wert	Beschreibung
'default'	Es werden die beiden Grundfarben (i.allg. schwarz und weiß) verwendet.

Farbtabelle (Echtfarbe, statische Graustufen)

Wert	Beschreibung
'default'	Von der Hardware vorgegebene Tabelle.

Grauwerttabellen (256 Farben)

Wert	Beschreibung
'default'	Wie *'linear'*.
'linear'	Lineares Ansteigen der Grauwerte von 0(schwarz) nach 255 (weiß).
'inverse'	Inverse Funktion zu *'linear'*.
'sqr'	Ansteigen der Grauwerte als quadratische Funktion.
'inv_sqr'	Inverse Funktion zu *'sqr'*.
'cube'	Ansteigen der Grauwerte als 3. Potenz-Funktion.
'inv_cube'	Inverse Funktion zu *'cube'*.
'sqrt'	Ansteigen der Grauwerte als Quadratwurzelfunktion.
'inv_sqrt'	Inverse Funktion zu *'sqrt'*.
'cubic_root'	Ansteigen der Grauwerte als 3. Wurzel-Funktion.
'inv_cubic_root'	Inverse Funktion zu *'cubic_root'*.

Farbtabellen (256 Farben)

Wert	Beschreibung
'color1'	Linearer Übergang von rot über grün nach blau.
'color2'	Fließender Übergang von gelb über rot, blau nach grün.
'color3'	Fließender Übergang von gelb über rot, blau, grün, rot nach blau.
'color4'	Fließender Übergang von gelb über rot nach blau.
'three'	Darstellung der drei Farben rot, grün, blau.
'six'	Darstellung der sechs Grundfarben gelb, rot, magenta, blau, cyan, grün.
'twelve'	Darstellung von 12 Farben.
'twenty_four'	Darstellung von 24 Farben.
'rainbow'	Darstellung der Spektralfarben von rot über grün nach blau.
'temperature'	Wärmetabelle von schwarz über rot und gelb nach weiß.
'change1'	Farbänderung nach jedem Pixel der Tabelle, und zwar abwechselnd die sechs Grundfarben.
'change2'	5-fache Farbänderung von grün über rot nach blau.
'change3'	3-fache Farbänderung von grün über rot nach blau.
'jet'	Fließender Übergang von blau über grün, gelb nach rot.
'inverse_jet'	Fließender Übergang von rot über gelb, grün nach blau.

Weiterhin kann eine Farbtabelle aus einer Datei eingelesen werden. Diese Datei muss in jeder Zeile drei Zahlen von 0 bis 255 enthalten, wobei die erste Zahl dem Rotanteil, die zweite dem Grünanteil und die dritte dem Blauanteil für die Darstellung einer Farbe entspricht. Die Anzahl der Zeilen ist beliebig. Dabei wird die erste Zeile als erster Grauwert, die letzte als letzter Grauwert verwendet. Ist die Anzahl der Zeilen geringer als die Zahl der Grauwerte in der Farbtabelle, so werden die vorhandenen Werte gleichmäßig über das ganze Intervall verteilt. Ist die Anzahl zu hoch, werden (gleichmäßig verteilt) Zeilen bei der Besetzung der Farbtabelle ignoriert. Diese Datei

muss unter dem Namen „LookUpTable.lut“ abgespeichert sein. Der Dateiname wird ohne Extension angegeben. Die Datei wird im aktuellen Directory und dann in einem ausgezeichneten Directory (siehe set_system (::'lut_dir',<Pfad>:)) gesucht. Der Aufruf von set_lut kann auch mit einem Tupel von RGB-Werten erfolgen. Diese werden dann direkt gesetzt. Es ist zu beachten, dass die Anzahl der Parameterwerte, der Anzahl der aktuell verwendeten Pixel in der Farbtabelle entsprechen müssen.

Achtung

set_lut kann nur bei Bildschirmen mit 256 Graustufen/Farben verwendet werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fensteridentifikator.
- ▷ **LookUpTable** (input_control) . . . . . filename.read(-array) ⇝ *string* / integer
  Name der Farbtabelle, Werte der Farbtabelle (RGB) oder der Name der Datei.
  Defaultwert: 'default'
  Wertevorschläge: LookUpTable ∈ {'default', 'linear', 'inverse', 'sqr', 'inv_sqr', 'cube', 'inv_cube', 'sqrt', 'inv_sqrt', 'cubic_root', 'inv_cubic_root', 'color1', 'color2', 'color3', 'color4', 'three', 'six', 'twelve', 'twenty_four', 'rainbow', 'temperature', 'cyclic_gray', 'cyclic_temperature', 'hsi', 'change1', 'change2', 'change3', 'jet', 'jet_inverse'}
  Dateiendung: .lut

Beispiel

```
read_image(Image,'monkey')
query_lut(WindowHandle,LUTs)
for i := 0 to |LUTs|-1 by 1
  set_lut(WindowHandle,LUTs[i])
  fwrite_string(FileHandle, ['current table ',LUTs[i]])
  fnew_line(FileHandle)
  get_mbutton(WindowHandle,_,_,_)
endfor
```

Ergebnis

set_lut liefert den Wert TRUE, falls der Parameter korrekt ist, die Hardware über eine Farbtabelle verfügt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

query_lut, get_lut

Siehe auch

get_lut, query_lut, set_color, set_rgb, set_hsi, write_lut

Modul

Foundation

11.4 Fenster

clear_window (: : WindowHandle :)

Löschen des Inhalts eines Ausgabefensters.

clear_window löscht alle Einträge im Ausgabefenster. Das Fenster (Hintergrund und Rand) wird in seinen Anfangszustand versetzt. Parameter, die dem Fenster zugeordnet sind (z.B. mit set_color, set_paint, etc.), bleiben unverändert.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

Ergebnis

Ist das Ausgabefenster gültig, dann liefert `clear_window` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Alternativen

`clear_rectangle`, `disp_rectangle1`

Siehe auch

`open_window`

Modul

Foundation

close_window (: : WindowHandle :)

Schließen eines Ausgabefensters.

`close_window` schließt ein Fenster, das mit `open_window` geöffnet wurde. Das Ausgabegerät bzw. der Fensterbereich stehen danach wieder für neue Aufrufe von `open_window` zur Verfügung.

Parameter

▷ **WindowHandle** (input_control) . . . . . window(-array) ⇝ *handle*
Fenster-Handle.

Ergebnis

Ist das Ausgabefenster gültig, dann liefert `close_window` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- WindowHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_window`

Siehe auch

`open_window`

Modul

Foundation

copy_rectangle (: : WindowHandleSource, WindowHandleDestination, Row1, Column1, Row2, Column2, DestRow, DestColumn :)

Kopieren aller Pixel eines Rechtecks zwischen Ausgabefenstern.

copy_rectangle kopiert alle Pixel, die innerhalb des durch die Parameter Row1, Column1, Row2 und Column2 beschriebenen Rechtecks liegen, aus dem angegebenen Window mit der logischen Fensternummer WindowHandleSource in das angegebene Window mit der logischen Fensternummer WindowHandleDestination. Die Zielposition wird durch die linke obere Ecke des Rechtecks (DestRow, DestColumn) festgelegt.

Falls mehr als ein Rechteck bewegt werden soll, können diese gleichzeitig (in Form von Tupeln) übergeben werden.

copy_rectangle ist dafür gedacht, aufbereitete Grafiken aus einem „unsichtbaren“ Fenster in ein sichtbares Fenster zu kopieren. Hierzu wird ein Fenster mit der Option 'buffer' geöffnet. In dieses Fenster wird die Grafik ausgegeben und danach in ein sichtbares Fenster kopiert. Der Vorteil dieser Vorgehensweise besteht darin, dass copy_rectangle viel schneller als Ausgabeprozeduren wie z.B. disp_channel ist. Dies ist insbesondere bei Demo-Programmen von Vorteil. Es lassen sich hiermit auch kurze „clips“ realisieren: Hierzu wird für jedes Bild der Folge ein Fenster vom Typ 'buffer' angelegt und die Daten darauf ausgegeben. Ausgegeben wird die Bildfolge, indem alle Puffer nacheinander in ein sichtbares Fenster kopiert werden.

Achtung

Beide Fenster müssen auf demselben Display liegen.

Parameter

- ▷ **WindowHandleSource** (input_control) . . . . . window ⇝ *handle*
  Nummer des Quellfensters.
- ▷ **WindowHandleDestination** (input_control) . . . . . window ⇝ *handle*
  Nummer des Zielfensters.
- ▷ **Row1** (input_control) . . . . . rectangle.origin.y(-array) ⇝ *integer*
  Zeilenindex der linken oberen Ecke im Quellfenster.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ Row1 ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Column1** (input_control) . . . . . rectangle.origin.x(-array) ⇝ *integer*
  Spaltenindex der linken oberen Ecke im Quellfenster.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ Column1 ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Row2** (input_control) . . . . . rectangle.corner.y(-array) ⇝ *integer*
  Zeilenindex der rechten unteren Ecke im Quellfenster.
  Defaultwert: 128
  Typischer Wertebereich: $0 \leq$ Row2 ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: Row2 >= Row1
- ▷ **Column2** (input_control) . . . . . rectangle.corner.x(-array) ⇝ *integer*
  Spaltenindex der rechten unteren Ecke im Quellfenster.
  Defaultwert: 128
  Typischer Wertebereich: $0 \leq$ Column2 ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: Column2 >= Column1
- ▷ **DestRow** (input_control) . . . . . point.y(-array) ⇝ *integer*
  Zeilenindex der linken oberen Ecke im Zielfenster.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ DestRow ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

▷ **DestColumn** (input_control) . . . . . point.x(-array) ⇝ *integer*
Spaltenindex der linken oberen Ecke im Zielfenster.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ DestColumn ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Beispiel

```
read_image(Image,'monkey')
open_window(0,0,-1,-1,'root','buffer','',WindowHandleBuffer)
disp_obj(Image,WindowHandleBuffer)
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
draw_rectangle1 (WindowHandle, Row1, Column1, Row2, Column2)
copy_rectangle(WindowHandleBuffer,WindowHandle,\
               Row1, Column1, Row2, Column2,Row1,Column1)
close_window(WindowHandleBuffer)
```

Ergebnis

Ist das Ausgabefenster gültig und sind die angegebenen Parameter korrekt, dann liefert copy_rectangle den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

close_window

Alternativen

move_rectangle, slide_image

Siehe auch

open_window

Modul

Foundation

dump_window (: : WindowHandle, Device, FileName :)

Ausgeben des Fensterinhaltes in eine Datei.

dump_window gibt den Inhalt des Fensters in eine Datei aus. Diese Datei kann dann mit geeigneten Druckern oder anderen Programmen weiterverarbeitet werden. Der Bildschirminhalt wird dabei jeweils für ein spezielles Gerät (Device) aufbereitet, d.h. so formatiert, dass die Datei direkt ausgedruckt oder von einem Grafikprogramm weiterverarbeitet werden kann.

Für die Umsetzung der Grauwerte wird die aktuelle Farbtabelle des Fensters verwendet.

Mögliche Werte für Device

'postscript': PostScript - Datei.
Dateiendung: '.ps'

'postscript',Breite,Höhe: PostScript - Datei mit Angabe der Ausgabegröße. $Width$ und $Height$ beziehen sich dabei auf die Größe des Ausdrucks. In diesem Fall wird ein Tupel mit den drei Werten als Device übergeben.
Dateiendung: '.ps'

'tiff': TIFF-Format; Komprimierung ist möglich mit Adobe-Deflate-Komprimierung (*'deflate [num]'*, verlustfrei), JPEG (*'jpeg [num]'*, verlustbehaftet), LZW (*'lzw'*, verlustfrei) und PackBits (*'packbits'*, verlustfrei). *'[num]'* bedeutet hier eine optionale Angabe der Komprimierung. Bei *'deflate'* kann eine Zahl zwischen *0* (keine Komprimierung) und *9* (maximale Komprimierung) angegeben werden. Bei *'jpeg'* kann eine Zahl zwischen *0* und *100* angegeben werden. Die Semantik ist identisch mit der unten bei 'jpeg' beschriebenen Semantik. Die Optionen können durch Leerzeichen getrennt aneinander gefügt werden. Beispiele: *'tiff deflate 9'*: Maximale Adobe-Deflate-Komprimierung; *'tiff jpeg 90'*: JPEG-Komprimierung mit hoher Qualität; *'tiff lzw'*: LZW-Komprimierung; *'tiff'* oder *'tiff none'*: Keine Komprimierung.

Dateiendung: '.tif'

'bmp': Windows-BMP-Format, RGB-Bild, 3 Byte pro Pixel;

Dateiendung: '.bmp'

'jpeg': JPEG-Format (verlustbehaftete Komprimierung); Zusammen mit dem Format kann ein Qualitätsmaß angeben werden, das die Komprimierungsrate und die Qualität des gespeicherten Bildes bestimmt. Große Werte (Maximum 100) erzeugen eine relativ große Datei mit hoher Qualität, bei kleine Werten nimmt die Qualität und die Dateigröße deutlich ab; Bsp.: 'jpeg 30'.

Dateiendung: '.jpg'

'jpegxr': JPEG-Format (verlustfreie und verlustbehaftete Komprimierung); Zusammen mit dem Format kann ein Qualitätsmaß angeben werden, das die Komprimierungsrate und die Qualität des gespeicherten Bildes bestimmt. Große Werte (Maximum 100) erzeugen eine relativ große Datei mit hoher Qualität, bei kleine Werten nimmt die Qualität und die Dateigröße deutlich ab; Bsp.: 'jpegxr 30'.

Dateiendung: '.jpg'

'jp2': JPEG2000-Format (verlustfreie und verlustbehaftete Komprimierung); Zusammen mit dem Format kann ein Maß angegeben werden, welches die Komprimierungsrate und damit die Qualität des gespeicherten Bildes bestimmt (z.B. 'jp2 40'). Der angegebene Wert entspricht ungefähr dem Verhältnis der Größe des komprimierten Bildes zur Größe des unkomprimierten Bildes (in Prozent). Da JPEG2000 ohnehin stark komprimiert, haben erst kleinere Werte (typischerweise unter 50) Einfluss auf die Dateigröße. Wird kein Wert angegeben (und nur dann) wird das Bild verlustfrei komprimiert.

Dateiendung: '.jp2'

'png': PNG-Format (verlustfreie Komprimierung); zusammen mit dem Format kann ein Komprimierungsgrad zwischen 0 und 9 angegeben werden, wobei 0 keiner Kompression und 9 der bestmöglichen Kompression entspricht. Alternativ kann die Kompression über folgende Konstanten ausgewählt werden: 'best', 'fastest' und 'none'. Korrekte Formatparameter sind also z.B. 'png', 'png 7' und 'png none'.

Dateiendung: '.png'

Achtung

Unter unixartigen Systemen muss das Grafikfenster vollständig auf dem Root-Fenster sichtbar sein, da sonst der Fensterinhalt aufgrund von Einschränkungen in X Windows nicht ausgelesen werden kann. Sollen Ausgaben von größeren Darstellungen in eine Datei geschrieben werden, bietet sich der Fenstertyp *'pixmap'* an.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Device`** (input_control) ..... string(-array) ⇝ *string* / integer
Name des Zielgerätes bzw. des Grafikformats.
Defaultwert: 'postscript'
Wertevorschläge: `Device` ∈ {'postscript', 'tiff', 'tiff deflate 9', 'tiff jpeg 90', 'tiff lzw', 'tiff packbits', 'bmp', 'jpeg', 'jpeg 100', 'jpeg 80', 'jpeg 60', 'jpeg 40', 'jpeg 20', 'jpegxr', 'jpegxr 50', 'jpegxr 40', 'jpegxr 30', 'jpegxr 20', 'jp2', 'jp2 50', 'jp2 40', 'jp2 30', 'jp2 20', 'png', 'png best', 'png fastest', 'png none'}

▷ **`FileName`** (input_control) ..... filename.write ⇝ *string*
Dateiname (ohne Extension).
Defaultwert: 'halcon_dump'

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
read_image (Image, 'fabrik')
gen_circle (Circle, 200, 200, 100.5)
```

```
dev_display (Image)
dev_display (Circle)
dump_window(WindowHandle,'postscript','halcon_dump')
```

Ergebnis

`dump_window` liefert den Wert TRUE, falls das passende Fenster gültig ist und die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`, `disp_region`

Nachfolger

`system_call`

Siehe auch

`open_window`, `set_system`, `dump_window_image`

Modul

Foundation

dump_window_image (: Image : WindowHandle :)

Ausgeben des Fensterinhaltes in ein Bildobjekt.

`dump_window_image` gibt den Inhalt des Fensters (`WindowHandle`) in ein Bildobjekt (`Image`) aus. Für die Umsetzung der Grauwerte wird die aktuelle Farbtabelle des Fensters verwendet.

Achtung

Unter unixartigen Systemen muss das Grafikfenster vollständig auf dem Root-Fenster sichtbar sein, da sonst der Fensterinhalt aufgrund von Einschränkungen in X Windows nicht ausgelesen werden kann.

Parameter

- ▷ **Image** (output_object) ..... image ⇝ *object* : byte
  Gespeichertes Bild.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.

Ergebnis

`dump_window_image` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`, `disp_region`

Siehe auch

`open_window`, `set_system`, `dump_window`

Modul

Foundation

flush_buffer (: : WindowHandle :)

Aktualisieren des Inhalts eines Fensters.

flush_buffer aktualisiert den Inhalt des Fensters WindowHandle.

flush_buffer muss dann verwendet werden, wenn der Fensterparameter *'flush'* mit set_window_param auf *'false'* gesetzt wurde. In diesem Fall werden alle Anzeigeoperationen wie disp_obj in einen Puffer umgeleitet und haben keinen Effekt auf das Fenster WindowHandle (dies gilt für alle Fenstermodi). flush_buffer kopiert den Inhalt dieses Puffers in das Fenster WindowHandle.

Dies ist nützlich, um eine flackerfreie Darstellung zu erreichen. So können mehrere Anzeigeoperationen (z.B. clear_window gefolgt von einem disp_obj) zusammengefasst werden und das finale Resultat mit flush_buffer angezeigt werden.

Dies betrifft nicht die Zeichenobjekte, welche immer aktualisiert werden.

Achtung

flush_buffer benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

Beispiel

```
read_image (Image, 'printer_chip/printer_chip_01')
threshold (Image, Region, 128, 255)
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
set_window_param (WindowHandle, 'flush', 'false')
dev_display (Image)
dev_display (Region)
disp_text (WindowHandle, 'Result of threshold', 'window', \
           12, 12, 'black', [], [])
* nothing is displayed until flush_buffer is called
flush_buffer (WindowHandle)
```

Ergebnis

flush_buffer liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

disp_obj

Nachfolger

dump_window_image

Siehe auch

set_window_param

Modul

Foundation

get_disp_object_model_3d_info (: : WindowHandle, Row, Column, Information : Value)

Zurückgeben der Tiefe oder des Indexes eines dargestellten 3D Objektmodells.

get_disp_object_model_3d_info gibt Informationen über die 3D Objektmodelle, die im Fenster WindowHandle an den Positionen (Row, Column) mit disp_object_model_3d angezeigt wurden in Value zurück.

Mit dem Parameter Information wird ausgewählt, welche Werte zurückgegeben werden:

'object_index' Die Indices der 3D Objektmodelle, die an den Positionen (Row, Column) dargestellt wurden. Befindet sich kein 3D Objektmodell an einer dieser Stellen, wird -1 zurückgegeben. Um diese Information ermitteln zu können, muss bei disp_object_model_3d der generische Parameter *'object_index_persistence'* auf *'true'* gesetzt worden sein.

'depth' Die Tiefe (d.h. die Z Koordinate im Kamerakoordinatensystem) an den Positionen (Row, Column). Befindet sich kein 3D Objektmodell an einer dieser Stellen, wird -1.0 zurückgegeben. Um diese Information ermitteln zu können, muss bei disp_object_model_3d der generische Parameter *'depth_persistence'* auf *'true'* gesetzt worden sein.

Die Fensterkoordinaten Row und Column müssen so angegeben werden, dass sie dem gesetzten Bildausschnitt entsprechen. Sie können daher auch subpixelgenau sein. Ist der Bildausschnitt durch (Zeile1,Spalte1,Zeile2, Spalte2) gegeben, entspricht die linke obere Ecke den Koordinaten (Zeile1 - 0.5, Spalte1 - 0.5), die rechte untere Ecke den Koordinaten (Zeile2 - 0.5, Spalte2 - 0.5). Die Operatoren get_mposition_sub_pix oder get_mbutton_sub_pix können verwendet werden, um diese Koordinaten direkt zu erhalten.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fensteridentifikator.
- ▷ **Row** (input_control) ..... integer(-array) ⇝ *real* / integer
  Zeilenkoordinaten.
- ▷ **Column** (input_control) ..... integer(-array) ⇝ *real* / integer
  Spaltenkoordinaten.
- ▷ **Information** (input_control) ..... string(-array) ⇝ *string*
  Abzufragende Information.
  Defaultwert: 'depth'
  Werteliste: Information ∈ {'depth', 'object_index'}
- ▷ **Value** (output_control) ..... integer(-array) ⇝ *integer* / real
  Werte an (Row,Column).

Ergebnis

get_disp_object_model_3d_info liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

disp_object_model_3d, get_mbutton, get_mbutton_sub_pix, get_mposition, get_mposition_sub_pix

Siehe auch

disp_object_model_3d

Modul

3D Metrology

get_os_window_handle (: : WindowHandle : OSWindowHandle, OSDisplayHandle)

Auslesen der Betriebssystem-Fenster-Handles.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit `dev_open_window` geöffnet wurden.

`get_os_window_handle` gibt den Betriebssystem-Fenster-Handle des HALCON-Fensters `WindowHandle` in `OSWindowHandle` zurück. Unter unixartigen Systemen wird zusätzlich in `OSDisplayHandle` der Betriebssystem-Display-Handle des HALCON-Fensters zurückgeliefert. Dies kann dazu verwendet werden, um mit Betriebssystem-Funktionen auf das Fenster zuzugreifen, um z.B. darin benutzerdefiniert zu zeichnen. Unter Windows kann `OSWindowHandle` in eine Variable vom Typ HWND umgewandelt werden. Unter unixartigen Systemen kann `OSWindowHandle` in eine Variable vom Typ Window umgewandelt werden und `OSDisplayHandle` in den Typ Display.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`OSWindowHandle`** (output_control) ..... pointer ⇝ *integer*
  Betriebssystem-Fenster-Handle.
- ▷ **`OSDisplayHandle`** (output_control) ..... pointer ⇝ *integer*
  Betriebssystem-Display-Handle (nur unter unixartigen Systemen).

Beispiel

```
/* Draw a line into a HALCON window under Unix-like systems using X11 calls. */
#include "HalconC.h"
#include <X11/X.h>
#include <X11/Xlib.h>

int main(int argc, char **argv)
{
  Hlong hwin, win, disp;
  Display *display;
  Window window;
  GC gc;
  XGCValues values;
  static char dashes[] = { 20, 20 };

  open_window(0, 0, 500, 500, 0, "visible", "", &hwin);
  get_os_window_handle(hwin, &win, &disp);
  display = (Display *)disp;
  window = (Window)win;
  gc = XCreateGC(display, window, 0, &values);
  XSetFunction(display, gc, GXset);
  XSetLineAttributes(display, gc, 10, LineOnOffDash, CapRound, JoinRound);
  XSetDashes(display, gc, 0, dashes, 2);
  XSetForeground(display, gc, WhitePixel(display, DefaultScreen(display)));
  XSetBackground(display, gc, BlackPixel(display, DefaultScreen(display)));
  XDrawLine(display, win, gc, 20, 20, 480, 480);
  XFlush(display);
  XFreeGC(display, gc);
  wait_seconds(5);
  return 0;
}

/* Draw a line into a HALCON window under Windows using GDI calls. */
#include "HalconC.h"
#include "windows.h"

int main(int argc, char **argv)
{
  Hlong hwin, win, disp;
  HDC hdc;
```

```
  HPEN hpen;
  HPEN *hpen_old;
  LOGBRUSH logbrush;
  POINT point;
  static DWORD dashes[] = { 20, 20 };

  open_window(0, 0, 500, 500, 0, "visible", "", &hwin);
  get_os_window_handle(hwin, &win, &disp);
  logbrush.lbColor = RGB(255,255,255);
  logbrush.lbStyle = BS_SOLID;
  hpen =  ExtCreatePen(PS_USERSTYLE|PS_GEOMETRIC, 10, &logbrush, 2, dashes);
  hdc = GetDC((HWND)win);
  hpen_old = (HPEN *)SelectObject(hdc, hpen);
  MoveToEx(hdc, 20, 20, &point);
  LineTo(hdc, 480, 480);
  DeleteObject(SelectObject(hdc, hpen_old));
  ReleaseDC((HWND)win, hdc);
  wait_seconds(5);
  return 0;
}
```

Ergebnis

`get_os_window_handle` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Modul

Foundation

`get_window_attr` (: : AttributeName : AttributeValue)

Lesen von Fenstereigenschaften.

`get_window_attr` dient zum Auslesen von Eigenschaften von Grafikfenstern, die mit `set_window_attr` gesetzt wurden. Folgende Parameter eines Fenster können ausgelesen werden:

'border_width' Breite des Fensterrandes in Pixel.

'border_color' Farbe des Fensterrandes.

'background_color' Hintergrundfarbe des Fensters.

'window_title' Name des Fensters in der Titelleiste.

Parameter

▷ **`AttributeName`** (input_control) . . . . . string ⇝ *string*
Name des Attributs, das gelesen werden soll.
Werteliste: `AttributeName` ∈ {'border_width', 'border_color', 'background_color', 'window_title'}

▷ **`AttributeValue`** (output_control) . . . . . string ⇝ *string* / integer
Attributwert.

Ergebnis

`get_window_attr` liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`

Siehe auch

`open_window`, `set_window_attr`

Modul

Foundation

`get_window_extents` `( : : WindowHandle : Row, Column, Width, Height )`

Abfragen von Informationen über die Größe und Position eines Fensters.

`get_window_extents` gibt die Position der linken oberen Ecke, sowie die Breite und die Höhe des Ausgabefensters aus.

Achtung

Die Größe und die Position eines Fensters können durch den Window-Manager verändert werden, ohne dass dies vom Programm aus vorgenommen wurde. Deshalb können sich die Werte, die `get_window_extents` liefert, durch diesen Seiteneffekt ändern.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`Row`** (output_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilenindex der linken oberen Ecke des Fensters.
- ▷ **`Column`** (output_control) ..... rectangle.origin.x ⇝ *integer*
  Spaltenindex der linken oberen Ecke des Fensters.
- ▷ **`Width`** (output_control) ..... rectangle.extent.x ⇝ *integer*
  Breite des Fensters.
- ▷ **`Height`** (output_control) ..... rectangle.extent.y ⇝ *integer*
  Höhe des Fensters.

Beispiel

```
open_window(100,100,200,200,'root','visible','',WindowHandle)
fwrite_string(FileHandle, 'Move the window with the mouse!')
fnew_line(FileHandle)
repeat
get_mbutton(WindowHandle,_,_,Button)
get_window_extents(WindowHandle,Row,Column,Width,Height)
fwrite_string(FileHandle, ['(',Row,',',Column,')'])
fnew_line(FileHandle)
until(Button == 4)
```

Ergebnis

`get_window_extents` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`

Siehe auch

`set_window_extents`, `open_window`

Modul

Foundation

`get_window_pointer3` `( : : WindowHandle : ImageRed, ImageGreen, ImageBlue, Width, Height )`

Zugreifen auf die Pixeldaten eines Fensters.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit `dev_open_window` geöffnet wurden.

`get_window_pointer3` ermöglicht (bei einigen Fenstersystemen) den direkten Zugriff auf die Bitmap. Rückgabewerte sind die drei Zeiger auf die Farbauszüge eines 24-Bit Fensters (`ImageRed`, `ImageGreen`,`ImageBlue`) sowie die Fenstergröße (`Width`,`Height`). In der Programmiersprache C ist der Typ der Bildpunkte unsigned char.

Achtung

`get_window_pointer3` ist nur für den Fenster-Typ *'pixmap'* realisiert.

Parameter

- ▷ **`WindowHandle`** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **`ImageRed`** (output_control) . . . . . integer ⇝ *integer*
  Zeiger auf den Rotkanal der Pixeldaten.
- ▷ **`ImageGreen`** (output_control) . . . . . integer ⇝ *integer*
  Zeiger auf den Grünkanal der Pixeldaten.
- ▷ **`ImageBlue`** (output_control) . . . . . integer ⇝ *integer*
  Zeiger auf den Blaukanal der Pixeldaten.
- ▷ **`Width`** (output_control) . . . . . extent.x ⇝ *integer*
  Länge einer Bildzeile.
- ▷ **`Height`** (output_control) . . . . . extent.y ⇝ *integer*
  Anzahl der Bildzeilen.

Ergebnis

`get_window_pointer3` liefert den Wert TRUE, falls das Fenster vom Typ *'pixmap'* existiert und gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Siehe auch

`open_window`, `set_window_type`

Modul

Foundation

get_window_type (: : WindowHandle : WindowType)

Abfragen des Fenstertyps.

get_window_type bestimmt, von welchem Typ bzw. welcher Grafiksoftware das Ausgabegerät des Windows ist. Die verfügbaren Typen von Ausgabegeräten können mit dem Operator query_window_type abgefragt werden. get_window_type kann dort sinnvoll eingesetzt werden, wo man maschinenunabhängige Programme entwickeln möchte. Mögliche Werte sind:

'X-Window' X-Window Version 11.

'WIN32-Window' Microsoft Windows.

'Cocoa' macOS.

'pixmap' Fenster werden nicht angezeigt, sondern nur im Speicher verwaltet. Auf diese Weise können HALCON-Programme auch auf Rechnern ohne Grafikbildschirm portiert werden.

'PostScript' Die Ausgabe von Objekten erfolgt in eine PostScript-Datei.

'default' Aktueller Fenstertyp.

'system_default' Standardfenstertyp für aktuelle Plattform.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle* / string
Fenster-Handle.
Wertevorschläge: WindowHandle ∈ {'default', 'system_default'}

▷ **WindowType** (output_control) ..... string ⇝ *string*
Art des Fensters

Beispiel

```
open_window(100,100,200,200,'root','visible','',WindowHandle)
get_window_type(WindowHandle,WindowType)
fwrite_string(FileHandle, ['Window type: ',WindowType])
fnew_line(FileHandle)
```

Ergebnis

get_window_type liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Siehe auch

query_window_type, set_window_type, get_window_pointer3, open_window

Modul

Foundation

new_extern_window (: : WINHWnd, Row, Column, Width, Height : WindowHandle)

Erzeugen eines virtuellen Grafikfensters unter Windows.

new_extern_window öffnet ein neues virtuelles Fenster. Bitte beachten Sie, dass das nur auf Windows-Systemen funktioniert und Zeichenobjekte nicht unterstützt sind.

Virtuell bedeutet hier, dass kein neues Fenster erstellt wird, sondern dass das Fenster, dessen Window-Handle im Parameter WINHWnd übergeben wird, zur Ausgabe von Grauwertdaten, Regionen, Grafik sowie für Textausgabe verwendet wird.

Auf Mausinteraktionen basierende Operatoren wie zum Beispiel draw_rectangle1 können auf externen Fenstern nicht verwendet werden. Die folgenden Operatoren können verwendet werden:

- Grauwertausgabe: set_paint, set_lut
- Regionen: set_color, set_rgb, set_hsi, set_gray, set_shape, set_line_width, set_insert, set_line_style, set_draw
- Bildausschnitt: set_part
- Text: set_font

Die aktuell gesetzten Werte können mit Operatoren wie get_shape abgefragt werden. Da einige Parameter durch die Hardware (Auflösung/Farben) bestimmt werden, kann man mit Operatoren wie query_color die aktuell verfügbaren Ressourcen abfragen.

Im Parameter WINHWnd wird der Fensterhandle des Windows-Fensters übergeben, in das die Ausgaben von HALCON erfolgen sollen.

Der Ursprung des Koordinatensystems des Fensters liegt in der linken oberen Ecke (Koordinaten: (0,0)). Nach unten nimmt der Zeilenindex zu (maximal: Fensterhöhe-1), nach rechts steigt der Spaltenindex (maximal: Fensterbreite-1) an.

Für die Parameter Width und Height kann der Wert *-1* als Größe verwendet werden. Dies bedeutet, dass der entsprechende Wert automatisch bestimmt werden soll. Dies ist insbesondere dann von Bedeutung, wenn das Pixelverhältnis nicht gleich 1.0 ist (siehe set_system). Wird einer der beiden Parameter als *-1* angegeben, so wird er durch die Größe bestimmt, die sich aus dem Pixelverhältnis ergibt. Werden beide Parameter mit *-1* festgelegt, so werden sie auf das aktuelle Bildformat gesetzt.

Die Position und Größe des Fensters kann sich im Laufe des Programms ändern, z.B. durch externe Eingriffe (Window-Manager). Mit dem Operator set_window_extents kann die Größe des (externen) Fensters auch vom Programm aus geändert werden. Beachten Sie bitte, dass set_window_extents auch die Position von Fenstern ändern könnte, allerdings die Position eines externen Fensters unter Windows nicht geändert werden kann.

Die Farbe für Grafik und Schrift, die für Ausgabeprozeduren wie disp_region oder disp_circle verwendet wird, wird mit set_rgb, set_hsi oder set_gray festgelegt. Mit set_insert wird eingestellt, wie die Grafik mit dem Inhalt des Bildwiederholspeichers verknüpft wird. Hierbei kann z.B. mit set_insert(::'not':) erreicht werden, dass nach zweimaligem Zeichnen eines Kreises an der gleichen Position dieser wieder beseitigt wird.

Der Inhalt von Fenstern wird nicht gesichert, falls sich Fenster überlappen. Dies muss durch den Programmcode, der das Windows-Fenster verwaltet, sichergestellt werden.

Für die Ausgabe von Grafik (disp_image,disp_region, etc.) kann mit dem Operator set_part das Fenster so eingestellt werden, dass es einen logischen Ausschnitt des Bildformates darstellt. Dies hat insbesondere zur Folge, dass von Bildern und Regionen lediglich dieser Ausschnitt (entsprechend vergrößert) ausgegeben wird. Bevor Sie Ihr Fenster schließen, müssen Sie das HALCON-Fenster schließen.

Achtung

Es ist zu beachten, dass die Parameter Row, Column, Width und Height durch das Ausgabegerät, d.h. die Größe des Windows-NT-Desktops, beschränkt sind. Ebenso ist zu beachten, dass sich auf Mausinteraktionen basierende Operatoren wie zum Beispiel draw_rectangle1 auf externen Fenstern nicht verwenden lassen.

Parameter

▷ **WINHWnd** (input_control) . . . . . pointer ⇝ *integer*
Windows-Fensterhandle eines vorher geöffneten Fensters.
Restriktion: WINHWnd != 0

▷ **Row** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilenkoordinate der linken oberen Ecke.
Defaultwert: 0
Restriktion: Row >= 0

▷ **Column** (input_control) ..... rectangle.origin.x ⇝ *integer*
Spaltenkoordinate der linken oberen Ecke.
Defaultwert: 0
Restriktion: Column >= 0

▷ **Width** (input_control) ..... rectangle.extent.x ⇝ *integer*
Breite des Fensters.
Defaultwert: 512
Restriktion: Width > 0 || Width == -1

▷ **Height** (input_control) ..... rectangle.extent.y ⇝ *integer*
Höhe des Fensters.
Defaultwert: 512
Restriktion: Height > 0 || Height == -1

▷ **WindowHandle** (output_control) ..... window ⇝ *handle*
Fenster-Handle.

Beispiel

```
// Needs to be embedded into an application framework,
// see examples/cpp/mfc/MatchingExtWin.
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `new_extern_window` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`reset_obj_db`

Nachfolger

`set_color`, `query_window_type`, `get_window_type`, `set_window_type`, `get_mposition`, `set_tposition`, `set_tshape`, `set_window_extents`, `get_window_extents`, `query_color`, `set_check`, `set_system`

Alternativen

`open_window`

Siehe auch

`open_window`, `disp_region`, `disp_image`, `disp_color`, `set_lut`, `query_color`, `set_color`, `set_rgb`, `set_hsi`, `set_pixel`, `set_gray`, `set_part`, `set_part_style`, `query_window_type`, `get_window_type`, `set_window_type`, `get_mposition`, `set_tposition`, `set_window_extents`, `get_window_extents`, `set_window_attr`, `set_check`, `set_system`

Modul

Foundation

open_window (: : Row, Column, Width, Height, FatherWindow, Mode, Machine : WindowHandle)

Öffnen eines Grafikfensters.

`open_window` öffnet ein neues Fenster, das zur Ausgabe von Grauwertdaten, Regionen, Grafik sowie für Textausgabe verwendet werden kann. Alle Ausgaben (`disp_region`, `disp_image`, etc.) werden auf dieses Fenster geleitet, wenn dieselbe logische Fensternummer `WindowHandle` verwendet wird.

Der Hintergrund des erzeugten Fensters wird schwarz vorbesetzt und es hat einen weißen Rand von 2 Pixel Breite (siehe set_window_attr(::'border_width',<Breite>:)).

Einem Fenster werden bestimmte Parameter für die Aufbereitung der auszugebenden Daten zugeordnet, die bei der eigentlichen Ausgabe (z.B. mit disp_image oder disp_region) berücksichtigt werden. Diese Parameter werden aber nicht von der Ausgabeprozedur, sondern von „Konfigurationsprozeduren“ festgelegt. Will man z.B. die Farbe Rot für die Ausgabe von Regionen setzen, so muss vor dem Befehl disp_region der Aufruf set_color(::WindowHandle,'red':) erfolgen. Diese Parameter werden immer für das Fenster mit der logischen Fensternummer WindowHandle gesetzt und bleiben dem Fenster solange zugeordnet, bis sie überschrieben werden. Es stehen u.a. folgende Konfigurationsprozeduren zur Verfügung:

- Grauwertausgabe: set_paint, set_lut
- Regionen: set_color, set_rgb, set_hsi, set_gray, set_shape, set_line_width, set_insert, set_line_style, set_draw
- Bildausschnitt: set_part
- Text: set_font

Die aktuell gesetzten Werte können mit Operatoren wie get_shape abgefragt werden. Da einige Parameter durch die Hardware (Auflösung/Farben) bestimmt werden, kann man mit Operatoren wie query_color die aktuell verfügbaren Ressourcen abfragen.

Der Ursprung des Koordinatensystems des Fensters liegt in der linken oberen Ecke (Koordinaten: (0,0)). Nach unten nimmt der Zeilenindex zu (maximal: Bildhöhe - 1), nach rechts steigt der Spaltenindex (maximal: Bildbreite - 1) an. Es ist zu beachten, dass der Wertebereich des Koordinatensystems unabhängig von der Fenstergröße ist. Er wird nur durch das Bildformat (siehe reset_obj_db) festgelegt.

Der Parameter Machine gibt den Namen des Rechners an, auf dem das Fenster geöffnet werden soll. Im Falle von X-Window wird bei TCP-IP nur der Name, bei DEC-Net wird noch ein Doppelpunkt hinter den Namen gesetzt. Der „Server“ bzw. der „Screen“ wird nicht mit angegeben. Übergibt man den leeren String, so wird die Environmentvariable DISPLAY zur Festlegung des Zielrechners verwendet. Hierbei wird der Name in der üblichen Syntax

```
<Host>:0.0
```

angegeben.

Mit dem Parameter FatherWindow kann für die Fenstertypen *'WIN32-Window'*, *'X-Window'* und *'Cocoa'* das Vaterfenster für das zu öffnende Fenster angegeben werden. Ist die Kontrolle 'father' mittels set_check eingeschaltet, so muss FatherWindow ein HALCON-Fenster sein, andernfalls (set_check('~father')) kann es auch ein Betriebssystem-Fenster sein. Wird für FatherWindow der Wert 0 oder 'root' übergeben, so ist der Desktop (Windows) bzw. das Root-Fenster (unixartige Systeme) das Vaterfenster. In diesem Fall ist die Einstellung der Kontrolle 'father' mittels set_check nicht relevant. Der Aufrufer muss sicherstellen, dass FatherWindow ein gültiges Fensterhandle ist und dass dieses nicht zerstört wird, solange das eingebettete HALCON Fenster verwendet wird.

Für die Parameter Width und Height kann der Wert „-1“ als Größe verwendet werden. Dies bedeutet, dass der entsprechende Wert automatisch bestimmt werden soll. Dies ist insbesondere dann von Bedeutung, wenn das Pixelverhältnis nicht gleich 1.0 ist (siehe set_system). Wird einer der beiden Parameter als „-1“ angegeben, so wird er durch die Größe bestimmt, die sich aus dem Pixelverhältnis ergibt. Werden beide Parameter mit „-1“ festgelegt, so werden sie auf das maximale Bildformat gesetzt (weitergehende Informationen zum maximalen Bildformat finden sich in der Beschreibung von get_system unter „width“ und „height“).

Die Position und Größe des Fensters kann sich im Laufe des Programms ändern. Dies kann zum einen mit set_window_extents erfolgen, aber auch durch externe Eingriffe (Window-Manager) hervorgerufen werden. Für den zweiten Fall ist der Operator set_window_extents vorgesehen.

Dem Fenster wird beim Öffnen ein sog. Defaultfont zugeordnet. Dieser wird für Operatoren wie write_string verwendet und kann mit set_font nach dem Aufruf von open_window überschrieben werden. Es ist jedoch auch möglich, den Defaultfont mit dem Aufruf set_system(::'default_font',<Fontname>:) vor dem Öffnen eines Fensters (und auch aller weiteren) festzulegen (siehe hierzu auch query_font).

Die Farbe für Grafik und Schrift, die für Ausgabeprozeduren wie disp_region oder disp_circle verwendet wird, wird mit set_rgb, set_hsi oder set_gray festgelegt. Mit set_insert wird eingestellt, wie die Grafik mit dem Inhalt des Bildwiederholspeichers verknüpft wird. Hierbei kann z.B. mit set_insert(::

'not':) erreicht werden, dass nach zweimaligem Zeichnen eines Kreises an der gleichen Position dieser wieder beseitigt wird.

Normalerweise wird jede Ausgabe (z.B. disp_region, disp_image, disp_circle, etc.) in ein Fenster durch einen sog. „Flush“ abgeschlossen. Dies bewirkt, dass die Daten nach Beendigung der Ausgabeprozedur auf dem Bildschirm vollständig sichtbar werden. Dies ist aber nicht in allen Fällen erforderlich, insbesondere dann, wenn ständig Ausgaben erfolgen, oder ein Maus-Prozedur aktiv ist. Hier ist es günstiger (d.h. schneller) wenn die Daten gepuffert werden, bis genügend Daten vorhanden sind. Dieses Verhalten kann durch den Aufruf von set_system(::'flush_graphic','false':) eingestellt werden.

Der Inhalt von Fenstern wird (falls die Treibersoftware dies unterstützt) gesichert; d.h. er bleibt erhalten auch wenn es von anderen Fenstern überlappt wurde. Dies ist jedoch nicht in allen Fällen notwendig. Wenn der Inhalt eines Fensters ständig neu aufgebaut wird (copy_rectangle), so kann für dieses der Sicherungsmechanismus abgestellt und damit der hierfür nötige Speicher eingespart werden. Dies geschieht mit dem Aufruf von set_system (::'backing_store','false':) vor dem Öffnen des Fensters. Durch dieses Vorgehen wird nicht nur Speicher, sondern auch Rechenzeit gespart, was beispielsweise für die Ausgabe von Videoclips von Bedeutung ist (siehe copy_rectangle).

Für die Ausgabe von Grafik (disp_image,disp_region, etc.) kann mit dem Operator set_part das Fenster so eingestellt werden, dass es einen logischen Ausschnitt des Bildformates darstellt. Dies hat insbesondere zur Folge, dass von Bildern und Regionen lediglich dieser Ausschnitt (entsprechend vergrößert) ausgegeben wird.

Unterschied: Grafikfenster - Textfenster

- Bei Grafikfenstern ist das Layout nicht so variabel handhabbar wie bei Textfenstern.
- Nur Textfenster können zum Einlesen von Benutzerdaten verwendet werden (read_string).
- Bei der Ausgabe von Bildern, Regionen und Grafiken wird bei Grafikfenstern „gezoomt“: Unabhängig von Größe und Seitenverhältnis des Fensters werden die Bilder so transformiert, dass sie füllend in dem Fenster ausgegeben werden. Bei Textfenstern dagegen wird die Größe des Fensters bei der Ausgabe nicht beachtet (nur falls Clipping nötig ist).
- Bei Grafikfenstern ist das Koordinatensystem des Fensters immer gleich dem Koordinatensystem des Bildformates, während bei Textfenstern das Koordinatensystem unabhängig von der Bildgröße immer den Bildschirmkoordinaten entspricht.

Der Parameter Mode legt den Modus des Fensters fest. Dieser kann folgende Werte annehmen:

'visible': Dies ist der normale Modus für Grafikfenster: Das Fenster wird entsprechend den Parametern erzeugt und alle Ein- und Ausgaben sind möglich.

'invisible': Unsichtbare Fenster werden nicht auf dem Bildschirm dargestellt. Die Parameter Row, Column und FatherWindow sind ohne Bedeutung. Ausgaben auf diese Fenster haben keine Wirkung. Eingaben (read_string, Maus, etc.) sind nicht möglich. Diese Fenster werden verwendet, wenn man Darstellungsparameter für ein Ausgabegerät abfragen möchte, ohne ein (sichtbares) Fenster zu öffnen. Übliche Anfragen sind z.B. query_color oder get_string_extents.

'transparent': Diese Fenster sind durchsichtig: Das Fenster selbst ist nicht sichtbar (Rand und Hintergrund), jedoch sind alle sonstigen Operationen möglich und alle Ausgaben werden dargestellt. Eine übliche Verwendung für diesen Modus ist die Erzeugung von Maus-sensitiven Bereichen.

'buffer': Dies sind ebenfalls nicht sichtbare Fenster. Die Ausgabe von Bildern, Regionen und Grafiken ist auf dem Bildschirm nicht sichtbar, wird jedoch im Speicher gepuffert. Die Parameter Row, Column und FatherWindow sind ohne Bedeutung. Puffer-Fenster verwendet man beispielsweise, wenn man Ausgaben (im Hintergrund) vorbereitet und schließlich mit copy_rectangle in ein sichtbares Fenster kopiert. Eine andere Anwendung wäre die schnelle Verarbeitung von Bildausschnitten bei interaktiven Manipulationen. Texteingaben und Mausinteraktion ist mit dem Modus 'buffer' nicht möglich.

Achtung

Es ist zu beachten, dass die Parameter Row, Column, Width und Height durch das Ausgabegerät beschränkt sind. Wird ein Vaterfenster (FatherWindow <> 'root') angegeben, dann sind die Koordinaten relativ zu diesem Fenster.

Parameter

▷ **Row** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilenindex der linken oberen Ecke.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ Row (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Row >= 0

▷ **Column** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltenindex der linken oberen Ecke.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ Column (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Column >= 0

▷ **Width** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
Breite des Fensters.
Defaultwert: 256
Typischer Wertebereich: 0 ≤ Width (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: 0 < Width <= 32768 || Width == -1

▷ **Height** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
Höhe des Fensters.
Defaultwert: 256
Typischer Wertebereich: 0 ≤ Height (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: 0 < Height <= 32768 || Height == -1

▷ **FatherWindow** (input_control) . . . . . pointer ⇝ *integer* / string
Logische Nummer des Vaterfensters. Für den Bildschirm als Vater kann 'root' oder 0 eingegeben werden.
Defaultwert: 0
Restriktion: FatherWindow >= 0

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Fenstermodus.
Defaultwert: 'visible'
Werteliste: `Mode` ∈ {'visible', 'invisible', 'transparent', 'buffer'}

▷ **Machine** (input_control) . . . . . string ⇝ *string*
Name des Rechners, auf dem das Fenster geöffnet werden soll oder leerer String.
Defaultwert: ''

▷ **WindowHandle** (output_control) . . . . . window ⇝ *handle*
Fenster-Handle.

Beispiel

```
open_window(0,0,400,-1,'root','visible','',WindowHandle)
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
write_string(WindowHandle,'File, fabrik')
new_line(WindowHandle)
get_mbutton(WindowHandle,_,_,_)
set_lut(WindowHandle,'temperature')
set_color(WindowHandle,'blue')
write_string(WindowHandle,'temperature')
new_line(WindowHandle)
write_string(WindowHandle,'Draw Rectangle')
new_line(WindowHandle)
draw_rectangle1(WindowHandle,Row1,Column1,Row2,Column2)
```

```
set_part(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
new_line(WindowHandle)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `open_window` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`reset_obj_db`

Nachfolger

`set_color`, `query_window_type`, `get_window_type`, `set_window_type`, `get_mposition`, `set_tposition`, `set_tshape`, `set_window_extents`, `get_window_extents`, `query_color`, `set_check`, `set_system`

Siehe auch

`disp_region`, `disp_image`, `disp_color`, `set_lut`, `query_color`, `set_color`, `set_rgb`, `set_hsi`, `set_pixel`, `set_gray`, `set_part`, `set_part_style`, `query_window_type`, `get_window_type`, `set_window_type`, `get_mposition`, `set_tposition`, `set_window_extents`, `get_window_extents`, `set_window_attr`, `set_check`, `set_system`

Modul

Foundation

query_window_type (: : : WindowTypes)

Abfragen der verfügbaren Fenstertypen.

`query_window_type` gibt ein Tupel zurück, das alle Geräte bzw. Softwaresysteme enthält, mit denen Bildobjekte dargestellt werden können. `query_window_type` kann dort sinnvoll eingesetzt werden, wo man maschinenunabhängige Programme entwickeln möchte. Mögliche Werte sind:

'X-Window' X-Window Version 11.

'WIN32-Window' Microsoft Windows.

'Cocoa' macOS.

'pixmap' Fenster werden nicht angezeigt, sondern nur im Speicher verwaltet. Auf diese Weise können HALCON-Programme auch auf Rechnern ohne Grafikbildschirm portiert werden.

'PostScript' Die Ausgabe von Objekten erfolgt in eine PostScript-Datei.

Parameter

▷ **WindowTypes** (output_control) . . . . . string-array ⇝ *string*
Namen der verfügbaren Fenstertypen.

Ergebnis

`query_window_type` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

reset_obj_db

Modul

Foundation

set_window_attr (: : AttributeName, AttributeValue :)

Setzen von Fenstereigenschaften.

set_window_attr dient zum Setzen von bestimmten Eigenschaften von Grafikfenstern. Damit können folgende Defaultparameter eines Fenster verändert werden:

'border_width' Breite des Fensterrandes in Pixel. Nicht implementiert unter Windows.

'border_color' Farbe des Fensterrandes. Nicht implementiert unter Windows.

'background_color' Hintergrundfarbe des Fensters.

'window_title' Name des Fensters in der Titelleiste.

Achtung

set_window_attr muss *vor* dem Aufruf von open_window aufgerufen werden.

Parameter

▷ **AttributeName** (input_control) . . . . . string ⇝ *string*
Name des Attributs, das verändert werden soll.
Werteliste: AttributeName ∈ {'border_width', 'border_color', 'background_color', 'window_title'}

▷ **AttributeValue** (input_control) . . . . . string ⇝ *string* / integer
Attributwert, der gesetzt werden soll.
Werteliste: AttributeValue ∈ {0, 1, 2, 'white', 'black', 'MyName', 'default'}

Ergebnis

set_window_attr liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_draw, set_color, set_colored, set_line_width

Siehe auch

open_window, get_window_attr, set_window_param

Modul

Foundation

set_window_dc (: : WindowHandle, WINHDC :)

Setzen des Grafikkontextes eines virtuellen Grafikfensters (Windows NT).

set_window_dc setzt den Gerätekontext eines zuvor mit new_extern_window angelegten Fensters. Alle Ausgaben (disp_region, disp_image, etc.) werden auf dem zu diesen Gerätekontext gehörenden Fensters getätigt.

Im Parameter WINHDC wird der Gerätekontext des Windows-Fensters, das zur Ausgabe der Daten verwendet werden soll, übergeben. Dieser Gerätekontext wird in den Ausgaberoutinen von HALCON verwendet.

Achtung

Es ist zu beachten, dass das Fenster WindowHandle vorher mit new_extern_window angelegt worden sein muss.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **WINHDC** (input_control) ..... pointer ⇝ *integer*
  Gerätekontext des WINHWnd
  Restriktion: WINHDC != 0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_window_dc den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

new_extern_window

Nachfolger

disp_image, disp_region

Siehe auch

new_extern_window, disp_region, disp_image, disp_color, set_lut, query_color, set_color, set_rgb, set_hsi, set_pixel, set_gray, set_part, set_part_style, query_window_type, get_window_type, set_window_type, get_mposition, set_tposition, set_window_extents, get_window_extents, set_window_attr, set_check, set_system

Modul

Foundation

set_window_extents (: : WindowHandle, Row, Column, Width, Height :)

Verändern der Position und Größe eines Fensters.

set_window_extents positioniert die linke obere Ecke des Ausgabefensters auf (Row,Column) und verändert gleichzeitig die Größe des Fensters auf Width und Height. Negative Werte für Width und Height werden ignoriert.

Wird die Fenstergröße an die Größe des anzuzeigenden Bildteils angepasst, verhindert dies eine Verlangsamung der Anzeige durch notwendige Interpolationen. Daher ist das Fenster vorzugsweise gleich groß wie der anzuzeigende Bildteil, oder ansonsten halb so groß, ein Viertel so groß, etc.

Achtung

Wird die Größe des Fensters verändert, so wird der Fensterinhalt nicht automatisch neu dargestellt. Dies muss vom Programm aus vorgenommen werden, indem die Daten erneut ausgegeben werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.

▷ **Row** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilenindex der linken oberen Ecke der Zielposition.
Defaultwert: 0
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Column** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltenindex der linken oberen Ecke der Zielposition.
Defaultwert: 0
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Width** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
Breite des Fensters.
Defaultwert: 512
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Height** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
Höhe des Fensters.
Defaultwert: 512
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Ergebnis

`set_window_extents` liefert den Wert TRUE, falls das Fenster gültig ist und die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Siehe auch

`get_window_extents`, `open_window`

Modul

Foundation

set_window_type (: : WindowType :)

Festlegen des Fenstertyps.

`set_window_type` bestimmt, auf welchen Typ von Ausgabegerät die Ausgaben erfolgen sollen. Diese Einstellung wird von dem Operator `open_window` beim Öffnen der Fenster verwendet. Verschiedene Fenster können auch auf verschieden Typen von Ausgabegeräten geöffnet werden. Dazu stellt man vor dem Öffnen jeweils den gewünschten Typ ein. Die verfügbaren Typen von Ausgabegeräten können über den Operator `query_window_type` abgefragt werden. Mögliche Werte sind:

'X-Window' X-Window Version 11.

'WIN32-Window' Microsoft Windows.

'Cocoa' macOS.

'pixmap' Fenster werden nicht angezeigt, sondern nur im Speicher verwaltet. Auf diese Weise können HALCON-Programme auch auf Rechnern ohne Grafikbildschirm portiert werden.

'PostScript' Die Ausgabe von Objekten erfolgt in eine PostScript-Datei.

'WIN32-Window' Microsoft Windows.

'system_default' Standardeinstellung für die aktuelle Plattform.

`set_window_type` kann dort sinnvoll eingesetzt werden, wo man maschinenunabhängige Programme entwickeln möchte.

Parameter

▷ **`WindowType`** (input_control) . . . . . string ⇝ *string*
Namen des Fenstertyps, der gesetzt werden soll.
Defaultwert: 'X-Window'
Werteliste: `WindowType` ∈ {'X-Window', 'WIN32-Window', 'Cocoa', 'pixmap', 'PostScript', 'system_default'}

Ergebnis

`set_window_type` liefert den Wert TRUE, falls der Ausgabegerätetyp verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Siehe auch

`open_window`, `query_window_type`, `get_window_type`

Modul

Foundation

`unproject_coordinates` `( Image : : WindowHandle, Row, Column : ImageRow, ImageColumn, Height )`

Zurückliefern der Bildkoordinaten eines Punktes in einem 3D-Plot-Fenster

Wird `Image` als 3D-Plot (d.h. *'3d_plot'* wurde mittel `set_paint` ausgewählt) im Fenster `WindowHandle` angezeigt, berechnet `unproject_coordinates` die Bildkoordinaten `ImageRow` und `ImageColumn`, sowie die Höhe `Height` für einen gegebenen Punkt `Row`,`Column` in Fensterkoordinaten. Die Fensterkoordinaten `Row` und `Column` müssen so angegeben werden, dass sie dem gesetzten Bildausschnitt entsprechen und damit auch subpixelgenau sind. Ist der Bildausschnitt durch (Zeile1,Spalte1,Zeile2, Spalte2) gegeben, entspricht die linke obere Ecke den Koordinaten (Zeile1 - 0.5, Spalte1 - 0.5), die rechte untere Ecke den Koordinaten (Zeile2 - 0.5, Spalte2 - 0.5). Die Operatoren `get_mposition_sub_pix` oder `get_mbutton_sub_pix` können verwendet werden, um diese Koordinaten direkt zu erhalten.

Einer der Fensterparameter *'save_depth_buffer'* oder *'interactive_plot'* muss mittels `set_window_param` auf *'true'* gesetzt werden, damit `unproject_coordinates` benutzt werden kann.

Parameter

▷ **`Image`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real / complex / vector_field
Ausgegebenes Bild.

▷ **`WindowHandle`** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **`Row`** (input_control) . . . . . integer ⇝ *real* / integer
Zeilenkoordinate im Fenster.

▷ **`Column`** (input_control) . . . . . integer ⇝ *real* / integer
Spaltenkoordinate im Fenster.

▷ **ImageRow** (output_control) . . . . . integer ⇝ *integer*
Zeilenkoordinate im Bild.

▷ **ImageColumn** (output_control) . . . . . integer ⇝ *integer*
Spaltenkoordinate im Bild.

▷ **Height** (output_control) . . . . . real ⇝ *integer* / real
Höhenwert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_mbutton_sub_pix`, `get_mposition_sub_pix`, `disp_image`

Siehe auch

`disp_obj`, `update_window_pose`, `set_paint`

Modul

Foundation

update_window_pose (: : WindowHandle, LastRow, LastCol, CurrentRow, CurrentCol, Mode :)

Verändern der Pose eines 3D-Plots.

Der Operator `update_window_pose` vereinfacht das Setzen der Orientierung und der Lage des Höhenmodells, das vom Operator `disp_obj` angezeigt wird, sofern der Darstellungsmodus mittels `set_paint` auf *'3d_plot'* gesetzt wurde.

Zwei Fensterposition (`LastRow`,`LastCol`) und (`CurrentRow`, `CurrentCol`) werden in Rotation, Skalierung oder Bewegung umgewandelt, abhängig von `Mode`. Diese Fensterkoordinaten müssen so angegeben werden, dass sie dem gesetzten Bildausschnitt entsprechen und damit auch subpixelgenau sind. Ist der Bildausschnitt durch (Zeile1,Spalte1,Zeile2,Spalte2) gegeben, entspricht die linke obere Ecke der Koordinate (Zeile1 - 0.5, Spalte1 - 0.5), die rechte untere Ecke der Koordinate (Zeile2 - 0.5, Spalte2 - 0.5). Die Operatoren `get_mposition_sub_pix` oder `get_mbutton_sub_pix` können verwendet werden, um diese Koordinaten direkt zu erhalten.

Wird *'rotate'* als `Mode` gewählt, wird das Höhenmodell rotiert. Dabei werden beide Punkte auf eine virtuelle Kugel, die im Zentrum des Fensters ihren Mittelpunkt hat, projiziert. Der Bogen zwischen beiden Projektionen entspricht der Rotation, die auf das Höhenmodell angewandt wird.

Wird *'scale'* als `Mode` gewählt, wird die Distanz zwischen dem Betrachter und dem Höhenmodell verringert, wenn `CurrentRow` größer ist als `LastRow`. Falls `CurrentRow` kleiner ist als `LastRow`, wird die Distanz vergrößert.

Wird *'move'* als `Mode` gewählt, werden die Eingabepunkte auf eine Ebene, die parallel zur Betrachterebene durch die Mitte des Höhenmodells geht, projiziert. Das Zentrum des Höhenmodells wird um die Distanz zwischen beiden Projektionen bewegt.

Wird *'move_plane'* als `Mode` gewählt, werden die Eingabepunkte auf die Ebene projiziert, die der Höhe Null entspricht. Das Höhenmodell wird um die Distanz zwischen beiden Projektionen bewegt.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **LastRow** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des ersten Punktes

▷ **LastCol** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des ersten Punktes.

▷ **CurrentRow** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des zweiten Punktes.

▷ **CurrentCol** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des zweiten Punktes.

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Navigationsmodus.
Defaultwert: 'rotate'
Werteliste: Mode ∈ {'rotate', 'scale', 'move', 'move_plane'}

Beispiel

```
* Interactive display of a height field
dev_set_paint ('3d_plot')
while (1)
  dev_set_check ('~give_error')
  get_mposition_sub_pix (WindowHandle, Row, Column, Button)
  dev_set_check ('give_error')
  if (ButtonDown and (Button == 0))
    ButtonDown := false
  endif
  if (not(Button == 0))
    if (ButtonDown)
      if (Button == 1)
        mode := 'rotate'
      endif
      if (Button == 4)
        mode := 'scale'
      endif
      if (Button == 5)
        mode := 'move'
      endif
      update_window_pose (WindowHandle, lastRow, lastCol, Row, Column, mode)
    else
      if (Button == 2)
        break
      endif
      ButtonDown := true
    endif
    lastCol := Column
    lastRow := Row
  endif
  dev_display (Image)
endwhile
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_paint, open_window, get_mposition_sub_pix, get_mbutton_sub_pix

Nachfolger

disp_image

Siehe auch

unproject_coordinates

Modul

Foundation

11.5 Maus

get_mbutton (: : WindowHandle : Row, Column, Button)

Abfragen der Mausposition bei Tastendruck.

get_mbutton liefert die pixelgenauen Bildkoordinaten der Maus im Ausgabefenster und die dabei betätigten Tasten. Zusätzlich wird der Zustand der Tasten Shift, Strg und Alt zurückgegeben. Die folgenden Werte sind den einzelnen Tasten zugeordnet:

1: Linke Taste,

2: Mittlere Taste,

4: Rechte Taste,

8: Shift Taste,

16: Strg Taste,

32: Alt Taste.

Die Summe der Werte aller gedrückten Tasten wird in Button zurückgegeben.

Es wird so lange gewartet, bis eine der Tasten in dem angegebenen HALCON-Fenster gedrückt wird. Im Falle eines Grafikfensters berücksichtigen die Koordinaten Row und Column den aktuellen Bildausschnitt (siehe set_part).

Falls subpixelgenaue Bildkoordinaten benötigt werden, kann der Operator get_mbutton_sub_pix verwendet werden.

Achtung

get_mbutton kommt erst zurück, wenn mit der Maus in das Fenster geklickt wird.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) ..... point.y ⇝ *integer*
  Zeilenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Column** (output_control) ..... point.x ⇝ *integer*
  Spaltenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Button** (output_control) ..... integer ⇝ *integer*
  Betätigte Taste(n).

Ergebnis

get_mbutton liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Alternativen

get_mposition, get_mbutton_sub_pix, get_mposition_sub_pix

Siehe auch

open_window

Modul

Foundation

get_mbutton_sub_pix (: : WindowHandle : Row, Column, Button)

Abfragen der subpixelgenauen Mausposition bei Tastendruck.

get_mbutton_sub_pix liefert die subpixelgenauen Bildkoordinaten der Maus im Ausgabefenster und die dabei betätigten Tasten. Zusätzlich wird der Zustand der Tasten Shift, Strg und Alt zurückgegeben. Die folgenden Werte sind den einzelnen Tasten zugeordnet:

1: Linke Taste,

2: Mittlere Taste,

4: Rechte Taste,

8: Shift Taste,

16: Strg Taste,

32: Alt Taste.

Die Summe der Werte aller gedrückten Tasten wird in Button zurückgegeben.

Es wird so lange gewartet, bis eine der Tasten in dem angegebenen HALCON-Fenster gedrückt wird. Im Falle eines Grafikfensters berücksichtigen die Koordinaten Row und Column den aktuellen Bildausschnitt (siehe set_part).

Falls nur pixelgenaue Bildkoordinaten benötigt werden, kann der Operator get_mbutton verwendet werden.

Achtung

get_mbutton_sub_pix kommt erst zurück, wenn mit der Maus in das Fenster geklickt wird.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Button** (output_control) ..... integer ⇝ *integer*
  Betätigte Taste(n).

Ergebnis

get_mbutton_sub_pix liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Alternativen

get_mbutton, get_mposition, get_mposition_sub_pix

Siehe auch

open_window

Modul

Foundation

get_mposition (: : WindowHandle : Row, Column, Button)

Abfragen der Mausposition.

get_mposition liefert die pixelgenauen Bildkoordinaten der Maus im Ausgabefenster und den Code für die betätigten Tasten. Die Daten werden unabhängig vom Zustand der Tasten (betätigt oder nicht betätigt) abgeliefert. Wurde mindestens eine Maustaste gedrückt, wird auch der Zustand der Tasten Shift, Strg und Alt zurückgegeben. Wurde keine Maustaste gedrückt, wird auf jeden Fall 0 in Button zurückgegeben. Die folgenden Werte sind den einzelnen Tasten zugeordnet:

0: Keine Taste,

1: Linke Taste,

2: Mittlere Taste,

4: Rechte Taste,

8: Shift Taste,

16: Strg Taste,

32: Alt Taste.

Die Summe der Werte aller gedrückten Tasten wird in Button zurückgegeben.

Im Falle eines Grafikfensters berücksichtigen die Koordinaten Row und Column den aktuellen Bildausschnitt (siehe set_part).

Falls subpixelgenaue Bildkoordinaten benötigt werden, kann der Operator get_mposition_sub_pix verwendet werden.

Achtung

get_mposition scheitert (FAIL), falls sich die Maus nicht innerhalb des Fensters befindet. In diesem Fall werden in den drei Parametern keine Werte übergeben.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) . . . . . point.y ⇝ *integer*
  Zeilenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Column** (output_control) . . . . . point.x ⇝ *integer*
  Spaltenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Button** (output_control) . . . . . integer ⇝ *integer*
  Betätigte Taste(n) oder 0.

Ergebnis

get_mposition liefert den Wert TRUE. Befindet sich der Cursor außerhalb des Fensters, wird 5 (H_MSG_FAIL) ausgegeben.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Alternativen

get_mbutton, get_mposition_sub_pix, get_mbutton_sub_pix

Siehe auch

open_window

Modul

Foundation

get_mposition_sub_pix (: : WindowHandle : Row, Column, Button)

Abfragen der subpixelgenauen Mausposition.

get_mposition_sub_pix liefert die subpixelgenauen Bildoordinaten der Maus im Ausgabefenster und den Code für die betätigten Tasten. Die Daten werden unabhängig vom Zustand der Tasten (betätigt oder nicht betätigt) abgeliefert. Wurde mindestens eine Maustaste gedrückt, wird auch der Zustand der Tasten Shift, Strg und Alt zurückgegeben. Wurde keine Maustaste gedrückt, wird auf jeden Fall 0 in Button zurückgegeben. Die folgenden Werte sind den einzelnen Tasten zugeordnet:

0: Keine Taste,

1: Linke Taste,

2: Mittlere Taste,

4: Rechte Taste,

8: Shift Taste,

16: Strg Taste,

32: Alt Taste.

Die Summe der Werte aller gedrückten Tasten wird in Button zurückgegeben.

Im Falle eines Grafikfensters berücksichtigen die Koordinaten Row und Column den aktuellen Bildausschnitt (siehe set_part).

Falls nur pixelgenaue Bildkoordinaten benötigt werden, kann der Operator get_mposition verwendet werden.

Achtung

get_mposition_sub_pix scheitert (FAIL), falls sich die Maus nicht innerhalb des Fensters befindet. In diesem Fall werden in den drei Parametern keine Werte übergeben.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Button** (output_control) ..... integer ⇝ *integer*
  Betätigte Taste(n) oder 0.

Ergebnis

get_mposition_sub_pix liefert den Wert TRUE. Befindet sich der Cursor außerhalb des Fensters, wird 5 (H_MSG_FAIL) ausgegeben.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Alternativen

get_mbutton, get_mposition, get_mbutton_sub_pix

Siehe auch

open_window

Modul

Foundation

get_mshape (: : WindowHandle : Cursor)

Abfragen der Mauscursor Form.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit `dev_open_window` geöffnet wurden.

`get_mshape` gibt für das Ausgabefenster den Namen des momentan verwendeten Mauscursors aus. Gesetzt wird ein neuer Mauscursor mit dem Operator `set_mshape`. Unter einem Mauscursor (im Gegensatz zum Textcursor; beide werden einfach als `Cursor` bezeichnet, wenn Missverständnisse ausgeschlossen sind) wird hier das Muster verstanden, das synchron zur Maus auf dem Bildschirm dargestellt wird.

Parameter

▷ **`WindowHandle`** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **`Cursor`** (output_control) . . . . . string ⇝ *string*
Name der Cursorform.

Ergebnis

`get_mshape` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `query_mshape`

Nachfolger

`set_mshape`

Siehe auch

`set_mshape`, `query_mshape`

Modul

Foundation

`query_mshape` (: : `WindowHandle` : `ShapeNames`)

Abfragen aller Formen für den Maus-Cursor.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit `dev_open_window` geöffnet wurden.

`query_mshape` gibt für das Ausgabefenster die Namen aller Mauscursormuster aus. Diese können von dem Operator `set_mshape` verwendet werden. Unter einem Mauscursor (im Gegensatz zu Textcursor; beide werden einfach als Cursor bezeichnet, wenn Missverständnisse ausgeschlossen sind) wird hier das Muster verstanden, das bei der Bewegung der Maus auf dem Bildschirm erscheint. Ist keine Maus vorhanden, dann wird das leere Tupel als Ergebnis abgeliefert.

Parameter

▷ **`WindowHandle`** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **`ShapeNames`** (output_control) . . . . . string-array ⇝ *string*
Namen der Cursor.

Ergebnis

`query_mshape` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, get_mshape

Nachfolger

set_mshape

Siehe auch

set_mshape, get_mshape

Modul

Foundation

send_mouse_double_click_event (: : WindowHandle, Row, Column, Button : Processed)

Senden eines Events, welches das doppelte Klicken eines Mausbuttons signalisiert an einem Pufferfenster.

send_mouse_double_click_event sendet dem Pufferfenster WindowHandle ein Event, das signalisiert, dass der durch Button kodiert Mausbutton an der Position (Row, Column) doppelt geklickt wurde. In der Dokumentation von get_mbutton wird beschrieben wie Button kodiert werden muss.

Mit diesem Operator können Zeichenobjekte in Konfigurationen bei denen mehrere Objekte übereinander liegen selektiert werden, wenn der Doppelklick auf einen der Marker oder auf dem Rand des aktiven Objektes geht. Um die Interaktion mit Zeichenobjekten zu vereinfachen, sind die Koordinaten Row und Column im Bildkoordinatensystem gegeben. Um Fenster- in Bildkoordinaten umzuwandeln kann der Operator convert_coordinates_window_to_image verwendet werden.

Sofern das Event verarbeitet werden konnte, gibt send_mouse_double_click_event *'true'* in Processed zurück. Falls keine Aktion aus dem Event abgeleitet werden konnte (weil sich zum Beispiel kein Zeichenobjekt an der Mausposition befindet), wird *'false'* zurückgegeben. In diesem Fall kann der Aufrufer das Event behandeln.

Die Operatoren send_mouse_down_event, send_mouse_up_event, send_mouse_drag_event und send_mouse_double_click_event sind die einzige Möglichkeit Zeichenobjekte in Pufferfenstern zu manipulieren.

Achtung

send_mouse_double_click_event benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle des Pufferfensters.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer* / real
  Zeilenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Column** (input_control) ..... point.x ⇝ *integer* / real
  Spaltenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Button** (input_control) ..... integer ⇝ *integer*
  Gedrückte Mausbuttons.
- ▷ **Processed** (output_control) ..... string ⇝ *string*
  'true', wenn das Event verarbeitet werden konnte.

Ergebnis

send_mouse_double_click_event liefert den Wert TRUE, falls WindowHandle ein Pufferfenster ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Siehe auch

send_mouse_up_event, send_mouse_down_event, open_window

Modul

Foundation

send_mouse_down_event (: : WindowHandle, Row, Column, Button : Processed)

Senden eines Events, welches das Drücken eines Mausbuttons signalisiert an einem Pufferfenster.

send_mouse_down_event sendet dem Pufferfenster WindowHandle ein Event, das signalisiert, dass der durch Button kodiert Mausbutton an der Position (Row, Column) gedrückt wurde. In der Dokumentation von get_mbutton wird beschrieben wie Button kodiert werden muss.

Mit diesem Operator können Zeichenobjekte manipuliert werden. Zeigen die Koordinaten Row und Column auf ein Zeichenobjekt, so wird das Zeichenobjekt aktiviert und kann durch senden weiterer Events mit send_mouse_down_event bewegt oder verändert werden. Um die Interaktion mit Zeichenobjekten zu vereinfachen, sind die Koordinaten Row und Column im Bildkoordinatensystem gegeben. Um Fenster- in Bildkoordinaten umzuwandeln kann der Operator convert_coordinates_window_to_image verwendet werden.

Sofern das Event verarbeitet werden konnte, gibt send_mouse_down_event *'true'* in Processed zurück. Falls keine Aktion aus dem Event abgeleitet werden konnte (weil sich zum Beispiel kein Zeichenobjekt an der Mausposition befindet), wird *'false'* zurückgegeben. In diesem Fall kann der Aufrufer das Event behandeln.

Die Operatoren send_mouse_down_event, send_mouse_up_event, send_mouse_drag_event und send_mouse_double_click_event sind die einzige Möglichkeit Zeichenobjekte in Pufferfenstern zu manipulieren.

Achtung

send_mouse_down_event benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle des Pufferfensters.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer* / real
  Zeilenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Column** (input_control) ..... point.x ⇝ *integer* / real
  Spaltenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Button** (input_control) ..... integer ⇝ *integer*
  Gedrückte Mausbuttons.
- ▷ **Processed** (output_control) ..... string ⇝ *string*
  'true', wenn das Event verarbeitet werden konnte.

Ergebnis

send_mouse_down_event liefert den Wert TRUE, falls WindowHandle ein Pufferfenster ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Siehe auch

send_mouse_up_event, send_mouse_double_click_event, open_window

Modul

Foundation

send_mouse_drag_event (: : WindowHandle, Row, Column, Button : Processed)

Senden eines Events, welches das Ziehen eines Mausbuttons signalisiert an einem Pufferfenster.

send_mouse_drag_event sendet dem Pufferfenster WindowHandle ein Event, das signalisiert, dass der durch Button kodiert Mauszeiger an der Position (Row, Column) gezogen wurde. In der Dokumentation von get_mbutton wird beschrieben wie Button kodiert werden muss.

Sofern das Event verarbeitet werden konnte, gibt send_mouse_drag_event *'true'* in Processed zurück. Falls keine Aktion aus dem Event abgeleitet werden konnte (weil sich zum Beispiel kein Zeichenobjekt an der Mausposition befindet), wird *'false'* zurückgegeben. In diesem Fall kann der Aufrufer das Event behandeln.

Die Operatoren send_mouse_down_event, send_mouse_up_event, send_mouse_drag_event und send_mouse_double_click_event sind die einzige Möglichkeit Zeichenobjekte in Pufferfenstern zu manipulieren.

Achtung

send_mouse_drag_event benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle des Pufferfensters.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer* / real
  Zeilenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Column** (input_control) ..... point.x ⇝ *integer* / real
  Spaltenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Button** (input_control) ..... integer ⇝ *integer*
  Gedrückte Mausbuttons.
- ▷ **Processed** (output_control) ..... string ⇝ *string*
  'true', wenn das Event verarbeitet werden konnte.

Ergebnis

send_mouse_drag_event liefert den Wert TRUE, falls WindowHandle ein Pufferfenster ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Siehe auch

send_mouse_up_event, send_mouse_down_event, send_mouse_double_click_event, open_window

Modul

Foundation

send_mouse_up_event (: : WindowHandle, Row, Column, Button : Processed)

Senden eines Events, welches das Loslassen eines Mausbuttons signalisiert an einem Pufferfenster.

send_mouse_up_event sendet dem Pufferfenster WindowHandle ein Event, das signalisiert, dass der durch Button kodiert Mausbutton an der Position (Row, Column) losgelassen wurde. In der Dokumentation von get_mbutton wird beschrieben wie Button kodiert werden muss.

Mit diesem Operator können Zeichenobjekte losgelassen werden. So wird zum Beispiel ein Zeichenobjekt, dass durch eine Folge von send_mouse_down_event bewegt wurde, nach dem Erhalt eines send_mouse_up_event freigegeben. Um die Interaktion mit Zeichenobjekten zu vereinfachen, sind die Koordinaten Row und Column im Bildkoordinatensystem gegeben. Um Fenster- in Bildkoordinaten umzuwandeln kann der Operator convert_coordinates_window_to_image verwendet werden.

Sofern das Event verarbeitet werden konnte, gibt send_mouse_up_event *'true'* in Processed zurück. Falls keine Aktion aus dem Event abgeleitet werden konnte (weil sich zum Beispiel kein Zeichenobjekt an der Mausposition befindet), wird *'false'* zurückgegeben. In diesem Fall kann der Aufrufer das Event behandeln.

Die Operatoren send_mouse_down_event, send_mouse_up_event, send_mouse_drag_event und send_mouse_double_click_event sind die einzige Möglichkeit Zeichenobjekte in Pufferfenstern zu manipulieren.

Achtung

send_mouse_up_event benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle des Pufferfensters.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer* / real
  Zeilenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Column** (input_control) ..... point.x ⇝ *integer* / real
  Spaltenkoordinate des Mauscursors im Bildkoordinatensystem.
- ▷ **Button** (input_control) ..... integer ⇝ *integer*
  Gedrückte Mausbuttons.
- ▷ **Processed** (output_control) ..... string ⇝ *string*
  'true', wenn das Event verarbeitet werden konnte.

Ergebnis

send_mouse_up_event liefert den Wert TRUE, falls WindowHandle ein Pufferfenster ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Siehe auch

send_mouse_down_event, send_mouse_double_click_event, open_window

Modul

Foundation

set_mshape (: : WindowHandle, Cursor :)

Setzen der Mauscursor Form.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit dev_open_window geöffnet wurden.

set_mshape setzt das Muster, das für den Mauscursor des Ausgabefensters verwendet werden soll. Unter Mauscursor (im Gegensatz zu Textcursor; beide werden einfach als Cursor bezeichnet, wenn Missverständnisse ausgeschlossen sind) wird hier das Muster verstanden, das auf dem Bildschirm bei einer Bewegung der Maus dargestellt wird. Ein Tupel aller verfügbaren Muster kann mit dem Operator query_mshape abgefragt werden. Der

Cursor ist dem Fenster im weiteren zugeordnet (d.h. das Muster wird bei Eintritt in das Fenster verwendet) und zwar unabhängig davon, welches Fenster die aktuellen Ausgaben verarbeitet.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Cursor** (input_control) . . . . . string ⇝ *string*
Name des Mauscursors.
Defaultwert: 'arrow'

Ergebnis

set_mshape liefert den Wert TRUE, falls für das Fenster der angegebene Cursor definiert ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, query_mshape, get_mshape

Siehe auch

get_mshape, query_mshape

Modul

Foundation

11.6 Objekt

attach_background_to_window (Image : : WindowHandle :)

Setzen des Hintergrundbildes eines HALCON-Fensters.

attach_background_to_window setzt das Hintergrundbild Image des HALCON-Fensters WindowHandle. Das Eingabebild wird kopiert, so dass es sicher wieder gelöscht werden kann.

Das Hintergrundbild wird sofort im HALCON-Fenster angezeigt, wenn attach_background_to_window aufgerufen wird. Alle HALCON-Objekte, die bisher in diesem Fenster angezeigt wurden, werden auch weiterhin dargestellt. Somit wird der bisherige Fensterinhalt nicht überschrieben.

Der Fensterinhalt wird in der folgenden Reihenfolge dargestellt: Zuerst das Hintergrundbild, dann HALCON-Objekte und anschließend die Zeichenobjekte. Somit wird ein Zeichenobjekt immer an der obersten Ebene dargestellt. Das Fenster wird für jede Benutzerinteraktion innerhalb des Fensters und jeden Aufruf von disp_obj aktualisiert. HALCON-Objekte können durch einen Aufruf von clear_window entfernt werden.

Das Hintergrundbild kann mit dem Operator detach_background_from_window wieder von dem Fenster entfernt werden.

Achtung

Es sollte kein Operator, der auf die Event-Queue des HALCON-Fensters zugreift, wie z.B. get_mbutton oder read_char, verwendet werden, solange Zeichenobjekte ans Fenster angehängt sind. Sie stehen im Gegensatz zu der Interaktion mit Zeichenobjekten, da sie aktiv auf die Eventloop des Fensters zugreifen.

Außerdem sollte keiner der blockierenden Operatoren draw_rectangle1, draw_rectangle2, draw_region, draw_xld oder draw_circle zusammen mit der Event-Funktionalität verwendet werden, da sie die Interaktion mit Zeichenobjekten verhindern.

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field

Hintergrundbild.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*

Fenster-Handle.

Ergebnis

`attach_background_to_window` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`detach_background_from_window`

Modul

Foundation

attach_drawing_object_to_window (: : WindowHandle, DrawHandle :)

Anhängen eines bestehenden Zeichenobjektes an ein HALCON-Fenster.

`attach_drawing_object_to_window` hängt ein bestehendes Zeichenobjekt `DrawHandle` an das HALCON-Fenster `WindowHandle` an. Sobald das Zeichenobjekt an das Fenster angehängt wurde, wird es in dem jeweiligen Fenster angezeigt. Das Zeichenobjekt kann im HALCON-Fenster interaktiv modifiziert werden. Das Fenster wird automatisch aktualisiert, wenn die Eigenschaften des Objekts modifiziert werden (siehe `set_drawing_object_params`).

Wenn sich zwei oder mehrere Zeichenobjekte überlappen, ist es durch einen Doppelklick auf das derzeit ausgewählte Zeichenobjekt möglich, das nächste Zeichenobjekt auszuwählen.

Zusätzlich ist es möglich, auf Modifikationen des Zeichenobjekts zu reagieren. Nähere Informationen hierzu findet sich in der Dokumentation zu `set_drawing_object_callback`.

Das Zeichenobjekt kann durch den Operator `detach_drawing_object_from_window` aus dem aktuellen Fenster entfernt werden.

Sobald ein Zeichenobjekt an einem HALCON-Fenster hängt, wird jedes HALCON-Objekt, das durch einen Operator, wie z.B. `disp_obj`, `disp_region`, `disp_image` oder `disp_xld` dargestellt wird, intern in einem Grafik-Stapel gespeichert, damit sie zusammen mit dem Zeichenobjekt angezeigt werden können. Die HALCON-Objekte werden solange angezeigt, bis `clear_window` aufgerufen wird oder das Fenster geschlossen wird.

Die Größe des Stapels kann durch `get_system` und `set_system` abgefragt und gesetzt werden. Siehe die entsprechende Referenzdokumentation für weitere Details.

Achtung

Es sollte kein Operator, der auf die Event-Queue des HALCON-Fensters zugreift, wie z.B. `get_mbutton` oder `read_char`, verwendet werden, solange Zeichenobjekte ans Fenster angehängt sind. Sie stehen im Gegensatz zu der Interaktion mit Zeichenobjekten, da sie aktiv auf die Eventloop des Fensters zugreifen.

Außerdem sollte keiner der blockierenden Operatoren `draw_rectangle1`, `draw_rectangle2`, `draw_region`, `draw_xld` oder `draw_circle` zusammen mit der Event-Funktionalität verwendet werden, da sie die Interaktion mit Zeichenobjekten verhindern.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **DrawHandle** (input_control) ..... drawing_object ⇝ *handle*
Handle des Zeichenobjektes.

Ergebnis

`attach_drawing_object_to_window` liefert den Wert TRUE, falls `DrawHandle` und `WindowHandle` gültig sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_drawing_object_rectangle1`, `create_drawing_object_xld`, `create_drawing_object_circle_sector`, `create_drawing_object_ellipse_sector`, `create_drawing_object_ellipse`, `create_drawing_object_line`, `create_drawing_object_rectangle2`, `create_drawing_object_circle`, `create_drawing_object_text`

Nachfolger

`detach_drawing_object_from_window`, `get_drawing_object_params`, `get_drawing_object_iconic`

Siehe auch

`set_drawing_object_callback`, `get_drawing_object_iconic`, `get_drawing_object_params`

Modul

Foundation

clear_drawing_object (: : DrawID :)

Löschen eines Zeichenobjekts.

`clear_drawing_object` löscht das Zeichenobjekt `DrawID`. Das Zeichenobjekt wird automatisch von allen Fenster entfernt, an die es zuvor angehängt wurde. Nach dem Aufruf von `clear_drawing_object` kann das Zeichenobjekt nicht weiter verwendet werden.

Parameter

▷ **DrawID** (input_control) ..... drawing_object ⇝ *handle*
Handle des Zeichenobjekts.

Ergebnis

`clear_drawing_object` gibt TRUE zurück, falls die `DrawID` gültig ist. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DrawID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_drawing_object_rectangle1`, `create_drawing_object_rectangle2`, `create_drawing_object_ellipse`, `create_drawing_object_circle_sector`, `create_drawing_object_ellipse_sector`, `create_drawing_object_xld`, `create_drawing_object_line`, `create_drawing_object_circle`, `create_drawing_object_text`, `detach_drawing_object_from_window`, `attach_drawing_object_to_window`

Nachfolger

`clear_window`, `close_window`

Siehe auch

`create_drawing_object_rectangle1`, `create_drawing_object_rectangle2`, `create_drawing_object_ellipse`, `create_drawing_object_circle_sector`, `create_drawing_object_ellipse_sector`, `create_drawing_object_xld`, `create_drawing_object_line`, `create_drawing_object_circle`, `create_drawing_object_text`

Modul

Foundation

create_drawing_object_circle (: : Row, Column, Radius : DrawID)

Erzeugen eines Kreises, der interaktiv verändert werden kann.

`create_drawing_object_circle` erstellt einen Kreis, der in einem HALCON-Fenster interaktiv vom Benutzer modifiziert werden kann. Der Kreis wird durch die Koordinaten `Row` und `Column` des Mittelpunkts und den Radius `Radius` definiert.

Diese Parameter können mit dem Operator `get_drawing_object_params` abgerufen werden. Das entsprechende HALCON-Objekt kann mit `get_drawing_object_iconic` abgefragt werden. Zusätzlich zu den Parametern verfügt jedes Zeichenobjekt über Attribute, die dessen grafische Darstellung in einem HALCON Fenster festlegen. Nähere Informationen zu den verfügbaren Attributen sind in der Dokumentation zu `set_drawing_object_params` zu finden.

Um das Zeichenobjekt interaktiv abändern zu können, muss es zunächst mit `attach_drawing_object_to_window` an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu `attach_drawing_object_to_window` zu finden. Es gibt zwei mögliche Transformationen für ein Kreis-Zeichenobjekt: Verändern der Größe und Verschieben. Der Kreis kann durch Ziehen am Mittelpunkt verschoben werden. Durch Ziehen am Kreisbogen kann die Größe des Kreises angepasst werden.

Im Gegensatz zum Operator `draw_circle` blockieren die Interaktionen mit einem Zeichenobjekt den aufrufenden Thread nicht.

Parameter

- ▷ **Row** (input_control) ..... circle.center.y ⇝ *real*
  Zeilenkoordinate des Mittelpunktes.
  Defaultwert: 100
- ▷ **Column** (input_control) ..... circle.center.x ⇝ *real*
  Spaltenkoordinate des Mittelpunktes.
  Defaultwert: 100
- ▷ **Radius** (input_control) ..... circle.radius ⇝ *real*
  Radius des Kreises.
  Defaultwert: 80
- ▷ **DrawID** (output_control) ..... drawing_object ⇝ *handle*
  Handle des Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_drawing_object_circle` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_window`

Nachfolger

`attach_drawing_object_to_window`, `get_drawing_object_params`, `get_drawing_object_iconic`

Alternativen

`draw_circle`, `draw_region`, `create_drawing_object_circle_sector`, `create_drawing_object_ellipse`, `create_drawing_object_ellipse_sector`

Siehe auch

`create_drawing_object_rectangle1`, `create_drawing_object_rectangle2`, `create_drawing_object_ellipse`, `create_drawing_object_circle_sector`, `create_drawing_object_ellipse_sector`, `create_drawing_object_xld`

Modul

Foundation

create_drawing_object_circle_sector (: : Row, Column, Radius, StartAngle, EndAngle : DrawID)

Erzeugen eines Kreisbogens, der interaktiv verändert werden kann.

`create_drawing_object_circle_sector` erzeugt einen Kreis, der in einem HALCON-Fenster interaktiv vom Benutzer modifiziert werden kann. Der Kreis wird durch die Koordinaten `Row` und `Column` des Mittelpunkts, den Radius `Radius` und die Start- und Endwinkel `StartAngle` und `EndAngle` definiert.

Diese Parameter können mit dem Operator `get_drawing_object_params` abgerufen werden. Das entsprechende HALCON-Objekt kann mit `get_drawing_object_iconic` abgefragt werden. Zusätzlich zu den Parametern verfügt jedes Zeichenobjekt über Attribute, die dessen grafische Darstellung in einem HALCON-Fenster festlegen. Nähere Informationen zu den verfügbaren Attributen sind in der Dokumentation zu `set_drawing_object_params` zu finden.

Um das Zeichenobjekt interaktiv abändern zu können, muss es zunächst mit `attach_drawing_object_to_window` an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu `attach_drawing_object_to_window` zu finden. Es gibt drei mögliche Transformationen für ein Kreisbogen-Zeichenobjekt: Verändern der Größe, Verschieben und Verändern der Winkel. Der Kreisbogen kann durch Ziehen am Mittelpunkt verschoben werden. Durch Ziehen am Kreisbogen kann die Größe des Kreises angepasst werden. Die Winkel können durch Ziehen an den Anfassern des Kreisbogens verändert werden.

Im Gegensatz zum Operator `draw_circle` blockieren die Interaktionen nicht den aufrufenden Thread.

Parameter

▷ **Row** (input_control) ..... coordinates.y ⇝ *real*
Zeilenkoordinate des Mittelpunkts.
Defaultwert: 100

▷ **Column** (input_control) ..... coordinates.x ⇝ *real*
Spaltenkoordinate des Mittelpunkts.
Defaultwert: 100

▷ **Radius** (input_control) ..... number ⇝ *real*
Radius des Kreises.
Defaultwert: 80

▷ **StartAngle** (input_control) . . . . . angle.rad ⇝ *real*
Startwinkel des Bogenausschnitts.
Defaultwert: 0

▷ **EndAngle** (input_control) . . . . . angle.rad ⇝ *real*
Endwinkel des Bogenausschnitts.
Defaultwert: 3.14159

▷ **DrawID** (output_control) . . . . . drawing_object ⇝ *handle*
Handle des Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_drawing_object_circle_sector` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_window`

Nachfolger

`attach_drawing_object_to_window`, `get_drawing_object_params`, `get_drawing_object_iconic`

Alternativen

`draw_rectangle1_mod`, `draw_rectangle2`, `draw_region`, `create_drawing_object_circle`, `create_drawing_object_ellipse_sector`

Siehe auch

`create_drawing_object_rectangle1`, `create_drawing_object_rectangle2`, `create_drawing_object_ellipse`, `create_drawing_object_circle`, `create_drawing_object_ellipse_sector`, `create_drawing_object_xld`, `create_drawing_object_line`

Modul

Foundation

create_drawing_object_ellipse (: : Row, Column, Phi, Radius1, Radius2 : DrawID)

Erzeugen einer Ellipse, die interaktiv verändert werden kann.

`create_drawing_object_ellipse` erzeugt eine Ellipse, die in einem HALCON-Fenster interaktiv durch den Benutzer verändert werden kann. Die Ellipse ist durch die Koordinaten `Row` und `Column` des Mittelpunkts, die Orientierung `Phi` und die Länge der Halbachsen `Radius1` und `Radius2` definiert.

Diese Parameter können mit dem Operator `get_drawing_object_params` abgerufen werden. Das entsprechende HALCON-Objekt kann mit `get_drawing_object_iconic` abgefragt werden. Zusätzlich zu den Parametern verfügt jedes Zeichenobjekt über Attribute, die dessen grafische Darstellung in einem HALCON-Fenster festlegen. Nähere Informationen zu den verfügbaren Attributen sind in der Dokumentation zu `set_drawing_object_params` zu finden.

Um das Zeichenobjekt interaktiv abändern zu können, muss es zunächst mit `attach_drawing_object_to_window` an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu `attach_drawing_object_to_window` zu finden. Es gibt drei mögliche Transformationen für ein Ellipsen-Zeichenobjekt: Verändern der Größe, Verschieben und Rotieren. Die Ellipse kann durch Ziehen am Mittelpunkt der Ellipse verschoben werden. Durch Ziehen an den Scheiteln kann

die Größe der Ellipse angepasst werden. Die Rotation kann durch Ziehen an einem Scheitel der ersten Halbachse verändert werden.

Im Gegensatz zum Operator `draw_ellipse` blockieren die Interaktionen mit einem Zeichenobjekt den aufrufenden Thread nicht.

Parameter

- ▷ **Row** (input_control) ..... ellipse.center.y ⇝ *real*
  Zeilenkoordinate des Mittelpunktes.
  Defaultwert: 200
- ▷ **Column** (input_control) ..... ellipse.center.x ⇝ *real*
  Spaltenkoordinate des Mittelpunktes.
  Defaultwert: 200
- ▷ **Phi** (input_control) ..... ellipse.angle.rad ⇝ *real*
  Orientierung der ersten Halbachse in Bogenmaß.
  Defaultwert: 0
- ▷ **Radius1** (input_control) ..... ellipse.radius1 ⇝ *real*
  Erste Halbachse.
  Defaultwert: 100
- ▷ **Radius2** (input_control) ..... ellipse.radius2 ⇝ *real*
  Zweite Halbachse.
  Defaultwert: 60
- ▷ **DrawID** (output_control) ..... drawing_object ⇝ *handle*
  Handle des Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_drawing_object_ellipse` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_window`

Nachfolger

`disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`, `attach_drawing_object_to_window`

Alternativen

`draw_ellipse_mod`, `draw_ellipse`, `draw_region`

Siehe auch

`gen_ellipse`, `draw_circle`, `draw_rectangle2`, `set_insert`

Modul

Foundation

create_drawing_object_ellipse_sector (: : Row, Column, Phi, Radius1, Radius2, StartAngle, EndAngle : DrawID)

Erzeugen eines Ellipsenbogens, der interaktiv verändert werden kann.

`create_drawing_object_ellipse_sector` erzeugt einen Ellipsenbogen, der in einem HALCON-Fenster interaktiv durch den Benutzer verändert werden kann. Der Ellipsenbogen ist definiert durch die Koordinaten `Row` und `Column` des Mittelpunktes, die Längen `Radius1` und `Radius2` der entsprechenden Halbachsen, die Start- und Endwinkel `StartAngle` und `EndAngle` des Bogens sowie der Orientierung `Phi`.

Diese Parameter können mit dem Operator get_drawing_object_params abgerufen werden. Das entsprechende HALCON-Objekt kann mit get_drawing_object_iconic abgefragt werden. Zusätzlich zu den Parametern verfügt jedes Zeichenobjekt über Attribute, die dessen grafische Darstellung in einem HALCON-Fenster festlegen. Nähere Informationen zu den verfügbaren Attributen sind in der Dokumentation zu set_drawing_object_params zu finden.

Um das Zeichenobjekt interaktiv abändern zu können, muss es zunächst mit attach_drawing_object_to_window an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu attach_drawing_object_to_window zu finden. Es gibt vier mögliche Transformationen für ein Ellipsenbogen-Zeichenobjekt: Verändern der Größe, Verschieben, Rotieren und Verändern des Winkels. Der Ellipsenbogen kann durch Ziehen am Mittelpunkt verschoben werden. Die Orientierung kann durch Ziehen an einem Scheitelanfasser der ersten Halbachse verändert werden. Die Größe und der Winkel des Ellipsenbogens kann durch Ziehen an den Scheitelanfassern angepasst werden.

Im Gegensatz zum Operator draw_ellipse blockieren die Interaktionen nicht den aufrufenden Thread.

Parameter

- ▷ **Row** (input_control) . . . . . coordinates.y ⇝ *real*
  Zeilenkoordinate des Mittelpunkts.
  Defaultwert: 200
- ▷ **Column** (input_control) . . . . . coordinates.x ⇝ *real*
  Spaltenkoordinate des Mittelpunkts.
  Defaultwert: 200
- ▷ **Phi** (input_control) . . . . . angle.rad ⇝ *real*
  Orientierung der ersten Halbachse in Bogenmaß.
  Defaultwert: 0
- ▷ **Radius1** (input_control) . . . . . number ⇝ *real*
  Erste Halbachse.
  Defaultwert: 100
- ▷ **Radius2** (input_control) . . . . . number ⇝ *real*
  Zweite Halbachse.
  Defaultwert: 60
- ▷ **StartAngle** (input_control) . . . . . angle.rad ⇝ *real*
  Startwinkel des Bogenausschnitts.
  Defaultwert: 0
- ▷ **EndAngle** (input_control) . . . . . angle.rad ⇝ *real*
  Endwinkel des Bogenausschnitts.
  Defaultwert: 3.14159
- ▷ **DrawID** (output_control) . . . . . drawing_object ⇝ *handle*
  Handle des Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_drawing_object_ellipse_sector den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

open_window

Nachfolger

attach_drawing_object_to_window, get_drawing_object_params, get_drawing_object_iconic

Alternativen

draw_ellipse_mod, draw_ellipse, draw_region, create_drawing_object_ellipse

Siehe auch

gen_ellipse_contour_xld, create_drawing_object_rectangle1, create_drawing_object_rectangle2, create_drawing_object_ellipse, create_drawing_object_xld, create_drawing_object_line, create_drawing_object_circle

Modul

Foundation

create_drawing_object_line (: : Row1, Column1, Row2, Column2 : DrawID)

Erzeugen einer Linie, die interaktiv verändert werden kann.

create_drawing_object_line erzeugt eine Linie, die interaktiv durch den Benutzer modifiziert werden kann. Die Linie ist definiert durch die Koordinaten Row1, Column1, Row2 und Column2 des ersten und zweiten Linienpunkts.

Diese Parameter können mit dem Operator get_drawing_object_params abgerufen werden. Das entsprechende HALCON-Objekt kann mit get_drawing_object_iconic abgefragt werden. Zusätzlich zu den Parametern verfügt jedes Zeichenobjekt über Attribute, die dessen grafische Darstellung in einem HALCON-Fenster festlegen. Nähere Informationen zu den verfügbaren Attributen sind in der Dokumentation zu set_drawing_object_params zu finden.

Um das Zeichenobjekt interaktiv abändern zu können, muss es zunächst mit attach_drawing_object_to_window an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu attach_drawing_object_to_window zu finden. Es gibt zwei mögliche Transformationen für ein Linien-Zeichenobjekt: Verändern der Länge und Verschieben. Ein Linienpunkt kann durch Ziehen am Punktanfasser verschoben und damit auch die Länge verändert werden. Die Linie selbst kann durch Ziehen am Mittelpunkt der Linie verschoben werden.

Im Gegensatz zum Operator draw_line blockieren die Interaktionen nicht den aufrufenden Thread.

Parameter

- ▷ **Row1** (input_control) ..... line.begin.y ⇝ *real*
  Zeilenkoordinate des ersten Punktes der Linie.
  Defaultwert: 100
- ▷ **Column1** (input_control) ..... line.begin.x ⇝ *real*
  Spaltenkoordinate des ersten Punktes der Linie.
  Defaultwert: 100
- ▷ **Row2** (input_control) ..... line.end.y ⇝ *real*
  Zeilenkoordinate des zweiten Punktes der Linie.
  Defaultwert: 200
- ▷ **Column2** (input_control) ..... line.end.x ⇝ *real*
  Spaltenkoordinate des zweiten Punktes der Linie.
  Defaultwert: 200
- ▷ **DrawID** (output_control) ..... drawing_object ⇝ *handle*
  Handle des Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_drawing_object_line den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_window`

Nachfolger

`attach_drawing_object_to_window`, `get_drawing_object_params`, `get_drawing_object_iconic`

Alternativen

`draw_line`, `draw_line_mod`, `create_drawing_object_xld`

Siehe auch

`create_drawing_object_rectangle1`, `create_drawing_object_rectangle2`, `create_drawing_object_ellipse`, `create_drawing_object_circle_sector`, `create_drawing_object_ellipse_sector`, `create_drawing_object_xld`

Modul

Foundation

create_drawing_object_rectangle1 (: : Row1, Column1, Row2, Column2 : DrawID)

Erzeugen eines achsenparallelen Rechtecks, das interaktiv verändert werden kann.

`create_drawing_object_rectangle1` erzeugt ein achsenparalleles Rechteck, das interaktiv in einem HALCON-Fenster modifiziert werden kann. Das Rechteck ist durch die Koordinaten `Row1` und `Column1` des linken oberen Eckpunkts und `Row2` und `Column2` des rechten unteren Eckpunkts definiert. Das erzeugte Rechteck wird in dem Handle `DrawID` zurückgeliefert.

Die Parameter können mit dem Operator `get_drawing_object_params` abgerufen werden. Das entsprechende HALCON-Objekt kann mit `get_drawing_object_iconic` abgefragt werden. Zusätzlich zu den Parametern verfügt jedes Zeichenobjekt über Attribute, die dessen grafische Darstellung in einem HALCON-Fenster festlegen. Nähere Informationen zu den verfügbaren Attributen sind in der Dokumentation zu `set_drawing_object_params` zu finden.

Um das Zeichenobjekt interaktiv abändern zu können, muss es zunächst mit `attach_drawing_object_to_window` an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu `attach_drawing_object_to_window` zu finden. Es gibt zwei mögliche Transformationen für ein Rectangle1-Zeichenobjekt: Verändern der Größe und Verschieben. Das Rechteck kann durch Ziehen in der Mitte des Rechtecks verschoben werden und durch Ziehen an den Anfassern an den Ecken vergrößert oder verkleinert werden.

Im Gegensatz zum Operator `draw_rectangle1` blockieren die Interaktionen mit einem Zeichenobjekt den aufrufenden Thread nicht.

Parameter

- **Row1** (input_control) . . . . . rectangle.origin.y ⇝ *real*
  Zeilenkoordinate des linken oberen Eckpunkts.
  Defaultwert: 100
- **Column1** (input_control) . . . . . rectangle.origin.x ⇝ *real*
  Spaltenkoordinate des linken oberen Eckpunkts.
  Defaultwert: 100
- **Row2** (input_control) . . . . . rectangle.corner.y ⇝ *real*
  Zeilenkoordinate des rechten unteren Eckpunkts.
  Defaultwert: 200
- **Column2** (input_control) . . . . . rectangle.corner.x ⇝ *real*
  Spaltenkoordinate des rechten unteren Eckpunkts.
  Defaultwert: 200
- **DrawID** (output_control) . . . . . drawing_object ⇝ *handle*
  Handle des Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_drawing_object_rectangle1` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_window`

Nachfolger

`disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`, `attach_drawing_object_to_window`

Alternativen

`draw_rectangle1_mod`, `draw_rectangle2`, `draw_region`

Siehe auch

`gen_rectangle1`, `draw_circle`, `draw_ellipse`, `set_insert`

Modul

Foundation

`create_drawing_object_rectangle2` (: : `Row`, `Column`, `Phi`, `Length1`, `Length2` : `DrawID`)

Erzeugen eines beliebig orientierten Rechtecks, das interaktiv verändert werden kann.

`create_drawing_object_rectangle2` erzeugt ein beliebig orientiertes Rechteck, das in deinem HALCON-Fenster interaktiv durch den Benutzer verändert werden kann. Das Rechteck ist durch die Koordinaten des Mittelpunkts, `Row` und `Column`, die Orientierung `Phi` und der Länge der Halbachsen `Length1` und `Length2` definiert.

Diese Parameter können mit dem Operator `get_drawing_object_params` abgerufen werden. Das entsprechende HALCON-Objekt kann mit `get_drawing_object_iconic` abgefragt werden. Zusätzlich zu den Parametern verfügt jedes Zeichenobjekt über Attribute, die dessen grafische Darstellung in einem HALCON-Fenster festlegen. Nähere Informationen zu den verfügbaren Attributen sind in der Dokumentation zu `set_drawing_object_params` zu finden.

Um das Zeichenobjekt interaktiv abändern zu können, muss es zunächst mit `attach_drawing_object_to_window` an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu `attach_drawing_object_to_window` zu finden. Es gibt drei mögliche Transformationen für ein Rectangle2-Zeichenobjekt: Verändern der Größe, Verschieben und Rotieren. Das Rechteck kann durch Ziehen am Mittelpunkt des Rechtecks verschoben werden. Durch Ziehen an den Seitenanfassern kann die Größe des Rechtecks angepasst werden. Die Orientierung kann durch Anfassen an einer Seite senkrecht zur ersten Halbachse verändert werden.

Im Gegensatz zum Operator `draw_rectangle2` blockieren die Interaktionen mit einem Zeichenobjekt den aufrufenden Thread nicht.

Parameter

▷ **`Row`** (input_control) ..... rectangle2.center.y ⇝ *real*
Zeilenkoordinate des Mittelpunktes.
Defaultwert: 150

▷ **`Column`** (input_control) ..... rectangle2.center.x ⇝ *real*
Spaltenkoordinate des Mittelpunktes.
Defaultwert: 150

▷ **Phi** (input_control) . . . . . rectangle2.angle.rad ⇝ *real*
Orientierung der größeren Halbachse in Bogenmaß.
Defaultwert: 0

▷ **Length1** (input_control) . . . . . rectangle2.hwidth ⇝ *real*
Erste Halbachse.
Defaultwert: 100

▷ **Length2** (input_control) . . . . . rectangle2.hheight ⇝ *real*
Zweite Halbachse.
Defaultwert: 100

▷ **DrawID** (output_control) . . . . . drawing_object ⇝ *handle*
Handle des Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_drawing_object_rectangle2` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_window`

Nachfolger

`disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`, `attach_drawing_object_to_window`

Alternativen

`draw_rectangle2_mod`, `draw_rectangle2`, `draw_region`

Siehe auch

`gen_rectangle2`, `draw_circle`, `draw_ellipse`, `set_insert`

Modul

Foundation

create_drawing_object_text (: : Row, Column, String : DrawID)

Erzeugen eines Text-Zeichenobjekts, das interaktiv in einem HALCON-Fenster verschoben werden kann.

`create_drawing_object_text` erstellt ein Text-Zeichenobjekt, das interaktiv vom Benutzer in einem HALCON-Fenster verschoben werden kann. Die Position des Textes ist durch die Parameter `Row` und `Column` definiert.

Diese Parameter können mit dem Operator `get_drawing_object_params` abgerufen werden.

Um das Text-Zeichenobjekt interaktiv verschieben zu können, muss es zunächst mit `attach_drawing_object_to_window` an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu `attach_drawing_object_to_window` zu finden. Der Text wird mit den aktuellen Einstellungen des HALCON-Fensters dargestellt. Die Einstellungen zur Schrift können jedoch auch mit dem Operator `set_drawing_object_params` angepasst werden. Sobald das Text-Zeichenobjekt einem Fenster zugewiesen ist, kann es interaktiv mit der Maus in dem Fenster verschoben werden.

Parameter

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenkoordinate der Textposition.
Defaultwert: 12

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenkoordinate der Textposition.
Defaultwert: 12

▷ **String** (input_control) ..... string ⇝ *string*
Zeichenkette.
Defaultwert: 'Text'

▷ **DrawID** (output_control) ..... drawing_object ⇝ *handle*
Handle des Text-Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, liefert `create_drawing_object_text` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_window`

Nachfolger

`attach_drawing_object_to_window`, `get_drawing_object_params`, `set_drawing_object_params`

Alternativen

`write_string`

Siehe auch

`write_string`, `attach_drawing_object_to_window`

Modul

Foundation

`create_drawing_object_xld` (: : Row, Column : DrawID)

Erzeugen einer Kontur, die interaktiv verändert werden kann.

`create_drawing_object_xld` erzeugt eine XLD-Kontur, die interaktiv in einem HALCON-Fenster verändert werden kann. Die Kontur ist durch die Koordinaten `Row` und `Column` ihrer Punkte definiert.

Diese Parameter können mit dem Operator `get_drawing_object_params` abgerufen werden. Das entsprechende HALCON-Objekt kann mit `get_drawing_object_iconic` abgefragt werden. Zusätzlich zu den Parametern verfügt jedes Zeichenobjekt über Attribute, die dessen grafische Darstellung in einem HALCON-Fenster festlegen. Nähere Informationen zu den verfügbaren Attributen sind in der Dokumentation zu `set_drawing_object_params` zu finden.

Um das Zeichenobjekt interaktiv abändern zu können, muss es zunächst mit `attach_drawing_object_to_window` an ein existierendes Fenster angehängt werden. Nähere Informationen sind in der Dokumentation zu `attach_drawing_object_to_window` zu finden. Weitere Punkte können durch einen Linksklick auf das Fenster hinzugefügt werden. Wenn auf einem Konturesegment zwischen zwei nachfolgenden Punkten geklickt wird, wird der neue Punkt zwischen diesen Punkten hinzugefügt. Jeder Konturpunkt kann durch Ziehen an dessen Anfasser verschoben werden. Ein einzelner Konturpunkt kann

gelöscht werden, indem er auf einen benachbarten, direkt verbundenen Konturpunkt gezogen wird. Die Kontur kann geschlossen werden, indem der letzte Punkt auf den ersten Konturpunkt gesetzt wird. Die ganze Kontur kann durch Ziehen am Anfasser des Mittelpunktes verschoben werden. Zusätzlich ist es auch möglich mit `set_drawing_object_xld` einem Zeichenobjekt eine neue Kontur zuzuweisen.

Im Gegensatz zum Operator `draw_xld` blockieren die Interaktionen nicht den aufrufenden Thread.

Parameter

▷ **`Row`** (input_control) . . . . . coordinates.y-array ⇝ *real* / integer
Zeilen-Koordinaten des Polygons.
Defaultwert: [100,200,200,100]
Wertevorschläge: `Row` ∈ {0, 10, 20, 50, 100, 200, 500}

▷ **`Column`** (input_control) . . . . . coordinates.x-array ⇝ *real* / integer
Spalten-Koordinaten des Polygons.
Defaultwert: [100,100,200,200]
Wertevorschläge: `Column` ∈ {0, 10, 20, 50, 100, 200, 500}

▷ **`DrawID`** (output_control) . . . . . drawing_object ⇝ *handle*
Handle des Zeichenobjekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_drawing_object_xld` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_window`

Nachfolger

`attach_drawing_object_to_window`, `get_drawing_object_params`, `get_drawing_object_iconic`, `set_drawing_object_xld`

Alternativen

`draw_xld`, `draw_xld_mod`, `draw_region`

Siehe auch

`create_drawing_object_rectangle1`, `create_drawing_object_rectangle2`, `create_drawing_object_ellipse`, `create_drawing_object_circle_sector`, `create_drawing_object_ellipse_sector`, `create_drawing_object_line`, `create_drawing_object_circle`

Modul

Foundation

`detach_background_from_window` (: : WindowHandle :)

Entfernen des Hintergrundbildes von einem HALCON-Fenster.

`detach_background_from_window` entfernt das Hintergrundbild, das zuvor mit `attach_background_to_window` angehängt wurde, von dem HALCON-Fenster `WindowHandle`. Der Hintergrund des Fensters wird somit gelöscht.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

Ergebnis

`detach_background_from_window` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`attach_background_to_window`

Modul

Foundation

detach_drawing_object_from_window (: : WindowHandle, DrawHandle :)

Entfernen eines Zeichenobjektes von einem HALCON-Fenster.

`detach_drawing_object_from_window` entfernt das Zeichenobjekt `DrawHandle` von dem HALCON-Fenster `WindowHandle`, an das es aktuell angehängt ist. Das Zeichenobjekt wird in diesem Fenster nicht länger angezeigt. Wenn mit `set_drawing_object_callback` eine Callback Funktion für das Zeichenobjekt zu dem Event 'on_detach' definiert wurde, wird diese ebenfalls aufgerufen.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **DrawHandle** (input_control) ..... drawing_object ⇝ *handle*
Handle des Zeichenobjektes.

Ergebnis

`detach_drawing_object_from_window` liefert den Wert TRUE, falls `DrawHandle` und `WindowHandle` gültig sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`attach_drawing_object_to_window`, `get_drawing_object_iconic`, `get_drawing_object_params`

Nachfolger

`close_window`, `clear_drawing_object`

Siehe auch

`attach_drawing_object_to_window`

Modul

Foundation

get_drawing_object_iconic (: Object : DrawID :)

Zurückliefern des ikonischen Objekts eines Zeichenobjekts.

get_drawing_object_iconic gibt ein ikonisches Object Object zurück, das dem Zeichenobjekt DrawID entspricht. Die Parameter des Zeichenobjekts können mit dem Operator get_drawing_object_params abgefragt werden.

Parameter

▷ **Object** (output_object) ..... object(-array) ⇝ *object*
Ikonisches Objekts.

▷ **DrawID** (input_control) ..... drawing_object ⇝ *handle*
Handle des Zeichenobjekts.

Ergebnis

get_drawing_object_iconic gibt TRUE zurück, falls die DrawID gültig ist. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_drawing_object_rectangle1, create_drawing_object_rectangle2, create_drawing_object_ellipse, create_drawing_object_circle, create_drawing_object_line, create_drawing_object_ellipse_sector, create_drawing_object_circle_sector, create_drawing_object_xld

Nachfolger

reduce_domain, gen_region_contour_xld, attach_drawing_object_to_window

Alternativen

get_drawing_object_params

Siehe auch

get_drawing_object_params, attach_drawing_object_to_window

Modul

Foundation

get_drawing_object_params (: : DrawID, GenParamName : GenParamValue)

Abfragen der Parameter eines Zeichenobjekts.

get_drawing_object_params liefert die aktuell gesetzten Werte GenParamValue für die Parameter GenParamName des Zeichenobjekts DrawID.

Zusätzlich zu den in set_drawing_object_params beschriebenen Werten können folgende Parameter GenParamName abgefragt werden:

'type': Liefert den Typ den Zeichenobjekts. Mögliche Rückgabewerte sind: *'circle'*, *'circle_sector'*, *'ellipse'*, *'ellipse_sector'*, *'rectangle1'*, *'rectangle2'*, *'line'*, *'xld'*, *'text'*

Parameter

▷ **DrawID** (input_control) ..... drawing_object ⇝ *handle*
Handle des Zeichenobjekts.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der Parameter des Zeichenobjekts.
Werteliste: GenParamName ∈ {'row1', 'row2', 'column1', 'column2', 'radius', 'row', 'column', 'length1', 'length2', 'string', 'phi', 'start_angle', 'end_angle', 'radius1', 'radius2', 'color', 'font', 'line_width', 'line_style', 'type'}

▷ **GenParamValue** (output_control) . . . . . attribute.name(-array) ⇝ *real* / integer / string
Werte der Parameter.

Ergebnis

get_drawing_object_params gibt TRUE zurück, falls die DrawID, GenParamName und GenParamValue gültig sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_drawing_object_rectangle1, create_drawing_object_rectangle2, create_drawing_object_ellipse, create_drawing_object_circle, create_drawing_object_line, create_drawing_object_ellipse_sector, create_drawing_object_circle_sector, create_drawing_object_xld, create_drawing_object_text, attach_drawing_object_to_window

Nachfolger

set_drawing_object_params, attach_drawing_object_to_window, get_drawing_object_iconic

Alternativen

get_drawing_object_iconic

Siehe auch

attach_drawing_object_to_window, get_drawing_object_iconic, set_drawing_object_params

Modul

Foundation

get_window_background_image (: BackgroundImage : WindowHandle :)

Kopieren des Hintergrundbildes des Fensters.

get_window_background_image gibt eine Kopie des Hintergrundbildes des HALCON-Fensters WindowHandle in BackgroundImage zurück. Der Operator ist vor allem sinnvoll, wenn man in Rückruffunktionen eines Zeichenobjektes mit dem Hintergrundbild arbeiten möchte. Siehe set_drawing_object_callback für weitere Details.

Parameter

▷ **BackgroundImage** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Kopie des Hintergrundbildes.

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

Ergebnis

get_window_background_image liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

attach_background_to_window

Nachfolger

close_window, detach_background_from_window, attach_drawing_object_to_window

Siehe auch

attach_background_to_window, detach_drawing_object_from_window

Modul

Foundation

set_content_update_callback (: : WindowHandle, CallbackFunction, CallbackContext :)

Setzen einer Callback-Funktion bei Aktualisierung des Pufferinhalts.

set_content_update_callback setzt die Callback-Funktion CallbackFunction, die immer dann aufgerufen wird, wenn sich der Inhalt des Pufferfensters WindowHandle ändert. Der Parameter CallbackContext wird dabei immer an die Callback-Funktion übergeben.

Der Typ der Callback-Funktion CallbackFunction ist: Herror callback(void *context)

Auf Windows Systemen wird die __stdcall Aufrufkonvention verwendet: Herror __stdcall callback(void *context)

Die Callback-Funktion darf keinen HALCON Operator aufrufen. Dies ist undefiniertes Verhalten. Es wird empfohlen, nur wenig rechenintensive Aufgaben in der Callback-Funktion durchzuführen, z.B. das Setzen eines Flags oder das Anstoßen einer asynchronen Operation.

In Verbindung mit den Operatoren send_mouse_down_event, send_mouse_up_event und send_mouse_double_click_event kann set_content_update_callback verwendet werden, um ein Fenster-Control für ein GUI Toolkit wie Qt zu erstellen. Das Control leitet Mausevents an den Puffer weiter, welcher das Control bei Änderungen des Fensterinhalts benachrichtigt, indem er CallbackFunction aufruft. Das Control kann dann den Inhalt das Puffers mit dump_window_image kopieren und schließlich anzeigen.

Achtung

set_content_update_callback benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **CallbackFunction** (input_control) ..... pointer ⇝ *integer*
  Callback Funktion bei Aktualisierung des Pufferinhalts.
- ▷ **CallbackContext** (input_control) ..... pointer ⇝ *integer* / handle
  Parameter für CallbackFunction.

Ergebnis

set_content_update_callback liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

flush_buffer

Modul

Foundation

set_drawing_object_callback (: : DrawHandle, DrawObjectEvent, CallbackFunction :)

Hinzufügen einer Callback-Funktion zu einem Zeichenobjekt.

set_drawing_object_callback fügt die Rückruffunktion CallbackFunction zu einem Zeichenobjekt hinzu, die bei einem definierten Event DrawObjectEvent ausgeführt wird. Ein Event ist eine Aktion die vom Benutzer oder einem HALCON-Operator, wie z.B. attach_drawing_object_to_window, ausgelöst wird und den Status des Zeichenobjekts verändert. Der Operator set_drawing_object_callback ermöglicht es zu definieren, wie auf solche Events reagiert wird.

Die vordefinierten Events sind:

'on_attach' Die entsprechende Rückruffunktion wird gleich nach dem Anhängen des Zeichenobjekts ans Fenster aufgerufen.

'on_detach' Die entsprechende Rückruffunktion wird gleich nach dem Entfernen des Zeichenobjekts vom Fenster aufgerufen.

'on_drag' Die entsprechende Rückruffunktion wird gleich nach dem Verschieben des Zeichenobjekts aufgerufen.

'on_resize' Die entsprechende Rückruffunktion wird gleich nach dem Anpassen der Größe des Zeichenobjekts aufgerufen.

'on_select' Die entsprechende Rückruffunktion wird gleich nach dem Auswählen und Aktivierung vom Zeichenobjekt aufgerufen.

Es ist möglich eine bestimmte Rückruffunktion für jedes gewünschte Event zu setzten. Dafür müssen die Eingabeparameter DrawObjectEvent und CallbackFunction die gleiche Länge haben. Es ist ebenso möglich die gleiche Rückruffunktion für verschiedene Events zu definieren. Dann kann DrawObjectEvent eine beliebige Länge haben und CallbackFunction enthält einen einzigen Wert.

Die Rückruffunktion ist ein Integer vom Typ long und beinhaltet einen Pointer zu einer C-Funktion folgender Signatur:

Herror HDrawObjCallback(long DrawHandle, long WindowHandle, char* type)

Unter Windows 32 bit Systemen, wird die __stdcall Namenskonvention benutzt:

Herror (__stdcall HDrawObjCallback)(long DrawHandle, long WindowHandle, char* type)

Der erste Parameter der Rückruffunktion enthält das Handle des Zeichenobjekts, das das Event auslöst. Der zweite Parameter enthält das Handle zu dem Fenster, in dem das Event ausgelöst wurde. Der Dritte gibt an, welches Event geschehen ist.

Achtung

Es darf kein grafischer Operator innerhalb der Rückruffunktion aufgerufen werden, wie. z.B. disp_obj, da sonst ein Deadlock vorkommen könnte.

Parameter

- ▷ **DrawHandle** (input_control) ..... drawing_object ⇝ *handle*
  Handle des Zeichenobjekts.
- ▷ **DrawObjectEvent** (input_control) ..... string(-array) ⇝ *string*
  Liste der zu setzenden Events.
  Wertevorschläge: DrawObjectEvent ∈ {'on_resize', 'on_drag', 'on_attach', 'on_detach', 'on_select'}
- ▷ **CallbackFunction** (input_control) ..... pointer(-array) ⇝ *integer*
  Liste entsprechenden Callback-Funktionen.

Ergebnis

set_drawing_object_callback liefert den Wert TRUE, falls DrawHandle und DrawObjectEvent gültig sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

attach_drawing_object_to_window

Nachfolger

close_window, clear_drawing_object, detach_drawing_object_from_window, get_drawing_object_iconic, get_drawing_object_params

Siehe auch

create_drawing_object_rectangle1, attach_drawing_object_to_window, detach_drawing_object_from_window

Modul

Foundation

set_drawing_object_params (: : DrawID, GenParamName, GenParamValue :)

Setzen der Parameterwerte eines Zeichenobjekts.

set_drawing_object_params setzt die Parameter GenParamName des Zeichenobjekts DrawID auf die Werte in GenParamValue.

Je nach Typ des Zeichenobjekts können folgende Parameter gesetzt werden:

circle: *'row'*: Zeilenkoordinate des Mittelpunkts.
'column': Spaltenkoordinate des Mittelpunkts.
'radius': Radius des Kreises.

circle_sector: *'row'*: Zeilenkoordinate des Mittelpunkts.
'column': Spaltenkoordinate des Mittelpunkts.
'radius': Radius des Kreissektors.
'start_angle': Anfangswinkel des Kreisbogens.
'end_angle': Endwinkel des Kreisbogens.

ellipse: *'row'*: Zeilenkoordinate des Mittelpunkts.
'column': Spaltenkoordinate des Mittelpunkts.
'phi': Orientierung der ersten Halbachse im Bogenmaß.
'radius1': Länge der ersten Halbachse der Ellipse.
'radius2': Länge der zweiten Halbachse der Ellipse.

ellipse_sector: *'row'*: Zeilenkoordinate des Mittelpunkts.
'column': Spaltenkoordinate des Mittelpunkts.
'phi': Orientierung der ersten Halbachse im Bogenmaß.
'radius1': Länge der ersten Halbachse.
'radius2': Länge der zweiten Halbachse.
'start_angle': Anfangswinkel des Ellipsenbogens.
'end_angle': Endwinkel des Ellipsenbogens.

rectangle1: *'row1'*: Zeilenkoordinate des linken oberen Eckpunkts.
'column1': Spaltenkoordinate des linken oberen Eckpunkts.
'row2': Zeilenkoordinate des rechten oberen Eckpunkts.
'column2': Spaltenkoordinate des rechten oberen Eckpunkts.

rectangle2: *'row'*: Zeilenkoordinate des Mittelpunkts des Rechtecks.
'column': Spaltenkoordinate des Mittelpunkts des Rechtecks.
'phi': Orientierung der ersten Halbachse im Bogenmaß.
'lenght1': Länge der ersten Halbachse.
'length2': Länge der zweiten Halbachse.

line: *'row1'*: Zeilenkoordinate des ersten Linienpunkts.
'column1': Spaltenkoordinate des ersten Linienpunkts.
'row2': Zeilenkoordinate des zweiten Linienpunkts.
'column2': Spaltenkoordinate des zweiten Linienpunkts.

xld: *'row'*: Zeilenkoordinaten der Konturpunkte.
'column': Spaltenkoordinaten der Konturpunkte.
Es ist möglich, entweder nur die Zeilen- oder Spaltenkoordinaten zu setzen oder beide zur gleichen Zeit. Im ersten Fall muss die Anzahl der Werte in `GenParamValue` der Anzahl der vorhandenen Konturpunkte entsprechen. Im zweiten Fall werden alle vorhandenen Konturpunkte verworfen und durch neue ersetzt. Die neuen Koordinaten werden in `GenParamValue` in folgender Reihenfolge übergeben: *[Row_0, Row_1, ..., Row_n, Column_0, Column_1, ... Column_n]*.

text: *'row'*: Zeilenkoordinate der Textposition.
'column': Spaltenkoordinate der Textposition.
'string': Textstring, der angezeigt wird.

Es gibt zusätzliche Attribute, die das Aussehen des Zeichenobjekts modellieren. Je nach Typ des Zeichenobjekts können die folgenden Attribute gesetzt werden:

- Zeichenobjekte vom Typ 'text':
  'color': Setzt die Farbe eines Text-Zeichenobjekts. Die Farbe kann entweder mit ihrem Namen angegeben werden, z.B. 'green', oder als RGB Wert, z.B. '#ffb529'. Siehe `set_color` für mehr Informationen.
  'font': Setzt den Font eines Text-Zeichenobjekts. Eine Liste aller verfügbaren Fonts kann mit dem Operator `query_font` abgerufen werden. Siehe `set_font` für mehr Informationen.
- Zeichenobjekte vom Typ 'circle', 'circle_sector', 'ellipse', 'ellipse_sector', 'rectangle1', 'rectangle1', 'xld', 'line':
  'color': Setzt die Farbe eines Text-Zeichenobjekts. Die Farbe kann entweder mit ihrem Namen, z.B. 'green', angegeben werden, oder als RGB Wert, z.B. '#ffb529'. Siehe `set_color` für mehr Informationen.
  'line_style': Setzt den Linienstil des Zeichenobjekts. Dieses Attribut benötigt mindestens zwei Attributwerte in Form eines Tupels. mindestens zwei Attributwerte in Form eines Tupels. Der erste Wert definiert die Länge des sichtbaren Linienstücks und der zweite Wert definiert die Länge des nicht sichtbaren Linienstücks. Siehe `set_line_style` für mehr Details. Da die Länge des Eingabetupels variieren kann, darf *'line_style'* nicht mit anderen Parametern oder Attributen in einem Aufruf kombiniert werden. Ansonsten wird eine Fehlerbehandlung durchgeführt.
  'line_width': Setzt die Linienbreite des Zeichenobjekts in Pixel. Siehe `set_line_width` für mehr Informationen.

Wenn mehr als ein Parameter gesetzt werden soll, müssen diese als Tupel in `GenParamName` und `GenParamValue` übergeben werden.

Wenn das Zeichenobjekt aktuell an ein Fenster gehängt ist, wird dieses beim Aufruf von `set_drawing_object_params` aktualisiert.

Parameter

▷ **`DrawID`** (input_control) ..... drawing_object ⇝ *handle*
Handle des Zeichenobjekts.

▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
Namen der Parameter des Zeichenobjekts.
Werteliste: `GenParamName` ∈ {'row1', 'row2', 'column1', 'column2', 'radius', 'row', 'column', 'length1', 'length2', 'string', 'phi', 'start_angle', 'end_angle', 'radius1', 'radius2', 'color', 'font', 'line_width', 'line_style'}

▷ **`GenParamValue`** (input_control) ..... attribute.name(-array) ⇝ *real* / integer / string
Werte der Parameter.

Ergebnis

`get_drawing_object_params` gibt TRUE zurück, falls die `DrawID`, `GenParamName` und `GenParamValue` gültig sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_drawing_object_rectangle1, create_drawing_object_rectangle2, create_drawing_object_ellipse, create_drawing_object_circle, create_drawing_object_line, create_drawing_object_ellipse_sector, create_drawing_object_circle_sector, create_drawing_object_xld, attach_drawing_object_to_window

Nachfolger

get_drawing_object_params, attach_drawing_object_to_window, get_drawing_object_iconic

Siehe auch

attach_drawing_object_to_window, get_drawing_object_iconic, set_drawing_object_callback

Modul

Foundation

set_drawing_object_xld (Contour : : DrawID :)

Setzen einer XLD-Kontur eines XLD-Zeichenobjekts.

set_drawing_object_xld ersetzt die XLD-Kontur, die durch das Zeichenobjekts DrawID repräsentiert wird, durch die neue XLD-Kontur Contour. Die Kontur muss mindestens zwei Punkte enthalten, so dass eine Interaktion, wie in create_drawing_object_xld beschrieben, möglich ist.

Wenn die Kontur bereits an ein HALCON-Fenster angehängt ist, wird dieses beim Aufruf von set_drawing_object_xld aktualisiert.

Parameter

▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
XLD-Kontur.

▷ **DrawID** (input_control) ..... drawing_object ⇝ *handle*
Handle des Zeichenobjekts.

Ergebnis

set_drawing_object_xld gibt TRUE zurück, falls die DrawID gültig ist und Contour eine Kontur ist. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_drawing_object_xld

Nachfolger

attach_drawing_object_to_window, clear_drawing_object

Alternativen

create_drawing_object_xld

Siehe auch

attach_drawing_object_to_window

Modul

Foundation

11.7 Parameter

convert_coordinates_image_to_window (: : WindowHandle, RowImage, ColumnImage : RowWindow, ColumnWindow)

Konvertieren von Bildkoordinaten in Fensterkoordinaten

convert_coordinates_image_to_window konvertiert die Bildkoordinaten RowImage und ColumnImage in Fensterkoordinaten RowWindow und ColumnWindow basierend auf dem Bildausschnitt und der Fenstergröße des durch WindowHandle gegebenen Fensters.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **RowImage** (input_control) ..... coordinates.y(-array) ⇝ *real*
  Zeile in Bildkoordinaten.
- ▷ **ColumnImage** (input_control) ..... coordinates.x(-array) ⇝ *real*
  Spalte in Bildkoordinaten.
- ▷ **RowWindow** (output_control) ..... coordinates.y(-array) ⇝ *real*
  Zeile (Y) in Fensterkoordinaten.
- ▷ **ColumnWindow** (output_control) ..... coordinates.x(-array) ⇝ *real*
  Spalte (X) in Fensterkoordinaten.

Beispiel

```
read_image (Image, 'printer_chip/printer_chip_01')
dev_get_window (WindowHandle)
get_window_extents (WindowHandle, Row, Column, Width, Height)
dev_set_part (450, 300, 750, 600)
dev_display (Image)
*
* Generate rectangle in image coordinates
Row := [474, 746]
Column := [314, 589]
gen_rectangle1 (Rectangle1, Row[0], Column[0], Row[1], Column[1])
* Convert rectangle corner points to window coordinates
convert_coordinates_image_to_window (WindowHandle, Row[[0,1,0,1]], \
    Column[[0,0,1,1]], RowWindow, ColumnWindow)
*
* Window center in window coordinates
WindowCenterRow := Height/2-1
WindowCenterColumn := Width/2-1
* Convert window center to image coordinates
convert_coordinates_window_to_image (WindowHandle, WindowCenterRow, \
    WindowCenterColumn, RowImage, ColumnImage)
*
* Display all points in image coordinates
dev_display (Image)
disp_cross (WindowHandle, Row[[0,1,0,1]], Column[[0,0,1,1]], 6, rad(45))
disp_cross (WindowHandle, RowImage, ColumnImage, 6, 0)
```

Ergebnis

convert_coordinates_image_to_window liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_window_extents, set_part

Siehe auch

convert_coordinates_window_to_image

Modul

Foundation

convert_coordinates_window_to_image (: : WindowHandle, RowWindow, ColumnWindow : RowImage, ColumnImage)

Konvertieren von Fensterkoordinaten in Bildkoordinaten

convert_coordinates_window_to_image konvertiert Fensterkoordinaten RowWindow und ColumnWindow in Bildkoordinaten RowImage und ColumnImage basierend auf dem Bildausschnitt und der Fenstergröße des durch WindowHandle gegebenen Fensters.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **RowWindow** (input_control) ..... coordinates.y(-array) ⇝ *real*
  Zeile (Y) in Fensterkoordinaten.
- ▷ **ColumnWindow** (input_control) ..... coordinates.x(-array) ⇝ *real*
  Spalte (X) in Fensterkoordinaten.
- ▷ **RowImage** (output_control) ..... coordinates.y(-array) ⇝ *real*
  Zeile in Bildkoordinaten.
- ▷ **ColumnImage** (output_control) ..... coordinates.x(-array) ⇝ *real*
  Spalte in Bildkoordinaten.

Beispiel

```
read_image (Image, 'printer_chip/printer_chip_01')
dev_get_window (WindowHandle)
get_window_extents (WindowHandle, Row, Column, Width, Height)
dev_set_part (450, 300, 750, 600)
dev_display (Image)
*
* Generate rectangle in image coordinates
Row := [474, 746]
Column := [314, 589]
gen_rectangle1 (Rectangle1, Row[0], Column[0], Row[1], Column[1])
* Convert rectangle corner points to window coordinates
convert_coordinates_image_to_window (WindowHandle, Row[[0,1,0,1]], \
    Column[[0,0,1,1]], RowWindow, ColumnWindow)
*
* Window center in window coordinates
WindowCenterRow := Height/2-1
WindowCenterColumn := Width/2-1
* Convert window center to image coordinates
convert_coordinates_window_to_image (WindowHandle, WindowCenterRow, \
    WindowCenterColumn, RowImage, ColumnImage)
*
* Display all points in image coordinates
dev_display (Image)
disp_cross (WindowHandle, Row[[0,1,0,1]], Column[[0,0,1,1]], 6, rad(45))
disp_cross (WindowHandle, RowImage, ColumnImage, 6, 0)
```

Ergebnis

`convert_coordinates_window_to_image` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`set_window_extents`, `set_part`

Siehe auch

`convert_coordinates_image_to_window`

Modul

Foundation

get_contour_style (: : WindowHandle : Style)

Abfragen des aktuellen Füllstils der Konturanzeige.

`get_contour_style` gibt den Füllstil zurück, der mit `set_contour_style` gesetzt wurde.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Style** (output_control) ..... string ⇝ *string*
  Füllstil der Konturanzeige.

Ergebnis

`get_contour_style` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`set_contour_style`, `disp_xld`

Siehe auch

`set_contour_style`, `disp_xld`

Modul

Foundation

get_draw (: : WindowHandle : Mode)

Abfragen des aktuellen Modus für das Ausfüllen von Regionen.

`get_draw` gibt den Ausfüllmodus `Mode` des Ausgabefensters aus, der von Operatoren wie `disp_region`, `disp_circle`, `disp_arrow`, `disp_rectangle1`, `disp_rectangle2` etc., verwendet wird. Gesetzt wird der Darstellungsparameter mit dem Operator `set_draw`.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.
▷ **Mode** (output_control) . . . . . string ⇝ *string*
Aktueller Regionen-Füllmodus.

Ergebnis

`get_draw` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`set_draw`, `disp_region`

Siehe auch

`set_draw`, `disp_region`, `set_paint`

Modul

Foundation

get_hsi (: : WindowHandle : Hue, Saturation, Intensity)

Abfragen der aktuellen Farbe in HSI-Kodierung.

`get_hsi` gibt die Ausgabefarben, bzw. Grauwerte für das Ausgabefenster an. Diese werden durch `Hue` (Farbwert), `Saturation` (Sättigung) und `Intensity` (Intensität) beschrieben. Die Werte die man mit `get_hsi` erhält können mit `set_hsi` wieder gesetzt werden.

Achtung

Die Werte von `get_hsi` können durch Rundungsfehler ungenau sein. Sie entsprechen nicht genau den Werten die z.B. mit `set_hsi` gesetzt wurden (intern wird die Farbe in RGB gehalten).

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.
▷ **Hue** (output_control) . . . . . integer-array ⇝ *integer*
Farbwert der aktuellen Zeichenfarbe.
▷ **Saturation** (output_control) . . . . . integer-array ⇝ *integer*
Sättigung der aktuellen Zeichenfarbe.
▷ **Intensity** (output_control) . . . . . integer-array ⇝ *integer*
Intensität der aktuellen Zeichenfarbe.

Ergebnis

`get_hsi` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`set_hsi`, `set_rgb`, `disp_image`

Siehe auch

`set_hsi`, `set_color`, `set_rgb`, `trans_to_rgb`, `trans_from_rgb`

Modul

Foundation

get_icon (: Icon : WindowHandle :)

Abfragen des Icons, das bei der Regionenausgabe verwendet wird

get_icon fragt das Icon, das mit set_icon gesetzt wurde.

Parameter

- ▷ **Icon** (output_object) ..... region ⇝ *object*
  Icon für Schwerpunkt.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.

Beispiel

```
/* draw a region and an icon.  */
/* set it and get it again.    */
T_draw_region(&Region,WindowHandle);
T_draw_region(&Icon,WindowHandle);
set_icon(Icon);
create_tuple_s(&icon,"icon");
T_set_shape(WindowHandle,icon);
destroy_tuple(icon);
T_disp_region(Region,WindowHandle);
get_icon(&OldIcon);
T_disp_region(OldIcon,WindowHandle);
```

Ergebnis

get_icon liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_icon

Nachfolger

disp_region

Modul

Foundation

get_line_style (: : WindowHandle : Style)

Abfragen des aktuellen Grafikmodus für Konturen.

get_line_style gibt die Darstellungsart für Konturen der Regionenausgabe des Ausgabefensters an, die von Operatoren wie disp_region, disp_line, disp_polygon, etc., verwendet wird. Gesetzt wird der Parameter mit dem Operator set_line_style. Der Parameter ist nur von Bedeutung, wenn die Ränder von Objekten dargestellt werden.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Style** (output_control) . . . . . integer-array ⇝ *integer*
Muster für Konturdarstellung.

Ergebnis

`get_line_style` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`set_line_style`, `disp_region`

Modul

Foundation

get_line_width (: : WindowHandle : Width)

Abfragen der aktuellen Strichstärke für Konturen.

`get_line_width` gibt die Strichstärke der Regionenausgabe des Ausgabefensters an, die von Operatoren wie `disp_region`, `disp_line`, `disp_polygon`, etc., verwendet wird. Gesetzt wird der Parameter mit dem Operator `set_line_width`. Der Parameter ist nur von Bedeutung, wenn die Ränder von Objekten dargestellt werden.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Width** (output_control) . . . . . real ⇝ *real*
Aktuelle Strichstärke für Konturen.

Ergebnis

`get_line_width` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`set_line_width`, `set_line_style`, `disp_region`

Siehe auch

`set_line_width`, `disp_region`

Modul

Foundation

get_paint (: : WindowHandle : Mode)

Abfragen des aktuellen Darstellungsmodus für Grauwerte.

get_paint gibt den Modus des Ausgabefensters für die Darstellung von Grauwerten, der von dem Operator disp_image verwendet wird, an. Das Abfragen wird benutzt, wenn der aktuelle Modus kurzzeitig überschrieben werden soll. Hierzu wird zunächst der aktuelle Wert abgefragt, dann überschrieben und schließlich auf den alten Wert zurückgesetzt. Das Zurücksetzen erfolgt mit set_paint. Alle verfügbaren Modi können mit dem Operator query_paint abgefragt werden. Der Parameter Mode enthält den Namen des Ausgabemodus. Falls ein Modus über Parameter genauer gesteuert werden kann, werden diese Werte (in einem Tupel) nach dem Namen übergeben. Die Reihenfolge der Werte ist die gleiche wie beim Setzen (set_paint).

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Mode** (output_control) ..... string-array ⇝ *string* / integer / real
  Name und Parameter des aktuellen Ausgabemodus.

Ergebnis

get_paint liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

query_paint

Nachfolger

set_paint, disp_region, disp_image

Siehe auch

set_paint, query_paint, disp_image

Modul

Foundation

get_part (: : WindowHandle : Row1, Column1, Row2, Column2)

Abfragen des Bildausschnitts.

get_part liefert den linken oberen und den rechten unteren Eckpunkt des Bildausschnitts, der im Ausgabefenster dargestellt wird. Dieser Bildausschnitt kann mittels set_part verändert werden (Als Voreinstellung wird immer das volle Bild ausgegeben).

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row1** (output_control) ..... rectangle.origin.y ⇝ *integer* / real
  Zeilennummer des linken oberen Eckpunkts des Bildausschnitts.
- ▷ **Column1** (output_control) ..... rectangle.origin.x ⇝ *integer* / real
  Spaltennummer des linken oberen Eckpunkts des Bildausschnitts.
- ▷ **Row2** (output_control) ..... rectangle.corner.y ⇝ *integer* / real
  Zeilennummer des rechten unteren Eckpunkts des Bildausschnitts.
- ▷ **Column2** (output_control) ..... rectangle.corner.x ⇝ *integer* / real
  Spaltennummer des rechten unteren Eckpunkts des Bildausschnitts.

Ergebnis

get_part liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_part, disp_region, disp_image

Siehe auch

set_part, disp_image, disp_region, disp_color

Modul

Foundation

get_part_style (: : WindowHandle : Style)

Abfragen der aktuellen Interpolationsmethode für die Graubildausgabe.

get_part_style fragt die Interpolationsmethode ab, die für die Ausgabe eines Bildausschnitts im Ausgabefenster verwendet wird. Interpoliert werden muss immer dann, wenn das Ausgabefenster größer als das Bildformat bzw. als der mittels set_part eingestellte Bildausschnitt für die Ausgabe ist. Derzeit werden drei Modi (Style) unterstützt:

0 keine Interpolation (eventuell niedrigere Qualität, sehr schnell).

1 ungewichtete Interpolation zwischen den Grauwerten (mittlere Qualität und Laufzeit).

2 gewichtete Interpolation zwischen den Grauwerten (höchste Qualität, langsam).

Die aktuelle Einstellung kann mittels set_part_style verändert werden.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Style** (output_control) . . . . . integer ⇝ *integer*
Interpolationsmethode für die Bildschirmdarstellung: 0 (schnell,niedrige Qualität) bis 2 (langsam, höchste Qualität).
Werteliste: Style ∈ {0, 1, 2}

Ergebnis

get_part_style liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_part_style, disp_region, disp_image

Siehe auch

set_part_style, set_part, disp_image, disp_color

Modul

Foundation

get_rgb (: : WindowHandle : Red, Green, Blue)

Abfragen der aktuellen Farbe in RGB-Kodierung.

get_rgb gibt die Ausgabefarben bzw. Grauwerte für das Ausgabefenster an. Diese sind durch die drei Farbanteile: rot, grün und blau beschrieben. Die Werte die man mit get_rgb erhält, können mit set_rgb wieder gesetzt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Red** (output_control) ..... integer-array ⇝ *integer*
  Rotanteil der aktuellen Farbe.
- ▷ **Green** (output_control) ..... integer-array ⇝ *integer*
  Grünanteil der aktuellen Farbe.
- ▷ **Blue** (output_control) ..... integer-array ⇝ *integer*
  Blauanteil der aktuellen Farbe.

Ergebnis

get_rgb liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_rgb, disp_region, disp_image

Siehe auch

set_rgb

Modul

Foundation

get_rgba (: : WindowHandle : Red, Green, Blue, Alpha)

Abfragen der aktuellen Farbe in RGBA-Kodierung.

get_rgba gibt die Ausgabefarben bzw. Grauwerte für das Ausgabefenster an. Diese sind durch die drei Farbanteile: rot, grün und blau und den Alphawert Alpha beschrieben. Die Werte die man mit get_rgba erhält, können mit set_rgba wieder gesetzt werden.

Achtung

get_rgba benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Red** (output_control) ..... integer-array ⇝ *integer*
  Rotanteil der aktuellen Farbe.
- ▷ **Green** (output_control) ..... integer-array ⇝ *integer*
  Grünanteil der aktuellen Farbe.
- ▷ **Blue** (output_control) ..... integer-array ⇝ *integer*
  Blauanteil der aktuellen Farbe.
- ▷ **Alpha** (output_control) ..... integer-array ⇝ *integer*
  Alphawert der aktuellen Farbe.

Ergebnis

get_rgba liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`set_rgba`, `disp_obj`

Siehe auch

`set_rgba`

Modul

Foundation

get_shape (: : WindowHandle : DisplayShape)

Abfragen der aktuellen Ausgabeform für Regionen.

`get_shape` gibt die Gestalt an, in der Regionen gezeichnet werden. Der erhaltene Wert kann mit `set_shape` gesetzt werden. Alle verfügbaren Darstellungsarten können mit `query_shape` abgefragt werden.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **DisplayShape** (output_control) . . . . . string ⇝ *string*
Ausgabeform für Regionen.

Ergebnis

`get_shape` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`query_shape`

Nachfolger

`set_shape`, `disp_region`

Siehe auch

`set_shape`, `query_shape`, `disp_region`

Modul

Foundation

get_window_param (: : WindowHandle, Param : Value)

Zugreifen auf Fensterparameter.

Der Operator `get_window_param` erlaubt es, verschiedene Parameter eines offenen Fensters abzufragen. Genauere Beschreibungen der einzelnen Parameter finden sich beim Operator `set_window_param`.

Allgemeine Fensterparameter:

'flush' *'true'*, falls der Fensterinhalt nach Anzeige jedes Objektes aktualisiert wird.

'region_quality' Qualität der Anzeige von Regionen.

'background_color' Die Hintergrundfarbe des Fensters `WindowHandle`.

'window_title' Der Text in der Titelzeile eines HALCON Fensters.

'anti_aliasing' *'true'*, falls Konturen und Regionen mit Antialiasing dargestellt werden.

'graphics_stack' *'true'*, falls der Grafikstack des Fensters aktiviert ist.

'graphics_stack_max_element_num' Maximale Anzahl von Elementen im Grafikstack.

'graphics_stack_max_memory_size' Maximaler Speicher, welchen der Grafikstack verbraucht.

Parameter die den *'3d_plot'* Plot betreffen:

'angle_of_view' Der Öffnungswinkel der virtuellen Kamera, die für das Anzeigen des 3D-Plots verwendet wird.

'axis_captions' Die Beschriftungen der Zeilen-, Spalten- und Höhenachse.

'plot_quality' Die Qualität des 3D Plots.

'scale_plot' *'true'*, falls die Höhenwerte eine Bildes vor dem Anzeigen in das Intervall [0,255] transformiert werden.

'display_grid' *'true'*, falls das Gitter angezeigt wird.

'display_axes' *'true'*, falls die Koordinatenachsen angezeigt werden.

Achtung

Die Parameter *'anti_aliasing'*, *'flush'*, *'graphics_stack'*, *'graphics_stack_max_element_num'*, *'graphics_stack_max_memory_size'* und *'region_quality'* benötigen die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`Param`** (input_control) ..... string ⇝ *string*
  Name des Parameters.
  Defaultwert: 'flush'
  Werteliste: `Param` ∈ {'plot_quality', 'axis_captions', 'scale_plot', 'angle_of_view', 'display_grid', 'display_axes', 'window_title', 'background_color', 'graphics_stack', 'graphics_stack_max_element_num', 'graphics_stack_max_memory_size', 'flush', 'region_quality'}
- ▷ **`Value`** (output_control) ..... string(-array) ⇝ *string* / real / integer
  Wert des Parameters.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`set_window_param`

Siehe auch

`get_window_attr`, `get_part_style`

Modul

Foundation

`query_all_colors` (: : `WindowHandle` : `Colors`)

Abfragen aller Farbnamen.

`query_all_colors` gibt die Namen aller Farben aus, die HALCON bekannt sind. Das heißt natürlich nicht, dass all diese Farben tatsächlich verwendet werden können. Vielmehr ist nur ein Teil dieser Farben verfügbar

(siehe query_color). Die HALCON Grafikfarben dienen zur Darstellung von Regionen (disp_region, disp_polygon, disp_circle, etc.). Sie können mit dem Operator set_color gesetzt werden.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Colors** (output_control) . . . . . string-array ⇝ *string*
Namen der Farben.

Ergebnis

query_all_colors liefert immer den Wert TRUE

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_system, set_color, disp_region

Siehe auch

query_color, set_system, set_color, disp_region, open_window

Modul

Foundation

query_color (: : WindowHandle : Colors)

Abfragen aller im Fenster darstellbaren Farbnamen.

query_color gibt für das Ausgabefenster die Namen aller Farben aus, die bei der Darstellung von Regionen (disp_region, disp_polygon, disp_circle, etc.) verwendet werden können. Diese können mit dem Operator set_color gesetzt werden. Bei einem S/W-Bildschirm liefert query_color die Werte 'black' und 'white'. Diese beiden „Farben" stehen bei jedem Bildschirm zur Verfügung. Bei Bildschirmen mit Graustufen werden auch einige Grautöne angegeben (z.B.: 'dim gray'). Bei Bildschirmen mit Farbtabellen wird eine Liste von darstellbaren Farben zurückgeliefert. Per Default beginnt das Tupel der Farben dabei mit S/W, gefolgt von den drei Grundfarben ('red','green','blue') und einigen Grautönen. Eine Liste aller dem System bekannten Farben erhält man dazu mit dem Aufruf query_all_colors(::WindowHandle:Colors). Genau diese Liste wird übrigens bei Bildschirmen mit Echtfarbenausgabe auch von query_color ausgegeben. Dies sind natürlich nicht alle Farben, die dargestellt werden können. Diese müssen dann mit set_rgb oder set_hsi direkt gewählt werden. Werden Farbnamen verwendet, die HALCON zwar bekannt sind, aber auf dem Rechner nicht darstellbar sind, dann wählt HALCON eine ähnliche Farbe aus (und erzeugt keine Fehlermeldung), falls set_check mit dem Parameter 'color' aufgerufen wird.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Colors** (output_control) . . . . . string-array ⇝ *string*
Namen der Farben.

Beispiel

```
open_window(0,0,-1,-1,'root','invisible','',WindowHandle)
query_color(WindowHandle,Colors)
close_window(WindowHandle)
fwrite_string (FileHandle, ['Displayable colors: ',Colors])
```

Ergebnis

query_color liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_color, disp_region

Siehe auch

query_all_colors, set_color, disp_region, open_window

Modul

Foundation

query_colored (: : : PossibleNumberOfColors)

Abfragen der Farbenzahl für bunte Ausgaben.

query_colored gibt alle zulässigen Parameterwerte für set_colored aus. Bei set_colored wird festgelegt in wie vielen Farben Regionen, bzw. Grafik ausgegeben wird.

Parameter

▷ **PossibleNumberOfColors** (output_control) . . . . . integer-array ⇝ *integer*
Tupel der möglichen Farbenzahlen.

Beispiel

```
regiongrowing(Image,Regions,5,5,6,100)
query_colored(Colors)
dev_set_colored(Colors[1])
dev_display (Regions)
```

Ergebnis

query_colored liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_colored, set_color, disp_region

Alternativen

query_color

Siehe auch

set_colored, set_color

Modul

Foundation

query_gray (: : WindowHandle : Grayval)

Abfragen der darstellbaren Graustufen.

query_gray gibt für das Ausgabefenster alle exakt reproduzierbaren Grauwerte aus, die bei der Darstellung von Grauwerten (disp_image) verwendet werden. Diese können bei dem Operator set_gray gesetzt werden.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Grayval** (output_control) ..... integer-array ⇝ *integer*
Tupel der darstellbaren Graustufen.

Ergebnis

query_gray liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_gray, disp_region

Siehe auch

set_gray, disp_image

Modul

Foundation

query_line_width (: : : Min, Max)

Abfragen der möglichen Strichstärken.

query_line_width gibt an, wie dick (in Pixel) der Rand einer Region mindestens (Min) und höchstens (Max) dargestellt werden kann. Das Setzen der Randstärke erfolgt mit set_line_width und wird von Operationen wie disp_region, disp_line, disp_circle, disp_rectangle1, disp_rectangle2 etc. verwendet, falls der Zeichenmodus auf „Rand“ (set_draw(::WindowHandle,'margin':)) eingestellt ist.

Parameter

▷ **Min** (output_control) ..... integer ⇝ *integer*
Darstellbare Mindeststärke.

▷ **Max** (output_control) ..... integer ⇝ *integer*
Darstellbare Maximalstärke.

Ergebnis

query_line_width liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

get_line_width, set_line_width, set_line_style, disp_line

Siehe auch

disp_circle, disp_line, disp_rectangle1, disp_rectangle2, disp_region, set_line_width, get_line_width, set_line_style

Modul

Foundation

query_paint (: : WindowHandle : Mode)

Abfragen der Darstellungsmöglichkeiten für Grauwerte.

query_paint gibt für das Ausgabefenster die Namen aller Grauwertdarstellungsarten (z.B. 'gray', '3d_plot', etc.) aus. Diese können von dem Operator set_paint verwendet, d.h. aktiviert werden. query_paint gibt nur die Namen der Darstellungsarten, nicht aber die zusätzlichen Parameter (die bei einigen Arten festgelegt werden können) an.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Mode** (output_control) . . . . . string-array ⇝ *string*
Namen der Darstellungsmodi.

Ergebnis

query_paint liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

get_paint, set_paint, disp_image

Siehe auch

set_paint, get_paint, disp_image

Modul

Foundation

query_shape (: : : DisplayShape)

Abfragen der Darstellungsmodi für Regionen.

query_shape gibt für das Ausgabefenster die Namen aller Regionendarstellungsarten (z.B. 'original', 'circle', 'rectangle1', 'rectangle2', 'ellipse', etc.) aus. Diese können von dem Operator set_shape verwendet (d.h. aktiviert) werden.

Parameter

▷ **DisplayShape** (output_control) . . . . . string-array ⇝ *string*
Namen der Darstellungsmodi für Regionen.

Ergebnis

query_shape liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

get_shape, set_shape, disp_region

Siehe auch

set_shape, get_shape, disp_region

Modul

Foundation

set_color (: : WindowHandle, Color :)

Setzen der Ausgabefarbe(n).

set_color setzt für das Ausgabefenster die Farbe(n), in der die Regionen dargestellt werden sollen. Ein Tupel aller verfügbaren Farben kann mit dem Operator query_color abgefragt werden. Darüberhinaus kann Color auch über ein RGB-Tripel oder RGBA-Quadrupel in der Form *'#rrggbb'* bzw. *'#rrggbbaa'* gesetzt werden, wobei 'rr', 'gg', 'bb' und 'aa' jeweils hexadezimale Zahlen zwischen '00' und 'ff' sind. 'aa' ist der Alphawert der Farbe und kann genutzt werden, um transparente Regionen anzuzeigen.

Der Parameter Color kann eine einzelne Farbe oder ein Tupel mit bis zu 256 Farben enthalten. Wird eine einzige Farbe angegeben, dann erfolgen alle Ausgaben in dieser Farbe. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung von Tupeln von Regionen und geometrischen Objekten modulo dieser Farben. Im Beispiel (s.u.) wird der erste Kreis in Rot, der zweite in einem transparenten Grün und der dritte wieder in Rot ausgegeben. Bei jedem Aufruf einer Ausgabeprozedur wird mit der ersten Farbe begonnen. Es ist dabei zu beachten, dass es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen: Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in der ersten Farbe. Dies gilt auch für Objekte, die aus mehreren Zusammenhangskomponenten bestehen.

Die gesetzte(n) Farbe(n) werden so lange verwendet, bis erneut set_color, set_rgb, set_hsi oder set_gray aufgerufen wird.

Verwendet wird dieser Darstellungsparameter bei Operatoren wie disp_region,disp_line, disp_rectangle1, oder disp_arrow.

Color name	75% alpha	50% alpha	25% alpha
'black'	'#000000c0'	'#00000080'	'#00000040'
'white'	'#ffffffc0'	'#ffffff80'	'#ffffff40'
'red'	'#ff0000c0'	'#ff000080'	'#ff000040'
'green'	'#00ff00c0'	'#00ff0080'	'#00ff0040'
'blue'	'#0000ffc0'	'#0000ff80'	'#0000ff40'
'dim gray'	'#696969c0'	'#69696980'	'#69696940'
'gray'	'#bebebec0'	'#bebebe80'	'#bebebe40'
'light gray'	'#d3d3d3c0'	'#d3d3d380'	'#d3d3d340'
'cyan'	'#00ffffc0'	'#00ffff80'	'#00ffff40'
'magenta'	'#ff00ffc0'	'#ff00ff80'	'#ff00ff40'
'yellow'	'#ffff00c0'	'#ffff0080'	'#ffff0040'
'medium slate blue'	'#7b68eec0'	'#7b68ee80'	'#7b68ee40'
'coral'	'#ff7f50c0'	'#ff7f5080'	'#ff7f5040'
'slate blue'	'#6a5acdc0'	'#6a5acd80'	'#6a5acd40'
'spring green'	'#00ff7fc0'	'#00ff7f80'	'#00ff7f40'
'orange red'	'#ff4500c0'	'#ff450080'	'#ff450040'
'dark olive green'	'#556b2fc0'	'#556b2f80'	'#556b2f40'
'pink'	'#ffc0cbc0'	'#ffc0cb80'	'#ffc0cb40'
'cadet blue'	'#5f9ea0c0'	'#5f9ea080'	'#5f9ea040'
'goldenrod'	'#daa520c0'	'#daa52080'	'#daa52040'
'orange'	'#ffa500c0'	'#ffa50080'	'#ffa50040'
'gold'	'#ffd700c0'	'#ffd70080'	'#ffd70040'
'forest green'	'#228b22c0'	'#228b2280'	'#228b2240'
'cornflower blue'	'#6495edc0'	'#6495ed80'	'#6495ed40'
'navy'	'#000080c0'	'#00008080'	'#00008040'
'turquoise'	'#40e0d0c0'	'#40e0d080'	'#40e0d040'
'dark slate blue'	'#483d8bc0'	'#483d8b80'	'#483d8b40'
'light blue'	'#add8e6c0'	'#add8e680'	'#add8e640'
'indian red'	'#cd5c5cc0'	'#cd5c5c80'	'#cd5c5c40'
'violet red'	'#d02090c0'	'#d0209080'	'#d0209040'
'light steel blue'	'#b0c4dec0'	'#b0c4de80'	'#b0c4de40'
'medium blue'	'#0000cdc0'	'#0000cd80'	'#0000cd40'
'khaki'	'#f0e68cc0'	'#f0e68c80'	'#f0e68c40'
'violet'	'#ee82eec0'	'#ee82ee80'	'#ee82ee40'
'firebrick'	'#b22222c0'	'#b2222280'	'#b2222240'
'midnight blue'	'#191970c0'	'#19197080'	'#19197040'

Beispiele für mögliche Farbstrings

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Color** (input_control) ..... string(-array) ⇝ *string*
Namen der Ausgabefarben.
Defaultwert: 'white'
Wertevorschläge: Color ∈ {'black', 'white', 'red', 'green', 'blue', 'cyan', 'magenta', 'yellow', 'dim gray', 'gray', 'light gray', 'medium slate blue', 'coral', 'slate blue', 'spring green', 'orange red', 'orange', 'dark olive green', 'pink', 'cadet blue', '#003075', '#e53019', '#ffb529', '#f28d26bb'}

Beispiel

```
set_color(WindowHandle, ['red', '#00ff00a0'])
disp_circle(WindowHandle, [100,200,300], [200,300,100], [100,100,100])
```

Ergebnis

set_color liefert den Wert TRUE, falls das Fenster gültig ist und für dieses die angegebene Farbe(n) darstellbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

query_color

Nachfolger

disp_region

Alternativen

set_rgb, set_hsi

Siehe auch

get_rgb, disp_region, set_paint

Modul

Foundation

set_colored (: : WindowHandle, NumberOfColors :)

Setzen einer bunten Ausgabe.

set_colored ist eine Kurzform für bestimmte set_color Aufrufe und dient z.B. dazu, eine Menge von Regionen in unterschiedlichen Farben darzustellen. NumberOfColors bestimmt die Anzahl von Farben, die dazu alternierend verwendet werden sollen. Zulässige Werte von NumberOfColors können mittels query_colored abgefragt werden.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **NumberOfColors** (input_control) ..... integer ⇝ *integer*
Anzahl der Ausgabefarben.
Defaultwert: 12
Werteliste: NumberOfColors ∈ {3, 6, 12}

Ergebnis

set_colored liefert den Wert TRUE, falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

query_colored, set_color

Nachfolger

disp_region

Siehe auch

query_colored, set_color, disp_region

Modul

Foundation

set_contour_style (: : WindowHandle, Style :)

Setzen des Füllstils der Konturanzeige.

set_contour_style setzt den Füllstil der Konturanzeige. Dabei werden folgende Werte für Style unterstützt:

- *'stroke'*: Nur die Linie der Kontur wird gezeichnet
- *'fill'*: Die von der Kontur eingeschlossene Fläche wird ausgefüllt.
- *'stroke_and_fill'*: Die Linie der Kontur wird gezeichnet und die von ihr eingeschlossene Fläche ausgefüllt. Dies macht insbesondere bei größeren Liniendicken einen Unterschied, siehe set_line_width.

Bei allen Stilen wird die aktuelle Zeichenfarbe verwendet.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Style** (input_control) ..... string ⇝ *string*
Füllstil der Konturanzeige.
Defaultwert: 'stroke'
Werteliste: Style ∈ {'stroke', 'fill', 'stroke_and_fill'}

Ergebnis

set_contour_style liefert den Wert TRUE, falls der Wert für Style korrekt ist und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_contour_style

Nachfolger

disp_xld, disp_obj

Siehe auch

get_contour_style, disp_xld, set_line_width, set_line_style

Modul

Foundation

set_draw (: : WindowHandle, Mode :)

Setzen des Ausfüllmodus für Regionen.

set_draw gibt an, ob Regionen ausgefüllt ('fill') oder nur der Rand ('margin') dargestellt werden soll. Der angegebene Wert wird dem Fenster mit der logischen Fensternummer WindowHandle zugeordnet. Die Ausgabe erfolgt mit Operatoren wie disp_region, disp_circle, disp_rectangle1, disp_rectangle2, disp_arrow etc. Der Ausfüllmodus ist auch bei der grafischen Darstellung von Grauwerten relevant. Falls der Modus auf 'margin' gesetzt ist, kann der Rand noch mit set_line_width und set_line_style beeinflusst werden.

Achtung

Falls der Modus 'margin' gewählt wurde und die Strichstärke größer als eins ist, werden manche Hohlflächen nicht ausgegeben.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Ausfüllmodus für Regionen.
  Defaultwert: 'fill'
  Werteliste: Mode ∈ {'fill', 'margin'}

Ergebnis

set_draw liefert den Wert TRUE, falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_draw

Nachfolger

disp_region

Siehe auch

get_draw, disp_region, set_paint, disp_image, set_line_width, set_line_style

Modul

Foundation

set_gray (: : WindowHandle, GrayValues :)

Setzen eines Grauwertes für die Regionenausgabe.

set_gray gibt an, dass Regionen in bestimmten Grauwerten (GrayValues) dargestellt werden sollen. Unter einem Grauwert wird hier der Bereich der Farbtabelle verstanden, der für die Darstellung von Grauwertdaten (disp_image mit set_paint(::WindowHandle,'gray':)) verwendet wird. Hierbei handelt es sich um die Einträge der Farbtabelle, die von set_lut modifiziert werden. Unter einem „Grauwert“ ist die Farbe zu verstehen, in der ein Bildpunkt mit dem gleichen Wert dargestellt wird (was nicht notwendigerweise ein Grauton ist). Weiterhin ist zu beachten, dass sich die Farbe der Grafik, deren Aussehen mit set_gray festgelegt wurde, beim Laden einer neuen Farbtabelle i.A. ändern wird.

Wird ein Grauton als Grafikfarbe benötigt (d.h. keine Farbänderung durch set_lut), so kann dieser z.B. mit set_color(::WindowHandle,'gray':) gesetzt werden.

Wird ein einziger Grauwert angegeben, dann erfolgen alle Ausgaben in diesem Grauwert. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung der Tupel von Regionen und geometrischen Objekten modulo

dieser Grauwerte. Der erste Kreis im Beispiel (s.u.) wird in dem Grauwert 100, der zweite in dem Grauwert 200 und der dritte wieder in dem Grauwert 100 ausgegeben. Bei jedem Aufruf einer Ausgabeprozedur wird mit dem ersten Grauwert begonnen. Es ist dabei zu beachten, dass es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen: Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in dem ersten Grauwert. Dies gilt auch für Objekte, die aus mehreren Zusammenhangskomponenten bestehen.

Der Aufruf des Operators (`set_gray`, `set_color`, `set_rgb`, `set_hsi`) überschreibt die Werte, die vorher gesetzt wurden. Je nach Geräteeigenschaft können nicht alle Grauwerte dargestellt werden. Der Wertebereich des Parameters `GrayValues` (0..255) wird entsprechend auf die darstellbaren Graustufen abgebildet. In jedem Fall wird der Grauwert 0 als schwarz und der Grauwert 255 als weiß dargestellt. Mit dem Operator `query_gray` können die darstellbaren Grauwerte abgefragt werden. Mit dem Aufruf `set_check(::'~color':)` wird HALCON aufgefordert keine Fehlermeldung zu erzeugen, falls eine Grauwert verwendet wurde, der auf dem Bildschirm nicht darstellbar ist, sondern eine Näherung für diesen Wert zu suchen.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`GrayValues`** (input_control) ..... integer(-array) ⇝ *integer*
  Grauwerte für Regionendarstellung.
  Defaultwert: 255
  Wertevorschläge: `GrayValues` ∈ {0, 1, 2, 10, 16, 32, 64, 100, 120, 128, 250, 251, 252, 253, 254, 255}
  Typischer Wertebereich: 0 ≤ `GrayValues` ≤ 255

Beispiel

```
set_gray(WindowHandle,[100,200])
disp_circle(WindowHandle,[100,200,300],[200,300,100],[100,100,100])
```

Ergebnis

`set_gray` liefert den Wert TRUE, falls der angegebene Grauwert darstellbar und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`disp_region`

Siehe auch

`set_color`

Modul

Foundation

`set_hsi` (: : `WindowHandle`, `Hue`, `Saturation`, `Intensity` :)

Festlegen der Ausgabefarbe in der HSI-Kodierung.

`set_hsi` setzt für das Ausgabefenster die Farbe(n), bzw. die Grauwerte, in der Regionen dargestellt werden sollen. Die gewünschten Werte werden durch `Hue` (Farbwert), `Saturation` (Sättigung) und `Intensity` (Intensität) festgelegt. Die Umrechnung von HSI nach RGB erfolgt mit der Formel:

$$
\begin{aligned}
H &= (\texttt{Hue}/255.0) * 2\pi \\
S &= (\texttt{Saturation}/255.0) * 2.0/\sqrt{6} \\
I &= \texttt{Intensity} \\
\\
M1 &= S\cos(H)/\sqrt{6} * 255.0 \\
M2 &= S\sin(H)/\sqrt{2} * 255.0 \\
\\
Red &= 2M1 + I \\
Green &= -M1 + M2 + I \\
Blue &= -M1 - M2 + I
\end{aligned}
$$

Der Wertebereich wird auf 0 bis 255 begrenzt.

Wird nur eine Kombination angegeben, dann erfolgen alle Ausgaben in dieser Farbe. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung von Tupeln von Region und geometrischen Objekten modulo dieser Farben. Bei jedem Aufruf einer Ausgabeprozedur wird mit der ersten Farbe begonnen. Es ist dabei zu beachten, dass es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen: Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in der ersten Farbe. Dies gilt auch für Objekte die aus mehreren Zusammenhangskomponenten bestehen.

Die gesetzte Farbe(n) werden so lange verwendet, bis erneut `set_color`, `set_rgb` oder `set_gray` aufgerufen wird. Verwendet wird dieser Darstellungsparameter bei Operatoren wie `disp_region`, `disp_line`, `disp_rectangle1`, `disp_rectangle2`, `disp_arrow`, etc. Die Farbe wird aber auch bei der grafischen Darstellung von Grauwerten verwendet (z.B.: *'3d_plot'*, etc.; siehe hierzu `set_paint`).

Achtung

Die Farben werden intern als RGB Tripel gespeichert. Manche HSI Tripel können nicht durch ein gültiges RGB Tripel (im Bereich 0..255) repräsentiert werden. In diesem Fall wird die der gewünschten Farbe nächste repräsentierbare Farbe verwendet.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Hue`** (input_control) ..... integer(-array) ⇝ *integer*
Farbwert der aktuellen Zeichenfarbe.
Defaultwert: 30
Typischer Wertebereich: $0 \leq$ `Hue` ≤ 255
Restriktion: 0 <= Hue && Hue <= 255

▷ **`Saturation`** (input_control) ..... integer(-array) ⇝ *integer*
Sättigung der aktuellen Zeichenfarbe.
Defaultwert: 255
Typischer Wertebereich: $0 \leq$ `Saturation` ≤ 255
Restriktion: 0 <= Saturation && Saturation <= 255

▷ **`Intensity`** (input_control) ..... integer(-array) ⇝ *integer*
Intensität der aktuellen Zeichenfarbe.
Defaultwert: 84
Typischer Wertebereich: $0 \leq$ `Intensity` ≤ 255
Restriktion: 0 <= Intensity && Intensity <= 255

Ergebnis

`set_hsi` liefert den Wert TRUE, falls das Fenster gültig ist und für dieses die angegebene Farbe(n) darstellbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_hsi

Nachfolger

disp_region

Siehe auch

get_hsi, trans_from_rgb, trans_to_rgb, disp_region

Modul

Foundation

set_icon (Icon : : WindowHandle :)

Setzen eines Icons für die Regionenausgabe.

set_icon setzt ein Icon, das bei der Ausgabe von Regionen (disp_region) im Schwerpunkt der Region dargestellt werden soll. Die Verwendung dieses Icons wird durch set_shape aktiviert.

Parameter

- ▷ **Icon** (input_object) ..... region ⇝ *object*
  Icon für Schwerpunkt.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.

Beispiel

```
/* draw a region and an icon */
T_draw_region(&Region,WindowHandle);
T_draw_region(&Icon,WindowHandle);
set_icon(Icon);
create_tuple_s(&icon,"icon");
T_set_shape(WindowHandle,icon);
destroy_tuple(icon);
T_disp_region(Region,WindowHandle);
```

Ergebnis

set_icon liefert den Wert TRUE, falls genau eine Region übergeben wird. Ansonsten wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region

Nachfolger

set_shape, disp_region

Modul

Foundation

set_line_style (: : WindowHandle, Style :)

Festlegen des Musters für die Konturdarstellung.

set_line_style gibt an, wie der Rand einer Region oder XLD Konturen dargestellt werden soll. Diese Information wird von Operatoren wie disp_region, disp_line, disp_polygon etc. benötigt. Der aktuell eingestellte Wert kann mit dem Operator get_line_style abgefragt werden. Konkret enthält Style (derzeit maximal fünf) Zahlenpaare. Der erste Wert ist jeweils die Länge des sichtbaren Konturstücks, der zweite die Länge des darauf folgenden Zwischenraums. Die Zahlenpaare werden für die Ausgabe einer Kontur zyklisch herangezogen.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Style** (input_control) ..... integer-array ⇝ *integer*
Muster der Kontur.
Defaultwert: []
Typischer Wertebereich: $1 \leq$ Style ≤ 120

Beispiel

```
* stroke line: X-Windows
set_line_style(WindowHandle,[20,7])
* point-stroke line: X-Windows
set_line_style(WindowHandle,[20,7,3,7])
* passing line (standard)
set_line_style(WindowHandle,[])
```

Ergebnis

set_line_style liefert den Wert TRUE, falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_line_style

Nachfolger

disp_region

Siehe auch

get_line_style, set_line_approx, disp_region

Modul

Foundation

set_line_width (: : WindowHandle, Width :)

Festlegen der Strichstärke für die Konturdarstellung.

set_line_width gibt an, wie dick (in Pixel) der Rand einer Region dargestellt werden soll. Die Darstellung erfolgt mit Operatoren wie disp_region, disp_line, disp_polygon, etc. Mit dem Operator get_line_width kann der Ausgabeparameter für das Ausgabefenster abgefragt werden. Nicht bei allen Geräten besteht die Möglichkeit die Randstärke zu verändern. Dies kann mit dem Operator query_line_width abgefragt werden.

Achtung

Die Strichstärke ist nur von Bedeutung, falls der Modus set_draw(::WindowHandle,'margin':) eingestellt wurde. Falls die Strichstärke größer als 1 ist, können Hohlflächen von Regionen nicht immer korrekt ausgegeben werden.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Width** (input_control) ..... real ⇝ *real*
Strichstärke für Konturdarstellung.
Defaultwert: 1.0
Restriktion: Width >= 1.0 && Width <= 2000.0

Ergebnis

`set_line_width` liefert den Wert TRUE, falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`query_line_width`, `get_line_width`

Nachfolger

`disp_region`

Siehe auch

`get_line_width`, `query_line_width`, `set_draw`, `disp_region`

Modul

Foundation

set_paint (: : WindowHandle, Mode :)

Festlegen des Modus für die Darstellung von Grauwerten.

`set_paint` setzt den Modus des Ausgabefensters für die Darstellung von Grauwerten (ein- oder mehrkanalig), der von `disp_obj`, `disp_image` und `disp_color` verwendet werden.

Verschiedene Bildtypen und ihr möglicher Einfluss auf `Mode`:

- Graubilder können auch als 3D-Höhenmodell aufgefasst werden. Hierzu steht eine 3D-Darstellung mit *'3d_plot'* zur Verfügung.
- Dreikanalige Bilder werden als RGB-Daten interpretiert.
- Vektorfeldbilder werden als Vektorfelder ausgegeben (*'vector_field'*).

In den meisten Fällen ist `Mode` *'default'* die richtige Wahl.

Je nach ausgewähltem Modus erlaubt `Mode` ein Tupel als Wert. In diesen Fällen sind folgende Eingaben möglich:

- Es wird nur der Name des Modus angegeben. Hier werden die Voreinstellungen oder die zuletzt gesetzten Werte verwendet.
  Beispiel: `set_paint(WindowHandle,'3d_plot')`
- Es werden alle Werte übergeben. Hierdurch ist es möglich alle Eigenschaften der Ausgabe festzulegen.
  Beispiel: `set_paint(WindowHandle,['3d_plot','shaded', 8,0.939,-0.052,0.296,-0.165,0.401,`
- Nur die ersten n Werte werden angeben. Hierbei werden die nicht angegebenen Werte nicht verändert.
  Beispiel: `set_paint(WindowHandle,['3d_plot','texture'])`
- Einige der Werte werden durch einen Stern ersetzt ('*'). Dies bedeutet, dass dieser Wert nicht verändert werden soll.
  Beispiel: `set_paint(WindowHandle,['3d_plot','*',8])`

Ist der Modus *'default'* gewählt, dann muss für die Ausgabe von 2- und 3-kanaligen Bildern kein Aufruf von `set_paint` erfolgen. Es wird dann automatisch ein geeignetes Verfahren aufgerufen.

Die Ausgabe der Grauwerte wird auch von anderen Operatoren (und deren Parametern) beeinflusst. Dies sind insbesondere `set_part`, `set_part_style` und `set_lut`. Bei einigen Ausgabemodi von `set_paint` werden die Grauwerte grafisch dargestellt (z.B. *'3d_plot'*). Dies hat zur Folge, dass auch Parameter, die mit `set_color`, `set_rgb`,`set_hsi`, `set_shape`, `set_line_width` und `set_insert` bestimmt wurden, Einfluss auf die Darstellung haben. Sollten also unerwartete Ergebnisse erscheinen, so sind die Werte der Darstellungsparameter für Regionen zu überprüfen.

Alle verfügbaren Modi können mit dem Operator `query_paint` abgefragt werden. Mögliche Werte für `Mode`:

'default' Optimale Grauwertdarstellung bei gegebener Hardware (Defaultwert). Das Verhalten ist abhängig von der Anzahl Kanäle des Eingabebildes:

- Einkanalige Bilder: Darstellung des einzigen Kanals.
- Zweikanalige Bilder: Darstellung des ersten Kanals.
- Dreikanalige Bilder: Ausgabe als RGB-Bild.

['3d_plot', Mode, Step, qa, qb, qc, qd, ZoomFac, CenterRow, CenterCol, CenterHeight] Die Grauwerte des Bildes werden als 3D-Höhenmodell interpretiert und mit Hilfe von OpenGL angezeigt. Bedeutung der einzelnen Parameter:

- *'Mode'*: Legt fest, auf welche Weise das Höhenmodell angezeigt werden soll. Folgende Modi stehen zur Verfügung:
  - *'texture'*: Das Höhenmodell wird als geschlossene Fläche dargestellt und texturiert. Die Textur wird als zweiter Kanal (für eine Grauwerttextur) oder als zweiter bis vierter Kanal (für eine Farbtextur) übergeben.
  - *'shaded'*: Das Höhenmodell wird als geschlossene Fläche dargestellt und mit der aktuellen LUT koloriert.
  - *'hidden_lines'*: Das Höhenmodell wird als Gitter dargestellt, wobei verdeckte Linien nicht dargestellt werden. Das Gitter wird mit der aktuellen LUT koloriert.
  - *'contour_lines'*: Höhenlinien werden extrahiert und dreidimensional dargestellt. Die Höhenlinien werden mit der aktuellen LUT koloriert.
- *'Step'*: Schrittweite, mit der das Bild sowohl in Zeilen- als auch in Spaltenrichtung abgetastet wird. Je kleiner dieser Parameter gewählt wird, desto feiner ist das resultierende Höhenmodell. In gleichem Maße nimmt die Berechnungszeit für die Anzeige zu. Im Falle von *'contour_lines'* entspricht dieser Parameter der Anzahl der Intervalle in die der Höhenbereich aufgeteilt wird.
- *'qa'*,*'qb'*,*'qc'*,*'qd'*: Die vier Werte eines Einheitsquaternion, welches die Orientierung des Höhenmodells beschreibt (für mehr Informationen über Quaternionen, siehe `axis_angle_to_quat`).
- *'ZoomFac'*: Zoomfaktor, wobei ein kleinerer Wert zu stärkerem Zooming führt.
- *'CenterRow'*, *'CenterCol'*, *'CenterHeight'*: Position des aktuellen Zentrums, definiert über die Zeilen, Spalten und Höhenposition. Diese Werte müssen auf den Bereich [*-0.5*,*0.5*] skaliert sein.

Automatische Parameterabschätzung: Der Wert von *'qa'* (vierter Wert des Tupels) oder *'ZoomFac'* (achter Wert des Tupels) kann auf *'auto'* gesetzt werden. Damit werden der Zoomfaktor und das Zentrum automatisch so bestimmt, dass der resultierende Plot das Fenster möglichst gut ausfüllt. Beim Setzen von *'qa'* auf *'auto'* wird die vorherige (bzw. die vorgegebene) Orientierung verwendet. Beim Setzen von *'ZoomFac'* auf *'auto'* wird die angegebene Orientierung verwendet. Es ist wichtig, dass der Bildausschnitt des Fensters mittels `set_part` korrekt gesetzt wurde, bevor `set_paint` aufgerufen wird. Es ist nicht möglich, nach *'auto'* noch Parameter im Tupel anzugeben. Außerdem funktioniert *'auto'* nur dann richtig, wenn *'scale_plot'* mittels `set_window_param` auf *'true'* gesetzt wurde. Beispiel:

```
set_paint(WindowHandle, ['3d_plot','shaded',4,'auto'])
```

Diese Darstellung erlaubt auch interaktives Betrachten des Höhenmodells. Für eine interaktive Anzeige muss die Grafikkarte genügend leistungsfähig sein und OpenGL unterstützen. Der Operator `update_window_pose` bietet einen intuitiven Weg, um obige Parameter des Höhenmodells beispielsweise mit Hilfe der Maus zu verändern.

Mit den Operatoren `set_rgb`, `set_color` und `set_colored` können die Farben für die Zeilen-, Spalten- und Höhenachse (in dieser Reihenfolge) beeinflusst werden.

Zusätzliche Parameter, die den 3D-Plot beeinflussen, können mit `set_window_param` gesetzt werden.

Falls Sie *'3d_plot'* auf einem unixartigen System nutzen wollen, muss sichergestellt sein, dass Sie ausreichende Rechte für ihre Grafikkarte besitzen.

Einschränkungen: Nur für einkanalige Bilder.

Default: *['3d_plot','shaded',8,0.939,-0.052,0.296, -0.165,0.401,0.072, -0.047,-0.073]*

['vector_field', Mode, Step, MinLength, ScaleLength, CircleSize] Ausgabe von Vektorfeldern. Bei `Mode` *'vector_field'* wird für jeden Vektor ein Kreis an der Position des Pixels sowie eine Strecke mit dem Vektor gezeichnet. Bedeutung der einzelnen Parameter:

- *'Mode'*: Sagt aus ob die visualisierten Vektorfelder absolute oder relative Koordinaten beinhalten. Mögliche Werte:
  - *'auto'*: Die Visualisierung ist abhängig vom semantischen Typ des Vektorfeldes.
  - *'absolute'*: Anzeigen von Vektorfeldern mit absoluten Koordinaten.
  - *'relative'*: Anzeigen von Vektorfeldern mit relativen Koordinaten.
- *'Step'*: Schrittweite beim Darstellen der Vektoren, d.h., Abstand zwischen den visualisierten Vektoren.
- *'MinLength'*: Minimale Länge die ein Vektor aufweisen muss um dargestellt zu werden.
- *'ScaleLength'*: Skaliert die Länge der Vektoren.
- *'CircleSize'*: Radius der dargestellten Kreise.

Es ist zu beachten, dass das Setzen von *'vector_field'* lediglich die Parameter *'Mode'*, *'Step'*, *'MinLength'*, *'ScaleLength'* und *'CircleSize'* ändert, jedoch nicht den tatsächlichen Anzeigemodus.

Vektorfeldbilder werden immer als Vektorfelder ausgegeben, unabhängig davon, welcher Modus mit `set_paint` eingestellt worden ist.

Beispiel: `set_paint(WindowHandle,['vector_field','auto',16,2,3,5])`

Dies resultiert in der Ausgabe jedes 16. Vektors der eine Länge von mindestens 2 Punkten hat. Jeder Vektor wird für die Darstellung um den Faktor 3 verlängert.

Einschränkungen: Nur für Vektorfeldbilder.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Mode`** (input_control) ..... string-array ⇝ *string* / integer / real
Name des Grauwertmodus, eventuell mit Parameter.
Defaultwert: 'default'
Werteliste: `Mode` ∈ {'default', '3d_plot', 'vector_field'}

Ergebnis

`set_paint` liefert den Wert TRUE, falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`query_paint`, `get_paint`

Nachfolger

`disp_image`

Siehe auch

`get_paint`, `query_paint`, `disp_image`, `set_shape`, `set_rgb`, `set_color`, `set_gray`, `set_window_param`, `update_window_pose`

Modul

Foundation

set_part (: : WindowHandle, Row1, Column1, Row2, Column2 :)

Verändern des darzustellenden Bildausschnitts.

set_part verändert den Bildausschnitt, der im Ausgabefenster dargestellt wird. (Row1,Column1) bezeichnet dabei den linken oberen und (Row2,Column2) den rechten unteren Eckpunkt des Ausschnitts. Er wird von allen Ausgaberoutinen sowohl für Grauwerte disp_image, disp_color als auch für Regionen (disp_region) beachtet.

Ist nur ein Teil eines Bildes darzustellen, wird dieser Ausschnitt auf die volle Fenstergröße vergrößert. Die dafür eingesetzte Interpolationsmethode kann mittels set_part_style eingestellt werden. Die Daten des Bildausschnitts liefert der Operator get_part. Wird die Fenstergröße an die Größe des anzuzeigenden Bildteils angepasst, verhindert dies eine Verlangsamung der Anzeige durch notwendige Interpolationen. Daher ist das Fenster vorzugsweise gleich groß wie der anzuzeigende Bildteil, oder ansonsten halb so groß, ein Viertel so groß, etc.

Neben dem direkten Setzen des Bildausschnitts werden die folgenden Sonderfälle unterstützt:

Row1 = Column1 = Row2 = Column2 = *-1*: Der Bildausschnitt entspricht genau der Fenstergröße, d.h. das Bild wird nicht gezoomt, gegebenenfalls aber abgeschnitten.

Row1, Column1 > *-1* **und Row2 = Column2 = *-1*:** Der Bildausschnitt entspricht der Größe des letzten dargestellten Bildes, d.h. das letzte Bild wird vollständig in dem Fenster dargestellt und dafür gegebenenfalls gezoomt.

Row1 = Column1 = *0* und Row2 = Column2 = *-2*: Der Bildausschnitt wird so angepasst, dass das letzte dargestellte Bild vollständig in das Fenster passt, wobei das Seitenverhältnis des Bildes erhalten bleibt. Dafür wird das Bild gegebenenfalls gezoomt.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- **Row1** (input_control) ..... rectangle.origin.y ⇝ *integer* / real
  Zeilennummer des linken oberen Eckpunkts des Bildausschnitts.
  Defaultwert: 0
- **Column1** (input_control) ..... rectangle.origin.x ⇝ *integer* / real
  Spaltennummer des linken oberen Eckpunkts des Bildausschnitts.
  Defaultwert: 0
- **Row2** (input_control) ..... rectangle.corner.y ⇝ *integer* / real
  Zeilennummer des rechten unteren Eckpunkts des Bildausschnitts.
  Defaultwert: -1
  Restriktion: Row2 >= Row1 || Row2 == -1 || Row1 == 0 && Row2 == -2
- **Column2** (input_control) ..... rectangle.corner.x ⇝ *integer* / real
  Spaltennummer des rechten unteren Eckpunkts des Bildausschnitts
  Defaultwert: -1
  Restriktion: Column2 >= Column1 || Column2 == -1 || Column1 == 0 && Column2 == -2

Beispiel

```
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Height-1,Width-1)
disp_image(Image,WindowHandle)
draw_rectangle1(WindowHandle,Row1,Column1,Row2,Column2)
set_part(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
```

Ergebnis

set_part liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_part`

Nachfolger

`set_part_style`, `disp_image`, `disp_region`

Alternativen

`affine_trans_image`

Siehe auch

`get_part`, `set_part_style`, `disp_region`, `disp_image`, `disp_color`

Modul

Foundation

set_part_style (: : WindowHandle, Style :)

Festlegen des Interpolationsmethode für die Grauwertdarstellung.

`set_part_style` legt die Interpolationsmethode fest, die für die Ausgabe eines Bildausschnittes im Ausgabefenster verwendet werden soll. Interpoliert werden muss immer dann, wenn das Ausgabefenster größer oder kleiner als das Bildformat bzw. als der mittels `set_part` eingestellte Bildausschnitt für die Ausgabe ist. Der Parameter wird also auch verwendet, wenn mit `set_part` kein Ausschnitt gesetzt wurde, aber das Bild für die Ausgabe verkleinert oder vergrößert werden muss (wegen der Größe des Fenster). Derzeit werden drei Modi (`Style`) unterstützt:

0 keine Interpolation (eventuell niedrigere Qualität, sehr schnell).

1 ungewichtete Interpolation zwischen den Grauwerten (mittlere Qualität und Laufzeit).

2 gewichtete Interpolation zwischen den Grauwerten (höchste Qualität, langsam).

Die aktuelle Einstellung kann mittels `get_part_style` abgefragt werden.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Style** (input_control) ..... integer ⇝ *integer*
Interpolationsmethode für die Bildschirmdarstellung: 0 (schnell,niedrige Qualität) bis 2 (langsam, höchste Qualität).
Defaultwert: 0
Werteliste: `Style` ∈ {0, 1, 2}

Ergebnis

`set_part_style` liefert den Wert TRUE, falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_part_style`

Nachfolger

`set_part`, `disp_image`, `disp_region`

Alternativen

affine_trans_image

Siehe auch

get_part_style, set_part, disp_image, disp_color

Modul

Foundation

set_rgb (: : WindowHandle, Red, Green, Blue :)

Setzen der Ausgabefarbe durch RGB-Werte.

set_rgb setzt für das Ausgabefenster die Farbe(n), bzw. die Grauwerte, in der die Regionen dargestellt werden sollen. Die gewünschten Werte werden durch die drei Farbanteile: rot, grün und blau festgelegt. Wird nur eine Kombination angegeben, dann erfolgen alle Ausgaben in dieser Farbe. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung von Tupeln von Regionen und geometrischen Objekten modulo dieser Farben.

Bei jedem Aufruf einer Ausgabeprozedur wird mit der ersten Farbe begonnen. Es ist dabei zu beachten, dass es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen. Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in der ersten Farbe. Dies gilt auch für Objekte, die aus mehreren Zusammenhangskomponenten bestehen. Die gesetzte(n) Farbe(n) wird so lange verwendet, bis erneut set_color, set_rgb oder set_gray aufgerufen wird. Verwendet wird dieser Darstellungsparameter bei Operatoren wie disp_region, disp_line, disp_rectangle1, disp_rectangle2, disp_arrow, etc.

Achtung

Falls die angegebene Farbe nicht verfügbar ist, führt dies die einem Exception. Nach dem Aufruf von set_check (::'~color':) sucht HALCON die nächste Farbe zu der angegebenen und unterdrückt den Fehler.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **Red** (input_control) . . . . . integer(-array) ⇝ *integer*
  Rotanteil der Farbe.
  Defaultwert: 255
  Typischer Wertebereich: 0 ≤ Red ≤ 255
  Restriktion: 0 <= Red && Red <= 255
- ▷ **Green** (input_control) . . . . . integer(-array) ⇝ *integer*
  Grünanteil der Farbe.
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ Green ≤ 255
  Restriktion: 0 <= Green && Green <= 255
- ▷ **Blue** (input_control) . . . . . integer(-array) ⇝ *integer*
  Blauanteil der Farbe.
  Defaultwert: 0
  Typischer Wertebereich: 0 ≤ Blue ≤ 255
  Restriktion: 0 <= Blue && Blue <= 255

Ergebnis

set_rgb liefert den Wert TRUE, falls das Fenster gültig ist und für dieses die angegebene(n) Farbe(n) darstellbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

disp_image, disp_region

Alternativen

set_hsi, set_color, set_gray

Siehe auch

disp_region

Modul

Foundation

set_rgba (: : WindowHandle, Red, Green, Blue, Alpha :)

Setzen der Ausgabefarbe durch RGBA-Werte.

set_rgba setzt für das Ausgabefenster die Farbe(n), bzw. die Grauwerte, in der die Regionen dargestellt werden sollen. Die gewünschten Werte werden durch die drei Farbanteile: rot, grün und blau und den Alphawert festgelegt. Wird nur eine Kombination angegeben, dann erfolgen alle Ausgaben in dieser Farbe. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung von Tupeln von Regionen und geometrischen Objekten modulo dieser Farben.

Bei jedem Aufruf einer Ausgabeprozedur wird mit der ersten Farbe begonnen. Es ist dabei zu beachten, dass es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen. Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in der ersten Farbe. Dies gilt auch für Objekte, die aus mehreren Zusammenhangskomponenten bestehen. Die gesetzte(n) Farbe(n) wird so lange verwendet, bis erneut set_color, set_rgb, set_rgba oder set_gray aufgerufen wird. Verwendet wird dieser Darstellungsparameter bei Operatoren wie disp_region, disp_line, disp_rectangle1, disp_rectangle2, disp_arrow, etc.

Achtung

set_rgba benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Red** (input_control) ..... integer(-array) ⇝ *integer*
  Rotanteil der Farbe.
  Defaultwert: 255
  Typischer Wertebereich: $0 \leq$ Red ≤ 255
  Restriktion: 0 <= Red && Red <= 255
- ▷ **Green** (input_control) ..... integer(-array) ⇝ *integer*
  Grünanteil der Farbe.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ Green ≤ 255
  Restriktion: 0 <= Green && Green <= 255
- ▷ **Blue** (input_control) ..... integer(-array) ⇝ *integer*
  Blauanteil der Farbe.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ Blue ≤ 255
  Restriktion: 0 <= Blue && Blue <= 255
- ▷ **Alpha** (input_control) ..... integer(-array) ⇝ *integer*
  Alphawert der Farbe.
  Defaultwert: 255
  Typischer Wertebereich: $0 \leq$ Alpha ≤ 255
  Restriktion: 0 <= Alpha && Alpha <= 255

Ergebnis

set_rgba liefert den Wert TRUE, falls das Fenster gültig ist und alle Farben verfügbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Nachfolger

disp_obj

Alternativen

set_rgb, set_color

Modul

Foundation

set_shape (: : WindowHandle, Shape :)

Festlegen der Ausgabeform für Regionen.

set_shape gibt die Gestalt an, in der Regionen gezeichnet werden sollen. Der angegebene Wert wird dem Fenster mit der logischen Fensternummer WindowHandle zugeordnet. Die Ausgabe von Regionen erfolgt mit disp_region. Alle verfügbaren Darstellungsarten können mit query_shape abgefragt werden. Modi:

'original': Die Form wird unverändert ausgegeben. Dabei können trotzdem Modifikationen durch Parameter wie set_line_width verursacht werden (die gilt genauso für die anderen Modi).

'outer_circle': Jede Region wird durch den kleinsten, umschließenden Kreis dargestellt. Siehe: smallest_circle.

'inner_circle': Jede Region wird durch den größten Umkreis dargestellt. Siehe: inner_circle.

'ellipse': Jede Region wird durch eine Ellipse mit gleichen Momenten und Orientierung angenähert. Siehe: elliptic_axis.

'rectangle1': Jede Region wird durch das kleinste umschließende Rechtecke, parallel zu den Koordinatenachsen dargestellt. Siehe: smallest_rectangle1.

'rectangle2': Jede Region wird durch das kleinste umschließende Rechteck dargestellt. Siehe: smallest_rectangle2.

'convex': Es wird die konvexe Hülle jeder Region ausgegeben. Siehe: shape_trans.

'icon' Es wird ein Icon, das mit set_icon gesetzt wurde, im Schwerpunkt der Region ausgegeben.

Achtung

Wird ein anderer Operator als disp_region verwendet, dann dürfte die Ausgabe zu verblüffenden Ergebnissen führen.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Shape** (input_control) ..... string ⇝ *string*
Art der Regionenausgabe.
Defaultwert: 'original'
Werteliste: Shape ∈ {'original', 'convex', 'outer_circle', 'inner_circle', 'rectangle1', 'rectangle2', 'ellipse', 'icon'}

Beispiel

```
read_image(Image,'fabrik')
regiongrowing(Image,Seg,5,5,6,100)
set_colored(WindowHandle,12)
set_shape(WindowHandle,'rectangle2')
disp_region(Seg,WindowHandle)
```

Ergebnis

set_shape liefert den Wert TRUE, falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_icon, query_shape, get_shape

Nachfolger

disp_region

Siehe auch

get_shape, query_shape, disp_region

Modul

Foundation

set_window_param (: : WindowHandle, Param, Value :)

Setzen von Fensterparametern.

Mit set_window_param können verschieden Parameter eines geöffneten Fensters verändert werden.

Allgemeine Fensterparameter:

'flush' Ist *'flush'* auf *'true'* gesetzt, wird der Fensterinhalt nach Anzeige jedes Objektes aktualisiert. Ist es auf *'false'* gesetzt, muss der Fensterinhalt explizit mit dem Operator flush_buffer aktualisiert werden. Wenn Sie mit dem Inhalt des Grafikfensters interagieren wollen (z. Bsp. den Inhalt zoomen oder bewegen), müssen Sie vorher *'flush'* auf *'true'* setzen.

Mögliche Werte: 'true', 'false'.

Default: 'true'.

'region_quality' Beeinflusst die Qualität der Anzeige von Regionen. Besonders wenn Regionen für die Visualisierung verkleinert werden müssen, liefert *'good'* ein genaueres und ansprechenderes Ergebnis. *'low'* ist in diesem Fall etwas ungenauer, dafür aber deutlich schneller. Wird nur der Rand von Regionen dargestellt (siehe set_draw), wird der Nachkommaanteil der Linienbreite (siehe set_line_width) ignoriert.

Mögliche Werte: 'low', 'good'.

Default: 'low'.

'background_color' Setzt die Hintergrundfarbe des Fensters WindowHandle. Der Operator clear_window benutzt diese Farbe, um das Fenster zu leeren.

Mögliche Werte: ein String mit dem Namen einer Farbe (z.B. *'black'*).

'window_title' Dieser Parameter erlaubt es, den Text in der Titelzeile eines HALCON Fensters zu ändern.

Mögliche Werte: ein String mit bis zu 1023 Zeichen.

'anti_aliasing' Ist *'anti_aliasing'* auf *'true'* gesetzt, werden Konturen und Regionen mit Antialiasing dargestellt.

Mögliche Werte: 'true', 'false'.

Default: 'true'.

'graphics_stack' Aktiviert/deaktiviert den Grafikstack des Fensters. Ist der Stack aktiviert, werden beim Ändern des Fensterausschnitts mit set_part vorher dargestellte Objekte unter Berücksichtigung des neuen Fensterausschnitts nochmals gezeichnet. Damit ist es einfach möglich den Fensterinhalt zu skalieren oder zu verschieben. Dieser Parameter hat keinen Einfluss auf den Grafikstack eines HDevelop Grafikfensters.

Mögliche Werte: 'true', 'false'.

Default: 'false'.

'graphics_stack_max_element_num' Maximale Anzahl von Elementen im Grafikstack. Das entspricht der Anzahl der Visualisierungsoperationen die im Stack gespeichert werden. Dieser Parameter hat keinen Einfluss auf den Grafikstack eines HDevelop Grafikfensters und ist erst beim Aufruf des nächsten Visualisierungsoperators wirksam.

Mögliche Werte: positive Anzahl oder *'unlimited'*.

Default: 50.

'graphics_stack_max_memory_size' Limitiert den Speicher, welchen der Grafikstack verbraucht. Wird dieses Limit überschritten werden ältere Visualisierungsoperationen aus dem Stack entfernt. Dieser Parameter hat keinen Einfluss auf den Grafikstack eines HDevelop Grafikfensters und ist erst beim Aufruf des nächsten Visualisierungsoperators wirksam.

Mögliche Werte: Speichergrenze in Bytes oder *'unlimited'*.

Default: 'unlimited'.

Parameter die den *'3d_plot'* Plot (siehe `set_paint`) betreffen:

'angle_of_view' Setzt den Öffnungswinkel der virtuellen Kamera, die für das Anzeigen des 3D-Plots verwendet wird. Der Winkel muss im Bogenmaß angegeben werden und im Intervall $[0, PI)$ liegen. Ein Winkel von 0 entspricht der Orthogonalprojektion. Es kann auch der String *'orthogonal'* übergeben werden, um Orthogonalprojektion auszuwählen.

Mögliche Werte: 'orthogonal' oder ein Winkel im Bogenmaß.

Default: 1.22173 (entspricht 70 Grad).

'axis_captions' Die Beschriftungen der Zeilen-, Spalten- und Höhenachse können gesetzt werden, indem diesem Parameter ein Tupel aus drei Strings übergeben wird. Die Reihenfolge der Strings ist Zeilenbeschriftung, Spaltenbeschriftung und Höhenbeschriftung.

Mögliche Werte: Tupel aus Strings mit bis zu 31 Zeichen.

Default: [' ',' ',' '].

'caption_color' Die Farbe der Beschriftungen der Zeilen-, Spalten- und Höhenachse. Es kann entweder eine Farbe für alle drei Achsen oder ein Tupel mit individuellen Farben für jede Achse übergeben werden.

Mögliche Werte: ein String oder ein Tupel mit drei Strings mit dem Namen einer Farbe (z.B. *'black'*).

Default: 'black'.

'plot_quality' Beeinflusst die Qualität des 3D Plots. Abhängig von der Leistungsfähigkeit der Grafikkarte kann bessere Qualität zu einem signifikanten Anstieg der benötigten Rechenzeit für den 3D Plot führen.

Mögliche Werte: 'low', *'medium'*, *'good'*, *'best'*.

Default: 'medium'.

'scale_plot' Ist *'scale_plot'* auf *'true'* gesetzt, werden die Höhenwerte eine Bildes vor dem Anzeigen in das Intervall [0,255] transformiert. Ist *'false'* gesetzt, werden die Seitenverhältnisse zwischen Spalte, Zeile und Höhe berücksichtigt. Bilder des Typs 'byte', 'cyclic' oder 'direction' werden nicht skaliert, d.h. in deren Fall wird dieser Parameter ignoriert.

Mögliche Werte: 'true', *'false'*.

Default: 'true'.

'display_grid' Ist *'display_grid'* auf *'true'* gesetzt, wird ein Gitter auf Höhe null eingeblendet.

Mögliche Werte: 'true', *'false'*.

Default: 'true'.

'display_axes' Ist *'display_axes'* auf *'true'* gesetzt, werden die Koordinatenachsen eingeblendet.

Mögliche Werte: 'true', *'false'*.

Default: 'true'.

Achtung

Die Parameter *'anti_aliasing'*, *'flush'*, *'graphics_stack'*, *'graphics_stack_max_element_num'*, *'graphics_stack_max_memory_size'* und *'region_quality'* benötigen die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Param`** (input_control) ..... string ⇝ *string*
Name des Parameters.
Defaultwert: 'flush'
Werteliste: `Param` ∈ {'plot_quality', 'axis_captions', 'caption_color', 'scale_plot', 'angle_of_view', 'display_grid', 'display_axes', 'window_title', 'background_color', 'anti_aliasing', 'flush', 'graphics_stack', 'graphics_stack_max_element_num', 'graphics_stack_max_memory_size', 'region_quality'}

▷ **Value** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
Zu setzender Wert.
Defaultwert: 'false'
Werteliste: Value ∈ {'true', 'false', 'low', 'good', 'medium', 'best', 'orthogonal', 'black', 'white', 'red', 'green', 'blue', 'unlimited'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

disp_image

Siehe auch

get_window_param, set_window_attr, set_part_style

Modul

Foundation

11.8 Text

disp_text (: : WindowHandle, String, CoordSystem, Row, Column, Color, GenParamName, GenParamValue :)

Ausgabe von Text in ein Fenster.

disp_text gibt Text im Grafikfenster WindowHandle an der Position (Row,Column) aus.

Wird nur eine Position übergeben, wird für jedes Element in String eine Textzeile ausgegeben. Außerdem wird auch *'\n'* als Zeilenumbruch interpretiert.

Werden mehrere Positionen übergeben, ist nur ein einzelner String oder ein String für jede Position zulässig. In diesem Fall kann ein Zeilenumbruch nur über *'\n'* erreicht werden.

Zeilenumbrüche (*'\n'*) am Ende von String werden ignoriert.

Die Position des Texts kann in Fensterkoordinaten (CoordSystem = *'window'*) oder in Bildkoordinaten (CoordSystem = *'image'*), was für gezoomte Bilder nützlich ist, angegeben werden.

Zusätzlich zu (Row,Column) Koordinaten können auch vordefinierte Werte an Row und Column übergeben werden um Text an festen Positionen im Fenster anzuzeigen (nur für CoordSystem = *'window'*).:

'top', *'left'*	*'top'*, *'center'*	*'top'*, *'right'*
'center', *'left'*	*'center'*, *'center'*	*'center'*, *'right'*
'bottom', *'left'*	*'bottom'*, *'center'*	*'bottom'*, *'right'*

Der Parameter Color akzeptiert auch Farben-Tupel. In diesem Fall werden die Farben zyklisch für jede neue Textposition (oder für jede Textzeile wenn nur eine Position angegeben wurde) verwendet.

Generische Parameter

disp_text kann den String innerhalb einer Box anzeigen. Dieses Verhalten und das Aussehen der Box wird mit den generischen Parametern in GenParamName und GenParamValue beeinflusst.

'box' Ist *'box'* auf *'true'* gesetzt, wird der Text innerhalb einer Box angezeigt. Die folgenden Parameter beeinflussen das Aussehen dieser Box und des optionalen Schattens der Box.
Mögliche Werte: *'true'* und *'false'*
Vorgegebener Wert: *'true'*

'box_color' Setzt die Farbe der Box.

Mögliche Werte: Ein String mit dem Namen der Farbe (z.B. *'white'*, *'red'* oder *'#aa00bba0'*)

Vorgegebener Wert: *'#fce9d4'* (ein helles Orange)

'shadow' Wird *'shadow'* auf *'true'* gesetzt, wird zusätzlich ein Schatten zu der Box angezeigt.

Mögliche Werte: *'true'* und *'false'*

Vorgegebener Wert: *'true'* wenn *'box_color'* auf eine Farbe ohne Alphawert gesetzt wurde, ansonsten *'false'*

'shadow_color' Setzt die Farbe des Schattens.

Mögliche Werte: Ein String mit dem Namen der Farbe (z.B. *'black'*, *'red'* oder *'#aa00bba0'*)

Vorgegebener Wert: *'#f28d26'* (ein dunkleres Orange) wenn *'box_color'* nicht gesetzt wurde, ansonsten *'white'*

'border_radius' Setzt die Abrundung der Ecken der Box. Für scharfe Ecken muss dieser Parameter auf *0*, für abgerundete Ecken auf höhere Werte gesetzt werden.

Mögliche Werte: positive reelle Zahl oder 0

Vorgegebener Wert: 2

'box_padding' Bestimmt, wie stark die Box über den Text hinaus geht.

Mögliche Werte: positive reelle Zahl

Vorgegebener Wert: 0

'shadow_sigma' Bestimmt, wie stark der Schatten hinter der Box weichgezeichnet wird. Für scharfe Schatten muss dieser Parameter auf 0 gesetzt werden.

Mögliche Werte: positive reelle Zahl oder 0

Vorgegebener Wert: 1.5

'shadow_dx' and 'shadow_dy' Der Offset des Schattens in Spalten- (*'shadow_dx'*) und Zeilenrichtung (*'shadow_dy'*) in Pixeln.

Mögliche Werte: reelle Zahlen

Vorgegebener Wert: 2

Achtung

`disp_text` benötigt die Bibliothek libcanvas, die auf Embedded-Systemen möglicherweise nicht verfügbar ist.

Parameter

▷ **`WindowHandle`** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **`String`** (input_control) . . . . . string(-array) ⇝ *string*
Ein Tupel von Strings mit den anzuzeigenden Texten. Jeder String wird in einer Zeile ausgegeben.
Defaultwert: 'hello'

▷ **`CoordSystem`** (input_control) . . . . . string ⇝ *string*
Koordinatensystem für die Textposition. 'window' steht für das Fensterkoordinatensystem, 'image' für das Bildkoordinatensystem.
Defaultwert: 'window'
Werteliste: `CoordSystem` ∈ {'window', 'image'}

▷ **`Row`** (input_control) . . . . . point.y(-array) ⇝ *integer* / real / string
Vertikale Textausrichtung oder Zeilenkoordinate der Textposition.
Defaultwert: 12
Werteliste: `Row` ∈ {12, 'bottom', 'center', 'top'}

▷ **`Column`** (input_control) . . . . . point.x(-array) ⇝ *integer* / real / string
Horizontale Textausrichtung oder Spaltenkoordinate der Textposition.
Defaultwert: 12
Werteliste: `Column` ∈ {12, 'center', 'left', 'right'}

▷ **`Color`** (input_control) . . . . . string(-array) ⇝ *string*
Ein Tupel von Strings für die Farben der Texte.
Defaultwert: 'black'
Werteliste: `Color` ∈ {'black', 'blue', 'yellow', 'red', 'green', 'cyan', 'magenta', 'forest green', 'lime green', 'coral', 'slate blue'}

▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'box', 'shadow', 'box_color', 'shadow_color', 'border_radius', 'box_padding', 'shadow_sigma', 'shadow_dx', 'shadow_dy'}

▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *string* / integer / real
Werte der generischen Parameter.
Defaultwert: []
Werteliste: GenParamValue ∈ {'true', 'false', 'white', 'red', 'forest green', 'black', 'blue', 5.0}

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
disp_text (WindowHandle, 'Display some text in a box', 'window', 12, 12, \
           'black', [], [])
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_font, get_string_extents

Alternativen

write_string

Siehe auch

get_string_extents, set_font

Modul

Foundation

get_font (: : WindowHandle : Font)

Abfragen des aktuellen Fonts.

get_font gibt den Namen des Fonts des Ausgabefensters WindowHandle aus, der von den Operatoren write_string, read_string etc. verwendet wird. Gesetzt wird der Parameter mit dem Operator set_font. Sowohl Textfenster als auch Bildfenster verfügen über Fonts. Beiden Fensterarten ist ein Default-Font zugeordnet, der vor dem Öffnen des Fensters durch set_system('default_font',Font) geändert werden kann. Alle verfügbaren Fonts können mit Hilfe des Operators query_font abgefragt werden.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Font** (output_control) ..... string ⇝ *string*
Name des aktuellen Fonts.

Beispiel

```
get_font(WindowHandle,CurrentFont)
set_font(WindowHandle,MyFont)
write_string(WindowHandle,['The name of my Font is:',Myfont])
new_line(WindowHandle)
set_font(WindowHandle,CurrentFont)
```

Ergebnis

`get_font` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `query_font`

Nachfolger

`set_font`

Siehe auch

`set_font`, `query_font`, `open_window`, `set_system`

Modul

Foundation

`get_font_extents` `( : : WindowHandle : MaxAscent, MaxDescent, MaxWidth, MaxHeight )`

Abfrage der Maximalgröße aller Zeichen eines Fonts.

`get_font_extents` bestimmt die maximale Breite (`MaxWidth`), Höhe (`MaxHeight`) und Ausdehnung oberhalb und unterhalb der Schreiblinie (`MaxAscent` bzw. `MaxDescent`) des Fonts, der momentan für das Fenster `WindowHandle` eingestellt ist. Die Größenangaben erfolgen in Pixelkoordinaten.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`MaxAscent`** (output_control) ..... extent.y ⇝ *integer* / real
  Maximale Höhe über der Schreiblinie.
- ▷ **`MaxDescent`** (output_control) ..... extent.y ⇝ *integer* / real
  Maximale Tiefe unter der Schreiblinie.
- ▷ **`MaxWidth`** (output_control) ..... extent.x ⇝ *integer* / real
  Maximale Zeichenbreite.
- ▷ **`MaxHeight`** (output_control) ..... extent.y ⇝ *integer* / real
  Maximale Zeichenhöhe.

Ergebnis

`get_font_extents` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_font`

Nachfolger

`set_tposition`, `write_string`, `read_string`, `read_char`

Siehe auch

`get_string_extents`, `set_tposition`, `set_font`

Modul

Foundation

get_string_extents (: : WindowHandle, Values : Ascent, Descent, Width, Height)

Größe eines Strings abfragen.

get_string_extents bestimmt die Breite und Höhe von Strings in dem Font, der momentan für das Fenster eingestellt ist. Dabei werden Strings mit mehreren Zeilen behandelt als ob sie zu einer Zeile konkateniert wären. Zusätzlich wird die Ausdehnung oberhalb und unterhalb der Schreiblinie angegeben (Ascent bzw. Descent).

Die Größenangaben erfolgen in dem Koordinatensystem des Fensters (d.h. bei Textfenstern in Pixelkoordinaten). Mit Hilfe von get_string_extents ist es möglich, Textausgabe und Texteingabe unabhängig vom Font zu programmieren. Die Umsetzung von ganzen Zahlen und Gleitpunktzahlen zu Zeichenreihen erfolgt genauso wie bei write_string.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **Values** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
  Zu untersuchende Werte.
  Defaultwert: 'test_string'
- ▷ **Ascent** (output_control) . . . . . extent.y ⇝ *integer* / real
  Maximale Höhe über der Schreiblinie.
- ▷ **Descent** (output_control) . . . . . extent.y ⇝ *integer* / real
  Maximale Tiefe unter der Schreiblinie.
- ▷ **Width** (output_control) . . . . . extent.x ⇝ *integer* / real
  Länge des Textes.
- ▷ **Height** (output_control) . . . . . extent.y ⇝ *integer* / real
  Höhe des Textes.

Ergebnis

get_string_extents liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_font

Nachfolger

set_tposition, write_string, read_string, read_char

Siehe auch

get_font_extents, set_tposition, set_font

Modul

Foundation

get_tposition (: : WindowHandle : Row, Column)

Position des Textcursors abfragen.

get_tposition liefert die momentane Position des Textcursors im Ausgabefenster. Die Positionsangabe erfolgt im Bildkoordinatensystem. Die nächste Textausgabe auf dieses Fenster erfolgt ab der Cursorposition, wobei diese die linke Ecke der Schreiblinie des auszugebenden Strings (d.h. ohne Berücksichtigung von Unterlängen) bezeichnet. Die Cursorposition verändert sich bei der Ausgabe bzw. Eingabe von Text (write_string, read_string) und durch explizite Neupositionierung (set_tposition, new_line).

Achtung

Wenn der auszugebende Text nicht vollständig in das Fenster passt, wird ein Exception ausgelöst. Dies kann durch `set_check('~text')` unterdrückt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) ..... point.y ⇝ *integer*
  Zeilenindex der Textposition.
- ▷ **Column** (output_control) ..... point.x ⇝ *integer*
  Spaltenindex der Textposition.

Ergebnis

`get_tposition` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_font`

Nachfolger

`set_tposition`, `write_string`, `read_string`, `read_char`

Siehe auch

`new_line`, `read_string`, `set_tposition`, `write_string`, `set_check`

Modul

Foundation

new_line (: : WindowHandle :)

Ausgabe eines Zeilenvorschubs im Ausgabefenster.

`new_line` setzt die Position des Textcursors im Ausgabefenster in Abhängigkeit vom momentan eingestellten Font auf den Anfang der nächsten Zeile. Die nächste Textausgabe auf dieses Fenster erfolgt ab der Cursorposition, wobei die Cursorposition die linke Ecke der Schreiblinie des auszugebenden Textes (d.h. ohne Berücksichtigung von Unterlängen) bezeichnet.

Falls im Fenster keine weitere Zeile mehr Platz hat, weil der untere Fensterrand erreicht ist, wird der Inhalt des Fensters um eine Zeilenhöhe nach oben geschoben („gescrolt“). Damit die Neupositionierung in der nächsten Zeile korrekt erfolgt, muss beim Aufruf von `new_line` schon der für die nächste Ausgabe gewünschte Font eingestellt sein. Die Cursorposition verändert sich bei der Ausgabe bzw. Eingabe von Text (`write_string`, `read_string`) und durch explizite Neupositionierung (`set_tposition`).

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.

Ergebnis

`new_line` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_font, write_string

Alternativen

get_tposition, get_string_extents, set_tposition

Siehe auch

write_string, set_font

Modul

Foundation

query_font (: : WindowHandle : Font)

Abfragen der verfügbaren Fonts.

query_font gibt für das Ausgabefenster die Namen aller Fonts aus, die für die Textausgabe verwendet werden können. Diese können mit der entsprechenden Syntax von dem Operator set_font gesetzt werden. Verwendet werden die Fonts von write_string, read_char, read_string und new_line.

Achtung

Die Fonts unterscheiden sich oft stark auf verschiedenen Rechnern. Man muss i.a. davon ausgehen, dass query_font auf verschiedenen Rechnern auch verschiedene Fonts liefert.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Font** (output_control) ..... string-array ⇝ *string*
Tupel mit verfügbaren Fontnamen.

Beispiel

```
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
set_check('~text')
query_font(WindowHandle,Fontlist)
set_color(WindowHandle,'white')
for i:=0 to |Fontlist|-1 by 1
  set_display_font (WindowHandle,16,Fontlist[i],'true','false')
  write_string(WindowHandle,Fontlist[i])
  new_line(WindowHandle)
endfor
```

Ergebnis

query_font liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

set_font, write_string, read_string, read_char

Siehe auch

set_font, write_string, read_string, read_char, new_line

Modul

Foundation

read_char (: : WindowHandle : Char, Code)

Einlesen eines Zeichens aus einem Fenster.

read_char liest im Eingabefenster (= Ausgabefenster) ein Zeichen von der Tastatur ein. Falls es sich um ein druckbares Zeichen handelt, wird dieses in Char zurückgegeben. Falls eine Steuertaste gedrückt wurde, ist dies am Wert von Code erkennbar. Einige wichtige Tasten sind dadurch identifizierbar. Mögliche Werte für Code sind:

'character': druckbares Zeichen

'left': Cursor links

'right': Cursor rechts

'up': Cursor oben

'down': Cursor unten

'insert': Einfügetaste

'none': keine der angeführten Tasten

'cancelled': read_char wurde abgebrochen (nur unter Windows möglich)

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Char** (output_control) . . . . . string ⇝ *string*
Eingelesenes Zeichen (falls kein Steuerzeichen).

▷ **Code** (output_control) . . . . . string ⇝ *string*
Code für eingelesenes Zeichen.

Ergebnis

read_char liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_font

Alternativen

read_string, fread_char, fread_string

Siehe auch

write_string, set_font

Modul

Foundation

read_string (: : WindowHandle, InString, Length : OutString)

Einlesen eines Strings in einem Textfenster.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit dev_open_window geöffnet wurden.

read_string liest im Eingabefenster (= Ausgabefenster) einen String mit vorgegebener Maximallänge (Length) von der Tastatur ein. Der String wird ab der momentanen Textcursorposition des Fensters unter Verwendung des eingestellten Fonts eingelesen. Die Maximallänge (Length) muss so gewählt sein, dass der rechte Rand des Fensters nicht überschritten wird. Der Operator kann ein Default-String (InString) übergeben werden, der vom Benutzer editiert oder direkt übernommen werden kann. Der Textcursor befindet sich nach Beendigung der Eingabe am Ende des editierten Strings. Editiermöglichkeiten:

RETURN Eingabe abschließen

BACKSPACE Zeichen links vom Cursor löschen und Textcursor um eine Position nach links.

Die Länge wird in Anzahl Zeichen angegeben. Wenn *'filename_encoding'* mit `set_system` auf *'locale'* gesetzt ist, bestimmt die Länge die Anzahl der Bytes.

Parameter

- ▷ **`WindowHandle`** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **`InString`** (input_control) . . . . . string ⇝ *string*
  String, der vor dem Einlesen bereits sichtbar ist.
  Defaultwert: ”
- ▷ **`Length`** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl von Zeichen.
  Defaultwert: 32
  Restriktion: Length > 0 && Length <= 1024
- ▷ **`OutString`** (output_control) . . . . . string ⇝ *string*
  Eingelesener String.

Ergebnis

`read_string` liefert den Wert TRUE, falls das Textfenster gültig ist und ein String mit maximaler Länge bis zum rechten Fensterrand Platz hat. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_font`

Alternativen

`read_char`, `fread_string`, `fread_char`

Siehe auch

`set_tposition`, `new_line`, `open_window`, `set_font`, `set_color`

Modul

Foundation

`set_font ( : : WindowHandle, Font : )`

Setzen eines Fonts.

`set_font` setzt den Font `Font` für das Ausgabefenster `WindowHandle`. Alle nachfolgenden Textausgaben in diesem Fenster (z.B. mit den Operatoren `disp_text`, `write_string` oder `read_string`) verwenden nun den neuen Font anstelle des Default-Fonts (siehe `set_system('default_font',Font)`). Alle verfügbaren Fonts können mit Hilfe des Operators `query_font` abgefragt werden.

Die Syntax für die Angabe eines Fonts `Font` ist folgende:

```
FONTNAME[-STYLE]-FONT\_SIZE.
```

Der optionale STYLE kann unter anderem einen der folgenden Werte annehmen:

- Normal,
- Bold,
- BoldItalic oder

- Italic.

Die Fontgröße FONT_SIZE wird in Pixel bemessen.

Ein Beispiel für einen gültigen String für `Font` ist

'Courier-Bold-14'.

Achtung

Die verfügbaren Fonts unterscheiden sich stark bei verschiedenen Rechnern. Es empfiehlt sich daher mit `query_font` oder der Prozedur `set_display_font` zu arbeiten.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Font`** (input_control) ..... string ⇝ *string*
Name des Fonts, der gesetzt wird.

Beispiel

```
dev_get_window (WindowHandle)
query_font (WindowHandle, Font)
* Specify font name and size
FontWithSize := Font[0]+'-20'
set_font (WindowHandle, FontWithSize)
dev_disp_text ('Font set to: '+FontWithSize, 'window', 20, 12, 'black', [], [])
* Specify font name, style, and size
FontWithStyleAndSize := Font[0]+'-Bold-20'
set_font (WindowHandle, FontWithStyleAndSize)
dev_disp_text ('Font set to: '+FontWithStyleAndSize, 'window', 50, 12, 'black', [], [])
```

Ergebnis

`set_font` liefert den Wert TRUE, falls der Font-Name aufgelöst werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`query_font`

Siehe auch

`get_font`, `query_font`, `open_window`

Modul

Foundation

`set_tposition ( : : WindowHandle, Row, Column : )`

Setzen der Schreibposition im einem Fenster.

`set_tposition` setzt die Position des Textcursors im Ausgabefenster. Als Referenz wird das linke obere Eck eines Großbuchstabens verwendet.

Die Positionsangabe erfolgt im Bildkoordinatensystem. Die Position des Textcursors wird z.B. durch einen Unterstrich markiert. Die nächste Textausgabe auf dieses Fenster erfolgt ab der Cursorposition, wobei die Position des

Textcursors die linke Ecke der Schreiblinie des auszugebenden Strings (d.h. ohne Berücksichtigung von Unterlängen, die Zeichenhöhe über der Schreiblinie sei als Fontoberlänge bezeichnet) angibt.

Die Cursorposition verändert sich bei der Ausgabe bzw. Eingabe von Text (write_string, read_string) und durch explizite Neupositionierung (set_tposition, new_line). Um die Anzeige der Cursorposition im Fenster zu löschen, kann die Form des Textcursors mit set_tshape auf 'invisible' gesetzt werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (input_control) . . . . . point.y ⇝ *integer*
  Zeilenindex der Schreibposition.
  Defaultwert: 24
- ▷ **Column** (input_control) . . . . . point.x ⇝ *integer*
  Spaltenindex der Schreibposition.
  Defaultwert: 12

Ergebnis

set_tposition liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

write_string, read_string

Alternativen

new_line

Siehe auch

read_string, write_string

Modul

Foundation

write_string (: : WindowHandle, String :)

Ausgabe von Text in ein Fenster.

write_string gibt im Ausgabefenster einen String aus, an der momentanen Textcursorposition. Der auszugebende Text darf nicht länger als der bis zum rechten Fensterrand verbleibende Platz sein (gegebenenfalls kann mit Hilfe des Operators get_string_extents die Länge des Strings abgefragt werden).

Bei der Ausgabe wird der gerade für das Fenster eingestellte Font verwendet. Der Textcursor steht im Anschluss an die Ausgabe hinter dem geschriebenen Text.

write_string kann alle drei Datentypen ausgeben. Die Konversion in eine Zeichenreihe erfolgt nach folgenden Regeln:

- Strings werden unverändert übernommen.
- Ganze Zahlen werden ohne Leerzeichen (vor, bzw. hinter der Zahl) konvertiert.
- Gleitpunktzahlen werden, soweit möglich mit einem Gleitpunkt und ohne Exponent ausgegeben.
- Die sich ergebenden Zeichenreihen werden ohne Leerzeichen hintereinander ausgegeben.

Für die Pufferung von Texten siehe set_system mit dem Flag 'flush_graphic'.

Achtung

Falls ein Clipping am Fensterrand erwünscht ist, kann die Fehlermeldung mit set_check('~text') ausgeschaltet werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **String** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Tupel von auszugebenden Werten (beliebige Typen).
  Defaultwert: 'hello'

Ergebnis

write_string liefert den Wert TRUE, falls das Fenster gültig ist und der auszugebende Text in der aktuellen Zeile noch Platz hat (siehe set_check). Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_font, get_string_extents

Alternativen

fwrite_string

Siehe auch

set_tposition, get_string_extents, set_font, set_system, set_check

Modul

Foundation

11.9 Zeichnen

In diesem Kapitel werden Operatoren beschrieben, die zum manuellen Erstellen geometrischer Formen genutzt werden können. Diese benötigen die Interaktion des Benutzers über die Maustasten und blockieren die Anwendung bis die Interaktion abgeschlossen ist. Die in Graphik / Objekt aufgelisteten Operatoren ermöglichen es geometrische Formen zu erzeugen, ohne die Anwendung zu blockieren.

Allgemein

Die Operatoren draw_region, draw_xld, draw_xld_mod, draw_nurbs, draw_nurbs_mod, draw_nurbs_interp, draw_nurbs_interp_mod sowie drag_region1, drag_region2 und drag_region3 geben als Ergebnis ikonische Objekte aus. Die restlichen Zeichenoperatoren geben geometrische Parameter zurück, die zur Erstellung von Regionen oder Konturen in der jeweiligen Form notwendig sind (vgl. Tabelle). Für die meisten Zeichenoperatoren existiert zudem ein Modifikationsoperator, mit dem die zunächst bestimmte Form nochmals angepasst werden kann.

Zeichnen in Pufferfenstern

Da für Zeichenoperatoren eine Interaktion des Nutzers über Maustasten nötig ist, ist die Verwendung in den unsichtbaren Pufferfenstern (siehe open_window) an sich nicht möglich. Bei Verwendung solcher Fenster sollten stattdessen Zeichenobjekte (siehe Graphik / Objekt) genutzt werden. Einige Zeichenoperatoren können dennoch in Pufferfenstern verwendet werden. Mit Hilfe der Operatoren send_mouse_double_click_event, send_mouse_down_event, send_mouse_drag_event und send_mouse_up_event müssen die Mausereignisse und der Zustand der Zusatztasten (Strg, Alt, Umschalttaste) gesendet werden, während der Zeichenoperator in einem anderen Thread aufgerufen werden muss. Der Bildausschnitt darf währenddessen nicht verändert werden.

Zeichenoperatoren	Ausgabe	Objekt generieren	Funktioniert in Puffer-Fenstern
draw_point	Row, Column	gen_cross_contour_xld	ja
draw_line	Row1, Column1, Row2, Column2	gen_contour_polygon_xld	ja
draw_rectangle1	Row1, Column1, Row2, Column2	gen_rectangle1	ja
draw_rectangle2	Row, Column, Phi, Length1, Length2	gen_rectangle2, gen_rectangle2_contour_xld	ja
draw_circle	Row, Column, Radius	gen_circle, gen_circle_contour_xld	ja
draw_ellipse	Row, Column, Phi, Radius1, Radius2	gen_ellipse, gen_ellipse_contour_xld	ja
draw_xld	Contour	-	ja
draw_region	Region	-	ja
draw_polygon	Region	-	ja
draw_nurbs	Contour, Rows, Columns, Weights	-	nein
draw_nurbs_interp	Contour, Rows, Columns, Knots, Tangents	-	nein
drag_region1	Region	-	nein
drag_region2	Region	-	nein
drag_region3	Region	-	nein

drag_region1 (SourceRegion : DestinationRegion : WindowHandle :)

Interaktives Verschieben einer Region.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit dev_open_window geöffnet wurden.

drag_region1 dient dazu, die Region mit der Maus auf dem Bildschirm zu verschieben. Nach dem Aufruf von drag_region1 wird die Region sichtbar, sobald die linke Maustaste betätigt wird. Es wird dabei nur der Rand der Region dargestellt. Als Darstellungsmodus wird für die Dauer der Prozedur der Modus 'not' (siehe set_draw) verwendet. Der Mauscursor befindet sich während des Verschiebens im Schwerpunkt der Region. Wird die Maus mit gedrückter linker Maustaste verschoben, folgt die dargestellte Region - verzögert - dieser Bewegung. Wird die rechte Maustaste gedrückt, so wird drag_region1 beendet. Unter macOS kann drag_region1 auch durch Drücken der Escape-Taste beendet werden. Die Darstellung der Region auf dem Bildschirm verschwindet. Ausgegeben wird eine Region, die der letzten Position auf dem Bildschirm entspricht. Es können auch mehrere Regionen auf einmal übergeben werden. Die Grauwerte können mit dem Operator affine_trans_image verschoben werden.

Achtung

Die Grauwerte der Regionen werden nicht verschoben. Da die Eingaberegion verschoben wird, ist nicht sichergestellt, dass die Grauwerte der Ausgaberegionen sinnvoll besetzt sind. Dieser Fall kann eintreten, wenn die Grauwerte der Eingaberegionen nicht das ganze Bild umfassen.

Parameter

- ▷ **SourceRegion** (input_object) ..... region-array ⇝ *object*
  Zu verschiebende Regionen.
- ▷ **DestinationRegion** (output_object) ..... region-array ⇝ *object*
  Verschobene Regionen.

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

Beispiel

```
draw_region(Obj,WindowHandle)
dev_set_color ('green')
drag_region1(Obj,New,WindowHandle)
dev_display(New)
get_region_runs (Obj, Rows1, ColumnBegins1, ColumnEnds1)
get_region_runs (New, Rows2, ColumnBegins2, ColumnEnds2)
Row1 := Rows1[0]
Column1 := ColumnBegins1[0]
Row2 := Rows2[0]
Column2 := ColumnBegins2[0]
dev_set_color ('white')
gen_arrow_contour_xld (Arrow, Row1,Column1,Row2,Column2, 5, 5)
dev_display (Arrow)
```

Ergebnis

drag_region1 liefert den Wert TRUE, falls eine Region eingegeben wird, das Fenster gültig ist und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt. Das Verhalten bei einer leeren Eingabe kann mit set_system(::'no_object_result',<Result>:) bestimmt werden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

get_mposition, move_region

Siehe auch

set_insert, set_draw, affine_trans_image

Modul

Foundation

drag_region2 (SourceRegion : DestinationRegion : WindowHandle, Row, Column :)

Interaktives Verschieben einer Region mit Angabe des Fixpunktes.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit dev_open_window geöffnet wurden.

drag_region2 dient dazu, die Region mit der Maus auf dem Bildschirm zu verschieben. Es entspricht dem Operator drag_region1, mit dem Unterschied, dass hier die Position des Maus-Cursors angegeben werden kann.

Achtung

Die Grauwerte der Regionen werden nicht verschoben. Da die Eingaberegionen verschoben wird, ist nicht sichergestellt, dass die Grauwerte der Ausgaberegionen sinnvoll besetzt sind. Dieser Fall kann eintreten, wenn die Grauwerte der Eingaberegionen nicht das ganze Bild umfassen.

Parameter

▷ **SourceRegion** (input_object) ..... region-array ⇝ *object*
Zu verschiebende Regionen.

▷ **DestinationRegion** (output_object) ..... region-array ⇝ *object*
Verschobene Regionen.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Bezugspunktes.
Defaultwert: 100
Wertevorschläge: `Row` ∈ {0, 64, 128, 256, 512}
Typischer Wertebereich: 0 ≤ `Row` ≤ 1024

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Bezugspunktes.
Defaultwert: 100
Wertevorschläge: `Column` ∈ {0, 64, 128, 256, 512}
Typischer Wertebereich: 0 ≤ `Column` ≤ 1024

Ergebnis

`drag_region2` liefert den Wert TRUE, falls eine Region eingegeben wird, das Fenster gültig ist und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt. Das Verhalten bei einer leeren Eingabe kann mit `set_system(::'no_object_result',<Result>:)` bestimmt werden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`, `affine_trans_image`

Alternativen

`get_mposition`, `move_region`, `drag_region1`, `drag_region3`

Siehe auch

`set_insert`, `set_draw`, `affine_trans_image`

Modul

Foundation

drag_region3 (SourceRegion, MaskRegion : DestinationRegion : WindowHandle, Row, Column :)

Interaktives Verschieben einer Region mit der Beschränkung der Positionen.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit `dev_open_window` geöffnet wurden.

`drag_region3` dient dazu, die Region mit der Maus auf dem Bildschirm zu verschieben. Dies entspricht dem Operator `drag_region2`, mit der Erweiterung, dass hier auch noch alle Punkte angegeben werden, auf denen sich die Maus bewegen darf. Wird die Maus außerhalb dieses Bereiches (`MaskRegion`) geführt, dann wird die Region auf dem Punkt mit dem geringsten Abstand zur Maus innerhalb von `MaskRegion` dargestellt.

Achtung

Die Grauwerte der Regionen werden nicht verschoben. Da die Eingaberegion verschoben wird, ist nicht sichergestellt, dass die Grauwerte der Ausgaberegionen sinnvoll besetzt sind. Dieser Fall kann eintreten, wenn die Grauwerte der Eingaberegionen nicht das ganze Bild umfassen.

Parameter

▷ **SourceRegion** (input_object) ..... region-array ⇝ *object*
Zu verschiebende Regionen.

▷ **MaskRegion** (input_object) ..... region-array ⇝ *object*
Punkte auf denen sich die Region bewegen darf.

▷ **DestinationRegion** (output_object) ..... region-array ⇝ *object*
Verschobene Regionen.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Bezugspunktes.
Defaultwert: 100
Wertevorschläge: `Row` ∈ {0, 64, 128, 256, 512}
Typischer Wertebereich: 0 ≤ `Row` ≤ 1024

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Bezugspunktes.
Defaultwert: 100
Wertevorschläge: `Column` ∈ {0, 64, 128, 256, 512}
Typischer Wertebereich: 0 ≤ `Column` ≤ 1024

Ergebnis

`drag_region3` liefert den Wert TRUE, falls eine Region eingegeben wird, das Fenster gültig ist und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt. Das Verhalten bei einer leeren Eingabe kann mit `set_system(::'no_object_result',<Result>:)` bestimmt werden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `get_mposition`

Nachfolger

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`, `affine_trans_image`

Alternativen

`get_mposition`, `move_region`, `drag_region1`, `drag_region2`

Siehe auch

`set_insert`, `set_draw`, `affine_trans_image`

Modul

Foundation

draw_circle (: : WindowHandle : Row, Column, Radius)

Interaktives Erstellen eines Kreises.

`draw_circle` liefert die Parameter für einen Kreis, der interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Kreises wird mit der linken Maustaste an die Stelle geklickt, die als Mittelpunkt für den Kreis verwendet werden soll. Während die Taste gedrückt bleibt, kann die Länge des `Radius` durch Verschieben der Maus verändert werden. Nach einem erneuten Mausklick in die Mitte des erzeugten Kreises, kann dieser beliebig verschoben werden. Mit einem Klick in die Nähe des Kreisbogens kann der Kreis „angefasst“ werden, um den `Radius` zu verändern. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann

draw_circle auch durch Drücken der Escape-Taste beendet werden. Der Kreis ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) ..... circle.center.y ⇝ *real*
  Zeilenindex des Mittelpunkts.
- ▷ **Column** (output_control) ..... circle.center.x ⇝ *real*
  Spaltenindex des Mittelpunkts.
- ▷ **Radius** (output_control) ..... circle.radius ⇝ *real*
  Radius des Kreises.

Beispiel

```
read_image(Image,'monkey')
draw_circle(WindowHandle,Row,Column,Radius)
gen_circle(Circle,Row,Column,Radius)
reduce_domain(Image,Circle,ImageReduced)
invert_image (ImageReduced, ImageInvert)
dev_display (ImageInvert)
```

Ergebnis

draw_circle liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_circle_mod, draw_ellipse, draw_region

Siehe auch

gen_circle, draw_rectangle1, draw_rectangle2, draw_polygon, set_insert

Modul

Foundation

draw_circle_mod (: : WindowHandle, RowIn, ColumnIn, RadiusIn : Row, Column, Radius)

Interaktives Erstellen eines Kreises.

draw_circle_mod liefert die Parameter für einen Kreis, der interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Kreises werden die Koordinaten RowIn und ColumnIn des Mittelpunktes des Kreises mit Radius RadiusIn erwartet. Nach einem erneuten Mausklick in die Mitte des erzeugten Kreises, kann dieser beliebig verschoben werden. Mit einem Klick in die Nähe des Kreisbogens kann der Kreis „angefasst“ werden,

um den Radius zu verändern. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann draw_circle_mod auch durch Drücken der Escape-Taste beendet werden. Der Kreis ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **RowIn** (input_control) ..... circle.center.y ⇝ *real*
  Zeilenindex des Mittelpunktes.
- ▷ **ColumnIn** (input_control) ..... circle.center.x ⇝ *real*
  Spaltenindex des Mittelpunktes.
- ▷ **RadiusIn** (input_control) ..... circle.radius ⇝ *real*
  Radius des Kreises.
- ▷ **Row** (output_control) ..... circle.center.y ⇝ *real*
  Zeilenindex des Mittelpunktes.
- ▷ **Column** (output_control) ..... circle.center.x ⇝ *real*
  Spaltenindex des Mittelpunktes.
- ▷ **Radius** (output_control) ..... circle.radius ⇝ *real*
  Radius des Kreises.

Beispiel

```
read_image(Image,'monkey')
draw_circle_mod(WindowHandle,20,20,15,Row,Column,Radius)
gen_circle(Circle,Row,Column,Radius)
reduce_domain(Image,Circle,ImageReduced)
invert_image (ImageReduced, ImageInvert)
dev_display (ImageInvert)
```

Ergebnis

draw_circle_mod liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_circle, draw_ellipse, draw_region

Siehe auch

gen_circle, draw_rectangle1, draw_rectangle2, draw_polygon, set_insert

Modul

Foundation

draw_ellipse (: : WindowHandle : Row, Column, Phi, Radius1, Radius2)

Interaktives Erstellen einer Ellipse.

draw_ellipse liefert die Parameter für eine beliebig orientierte Ellipse, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Die zu erzeugende Ellipse wird durch ihren Mittelpunkt, ihre zwei Halbachsen und den Winkel der ersten Halbachse zur horizontalen Koordinatenachse beschrieben.

Für die Erzeugung der Ellipse wird mit der linken Maustaste der Mittelpunkt der Ellipse festgelegt. Während die Taste gedrückt bleibt, kann die Länge (Radius1) und die Orientierung (Phi) der ersten Halbachse festgelegt werden. Dabei wird für die zweite Halbachse vorläufig eine Default-Länge verwendet, die anschließend aber noch verändert werden kann. Nach einem erneuten Mausklick in die Mitte der erzeugten Ellipse, kann diese beliebig verschoben werden. Mit einem Klick in die Nähe eines Scheitels kann dieser „angefasst“ werden, um die Länge der zugehörigen Halbachse zu verändern. Die Orientierung kann nur verändert werden, wenn ein Scheitel der ersten Halbachse angefasst wird.

Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann draw_ellipse auch durch Drücken der Escape-Taste beendet werden. Die Ellipse ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) ..... ellipse.center.y ⇝ *real*
  Zeilenindex des Mittelpunktes.
- ▷ **Column** (output_control) ..... ellipse.center.x ⇝ *real*
  Spaltenindex des Mittelpunktes.
- ▷ **Phi** (output_control) ..... ellipse.angle.rad ⇝ *real*
  Orientierung der ersten Halbachse in Bogenmaß.
- ▷ **Radius1** (output_control) ..... ellipse.radius1 ⇝ *real*
  Erste Halbachse.
- ▷ **Radius2** (output_control) ..... ellipse.radius2 ⇝ *real*
  Zweite Halbachse.

Beispiel

```
read_image(Image,'monkey')
draw_ellipse(WindowHandle,Row,Column,Phi,Radius1,Radius2)
gen_ellipse(Ellipse,Row,Column,Phi,Radius1,Radius2)
reduce_domain(Image,Ellipse,GrayEllipse)
sobel_amp(GrayEllipse,Sobel,'sum_abs',3)
dev_display(Sobel)
```

Ergebnis

draw_ellipse liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_ellipse_mod, draw_circle, draw_region

Siehe auch

gen_ellipse, draw_rectangle1, draw_rectangle2, draw_polygon, set_insert

Modul

Foundation

draw_ellipse_mod (: : WindowHandle, RowIn, ColumnIn, PhiIn, Radius1In, Radius2In : Row, Column, Phi, Radius1, Radius2)

Interaktives Erstellen einer Ellipse.

draw_ellipse_mod liefert die Parameter für eine beliebig orientierte Ellipse, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Die zu erzeugende Ellipse wird durch ihren Mittelpunkt, ihre zwei Halbachsen und den Winkel der ersten Halbachse zur horizontalen Koordinatenachse beschrieben.

Für die Erzeugung der Ellipse werden die Parametern RowIn, ColumnIn,PhiIn,Radius1In,Radius2In erwartet. Während die Taste gedrückt bleibt, kann die Länge (Radius1) und die Orientierung (Phi) der ersten Halbachse festgelegt werden. Dabei wird für die zweite Halbachse vorläufig eine Default-Länge verwendet, die anschließend aber noch verändert werden kann. Nach einem erneuten Mausklick in die Mitte der erzeugten Ellipse, kann diese beliebig verschoben werden. Mit einem Klick in die Nähe eines Scheitels kann dieser „angefasst“ werden, um die Länge der zugehörigen Halbachse zu verändern. Die Orientierung kann nur verändert werden, wenn ein Scheitel der ersten Halbachse angefasst wird.

Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann draw_ellipse_mod auch durch Drücken der Escape-Taste beendet werden. Die Ellipse ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **RowIn** (input_control) . . . . . ellipse.center.y ⇝ *real*
  Zeilenindex des Mittelpunkts.
- ▷ **ColumnIn** (input_control) . . . . . ellipse.center.x ⇝ *real*
  Spaltenindex des Mittelpunkts.
- ▷ **PhiIn** (input_control) . . . . . ellipse.angle.rad ⇝ *real*
  Orientierung der größeren Halbachse in Bogenmaß.
- ▷ **Radius1In** (input_control) . . . . . ellipse.radius1 ⇝ *real*
  Größere Halbachse.
- ▷ **Radius2In** (input_control) . . . . . ellipse.radius2 ⇝ *real*
  Kleinere Halbachse.
- ▷ **Row** (output_control) . . . . . ellipse.center.y ⇝ *real*
  Zeilenindex des Mittelpunktes.
- ▷ **Column** (output_control) . . . . . ellipse.center.x ⇝ *real*
  Spaltenindex des Mittelpunktes.
- ▷ **Phi** (output_control) . . . . . ellipse.angle.rad ⇝ *real*
  Orientierung der ersten Halbachse in Bogenmaß.
- ▷ **Radius1** (output_control) . . . . . ellipse.radius1 ⇝ *real*
  Erste Halbachse.
- ▷ **Radius2** (output_control) . . . . . ellipse.radius2 ⇝ *real*
  Zweite Halbachse.

Beispiel

```
read_image(Image,'monkey')
```

```
draw_ellipse_mod(WindowHandle,RowIn,ColumnIn,PhiIn,Radius1In,Radius2In,\
                 Row,Column,Phi,Radius1,Radius2)
gen_ellipse(Ellipse,Row,Column,Phi,Radius1,Radius2)
reduce_domain(Image,Ellipse,GrayEllipse)
sobel_amp(GrayEllipse,Sobel,'sum_abs',3)
dev_display(Sobel)
```

Ergebnis

`draw_ellipse_mod` liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_ellipse`, `draw_circle`, `draw_region`

Siehe auch

`gen_ellipse`, `draw_rectangle1`, `draw_rectangle2`, `draw_polygon`, `set_insert`

Modul

Foundation

`draw_line ( : : WindowHandle : Row1, Column1, Row2, Column2 )`

Zeichnen einer Gerade.

`draw_line` liefert die Parameter für eine Gerade, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung der Gerade wird mit der linken Maustaste für einen Startpunkt der Gerade geklickt. Während die Taste gedrückt bleibt, kann die Gerade in jede beliebige Richtung „aufgezogen“ werden. Nach einem erneuten Mausklick in die Mitte der erzeugten Gerade, kann diese beliebig verschoben werden. Klickt man auf einen Endpunkt der erzeugten Gerade, kann diesen Punkt verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_line` auch durch Drücken der Escape-Taste beendet werden.

Die Gerade ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`Row1`** (output_control) ..... line.begin.y ⇝ *real*
  Zeilenindex des ersten Punktes der Gerade.
- ▷ **`Column1`** (output_control) ..... line.begin.x ⇝ *real*
  Spaltenindex des ersten Punktes der Gerade.
- ▷ **`Row2`** (output_control) ..... line.end.y ⇝ *real*
  Zeilenindex des zweiten Punktes der Gerade.
- ▷ **`Column2`** (output_control) ..... line.end.x ⇝ *real*
  Spaltenindex des zweiten Punktes der Gerade.

Beispiel

```
draw_line(WindowHandle,Row1,Column1,Row2,Column2)
gen_contour_polygon_xld (Line, [Row1,Row2], [Column1,Column2])
dev_display (Line)
```

Ergebnis

`draw_line` liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`reduce_domain`, `disp_line`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`

Siehe auch

`draw_line_mod`, `gen_rectangle1`, `draw_circle`, `draw_ellipse`, `set_insert`

Modul

Foundation

`draw_line_mod` `( : : WindowHandle, Row1In, Column1In, Row2In, Column2In : Row1, Column1, Row2, Column2 )`

Zeichnen einer Gerade.

`draw_line_mod` liefert die Parameter für eine Gerade, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung der Gerade werden die Koordinaten `Row1In`, `Column1In` des Startpunktes und `Row2In`,`Column2In` des Endpunktes erwartet. Klickt man auf einen Endpunkt der erzeugten Gerade, kann diesen Punkt verschoben werden. Nach einem erneuten Mausklick in die Mitte der erzeugten Gerade, kann diese beliebig verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_line_mod` auch durch Drücken der Escape-Taste beendet werden.

Die Gerade ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`Row1In`** (input_control) ..... line.begin.y ⇝ *real*
  Zeilenindex des ersten Punktes der Gerade.
- ▷ **`Column1In`** (input_control) ..... line.begin.x ⇝ *real*
  Spaltenindex des ersten Punktes der Gerade.
- ▷ **`Row2In`** (input_control) ..... line.end.y ⇝ *real*
  Zeilenindex des zweiten Punktes der Gerade.
- ▷ **`Column2In`** (input_control) ..... line.end.x ⇝ *real*
  Spaltenindex des zweiten Punktes der Gerade.
- ▷ **`Row1`** (output_control) ..... line.begin.y ⇝ *real*
  Zeilenindex des ersten Punktes der Gerade.

▷ **Column1** (output_control) ..... line.begin.x ⇝ *real*
Spaltenindex des ersten Punktes der Gerade.

▷ **Row2** (output_control) ..... line.end.y ⇝ *real*
Zeilenindex des zweiten Punktes der Gerade.

▷ **Column2** (output_control) ..... line.end.x ⇝ *real*
Spaltenindex des zweiten Punktes der Gerade.

Beispiel

```
draw_line_mod(WindowHandle,10,20,55,124,Row1,Column1,Row2,Column2)
gen_contour_polygon_xld (Line, [Row1,Row2], [Column1,Column2])
dev_display (Line)
```

Ergebnis

`draw_line_mod` liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`reduce_domain`, `disp_line`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_line`, `draw_ellipse`, `draw_region`

Siehe auch

`gen_circle`, `draw_rectangle1`, `draw_rectangle2`

Modul

Foundation

draw_nurbs (: ContOut : WindowHandle, Rotate, Move, Scale, KeepRatio, Degree : Rows, Cols, Weights)

Interaktives Erstellen einer NURBS-Kurve.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit `dev_open_window` geöffnet wurden.

`draw_nurbs` liefert die Kontur `ContOut` und Kontrollinformationen (`Rows`, `Cols` und `Weights`) einer NURBS-Kurve, die interaktiv durch den Benutzer im Fenster `WindowHandle` erzeugt worden ist. Zusätzliche Informationen über NURBS-Kurven finden sich in der Dokumentation des Operators `gen_contour_nurbs_xld`. Um die Kontrollinformationen `Rows`, `Cols` und `Weights` in einem Aufruf des Operators `gen_contour_nurbs_xld` zu verwenden, muss dort der Knotenvektor `Knots` auf *'auto'* gesetzt werden.

Die NURBS-Kurve wird durch Manipulation ihres Kontrollpolygons gezeichnet. Dagegen wird die NURBS-Kurve mit dem Operator `draw_nurbs_interp` durch Interpolation von Stützstellen (die interaktiv manipuliert werden können) erzeugt. Direkt nach dem Aufruf von `draw_nurbs` kann man durch Drücken der linken Maustaste Punkte zum Kontrollpolygon hinzufügen. Der zuletzt angehängte Punkt kann durch Drücken der Steuerungstaste (Strg, Ctrl) gelöscht werden. Sobald das Kontrollpolygon aus 3 Punkten besteht, werden der erste und der letzte Punkt mit einem zusätzlichen Quadrat markiert. Durch Klicken auf den ersten oder letzten Kontrollpunkt wird die Kurve geschlossen bzw. wieder geöffnet. Sobald mehr als `Degree` Kontrollpunkte vorhanden sind, wird die NURBS-Kurve angezeigt, die durch das Kontrollpolygon und die Gewichte bestimmt ist.

Der zuletzt angefasste Kontrollpunkt wird von einem Kreis umgeben, der das Gewicht dieses Punktes visualisiert. Die Größe des Kreises und damit das Gewicht kann durch Ziehen der Kreislinie mit der Maus verändert werden.

Bestehende Kontrollpunkte können mit gedrückter Maustaste verschoben werden. Außerdem können neue Punkte auf dem Kontrollpolygon eingefügt werden (um dieses zu verfeinern), indem mit der Maus auf die gewünschte Stelle auf dem Kontrollpolygon geklickt wird.

Durch Drücken der Umschalttaste kann in den Modus zum Transformieren der Kurve gewechselt werden. In diesem Modus kann die Kontur rotiert, verschoben und skaliert werden, jedoch nur, wenn der entsprechende Parameter (`Rotate`, `Move`, `Scale` auf *'true'* gesetzt wurde. Die Kontur wird zusammen mit einem Kreuz in der Mitte, einem Pfeil nach rechts, falls `Rotate` auf *'true'* gesetzt wurde, und einem Doppelpfeil, falls `Scale` auf *'true'* gesetzt wurde, dargestellt.

Die Kurve kann

- verschoben werden, indem man mit der linke Maustaste auf das Kreuz im Mittelpunkt klickt und dann die Kurve an die neue Position schiebt.
- rotiert werden, indem man mit der linken Maustaste auf den Pfeil klickt und die Maus bewegt bis die Kurve die gewünschte Lage eingenommen hat.
- skaliert werden, indem der Doppelpfeil verschoben wird. Um das Seitenverhältnis beim Skalieren beizubehalten, muss der Parameter `KeepRatio` auf *'true'* gesetzt werden.

Durch ein weiteres Drücken der Umschalttaste wechselt man zurück in den Editiermodus. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_nurbs` auch durch Drücken der Escape-Taste beendet werden.

Das Aussehen der Kurve beim Zeichnen wird durch die Linienbreite, den Linienstil und die Farbe bestimmt, die mit den Operatoren `set_color`, `set_colored`, `set_line_width` und `set_line_style` gesetzt werden können. Das Kontrollpolygon und alle Anfasser werden in der zweiten Farbe gezeichnet, die via `set_color` oder `set_colored` gesetzt wurde. Ihre Linienbreite ist fest auf 1 gesetzt, außerdem werden ihre Linien durchgezogen gezeichnet.

Parameter

▷ **`ContOut`** (output_object) ..... xld_cont ⇝ *object*
Kontur der NURBS-Kurve.

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Rotate`** (input_control) ..... string ⇝ *string*
Rotation ermöglichen?
Defaultwert: 'true'
Werteliste: `Rotate` ∈ {'true', 'false'}

▷ **`Move`** (input_control) ..... string ⇝ *string*
Verschieben ermöglichen?
Defaultwert: 'true'
Werteliste: `Move` ∈ {'true', 'false'}

▷ **`Scale`** (input_control) ..... string ⇝ *string*
Skalieren ermöglichen?
Defaultwert: 'true'
Werteliste: `Scale` ∈ {'true', 'false'}

▷ **`KeepRatio`** (input_control) ..... string ⇝ *string*
Seitenverhältnis beim Skalieren beibehalten?
Defaultwert: 'true'
Werteliste: `KeepRatio` ∈ {'true', 'false'}

▷ **`Degree`** (input_control) ..... integer ⇝ *integer*
Grad p der Kurve. Vernünftige Werte liegen zwischen 3 und 25.
Defaultwert: 3
Wertevorschläge: `Degree` ∈ {2, 3, 4, 5}
Restriktion: Degree >= 2

▷ **`Rows`** (output_control) ..... coordinates.y-array ⇝ *real*
Zeilenkoordinaten des Kontrollpolygons.

▷ **Cols** (output_control) ..... coordinates.x-array ⇝ *real*
Spaltenkoordinaten des Kontrollpolygons.

▷ **Weights** (output_control) ..... real-array ⇝ *real*
Gewichtsvektor.

Ergebnis

draw_nurbs liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_xld, draw_nurbs_interp

Siehe auch

draw_nurbs_mod, draw_nurbs_interp, gen_contour_nurbs_xld

Modul

Foundation

draw_nurbs_interp (: ContOut : WindowHandle, Rotate, Move, Scale, KeepRatio, Degree : ControlRows, ControlCols, Knots, Rows, Cols, Tangents)

Interaktives Erstellen einer NURBS-Kurve durch Interpolation.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit dev_open_window geöffnet wurden.

draw_nurbs_interp liefert die Kontur ContOut einer NURBS-Kurve, die interaktiv durch den Benutzer im Fenster WindowHandle durch Interpolation erzeugt worden ist. Mit dem Operator draw_nurbs ist es möglich eine NURBS-Kurve durch Zeichnen des Kontrollpolygons zu erzeugen.

Zusätzlich zu ContOut werden die Kontrollinformation der Kurve (ControlRows,ControlCols und Knots), die vom Benutzer bestimmten Interpolationspunkte (Rows und Cols) und die Tangenten im ersten und letzten Punkt der Kurve (Tangents) zurückgegeben. Tangents besteht aus den Y- (Zeilen) und X- (Spalten) Werten der Tangente im Startpunkt, gefolgt von den beiden Werten der Tangente im Endpunkt der Kurve.

Ein Gewichtsvektor wird nicht zurückgegeben, da dieser aus gleichen Einträgen besteht. Das heißt, dass *'auto'* als Gewichtsvektor verwendet werden kann, wenn die Kontrollinformationen im Operator gen_contour_nurbs_xld verwendet werden. Mehr Informationen über NURBS-Kurven befinden sich in der Dokumentation des Operators gen_contour_nurbs_xld.

Direkt nach dem Aufruf von draw_nurbs_interp kann man durch Drücken der linken Maustaste an den gewünschten Positionen Interpolationspunkte hinzufügen. Sind genug Punkte vorhanden (mindestens Degree − 1), wird eine NURBS-Kurve, die durch alle Punkte geht berechnet und angezeigt.

Der zuletzt angehängte Punkt kann durch Drücken der Steuerungstaste (Strg, Ctrl) gelöscht werden. Sobald 3 Punkte vorhanden sind, werden der erste und der letzte Punkt durch ein zusätzliches Quadrat markiert. Durch Klicken darauf wird die Kontur geschlossen bzw. wieder geöffnet.

Außerdem werden die Tangenten (d.h. die erste Ableitung der Kurve) im ersten und im letzten Punkt der Kurve angezeigt. Diese Tangenten können durch Verschieben ihrer Endpunkte mit der Maus verändert werden.

Vorhandene Punkte können durch Ziehen mit der Maus bewegt werden. Außerdem können neue Punkte auf der Kurve eingefügt werden, indem man auf die gewünschte Position auf der Kurve mit der linken Maustaste klickt.

Durch Drücken der Umschalttaste kann in den Modus zum Transformieren der Kurve gewechselt werden. In diesem Modus kann die Kurve rotiert, verschoben und skaliert werden, jedoch nur wenn der entsprechende Parameter (`Rotate`, `Move`, `Scale`) auf *'true'* gesetzt wurde. Die Kurve wird zusammen mit einem Kreuz in der Mitte und einem Pfeil nach rechts, falls `Rotate` auf *'true'* gesetzt wurde, und einem Doppelpfeil, falls `Scale` auf *'true'* gesetzt wurde, dargestellt.

Die Kurve kann

- verschoben werden, indem man mit der linken Maustaste auf das Kreuz im Mittelpunkt klickt und dann die Kurve an die neue Position schiebt,
- rotiert werden, indem man mit der linken Maustaste auf den Pfeil klickt und die Maus bewegt bis die Kurve die gewünschte Lage eingenommen hat, und
- skaliert werden, indem der Doppelpfeil verschoben wird. Um das Seitenverhältnis beim Skalieren beizubehalten, muss der Parameter `KeepRatio` auf *'true'* gesetzt werden.

Durch ein weiteres Drücken der Umschalttaste wechselt man zurück in den Editiermodus. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_nurbs_mod` auch durch Drücken der Escape-Taste beendet werden.

Das Aussehen der Kurve beim Zeichnen wird durch die Linienbreite, den Linienstil und die Farbe bestimmt, die mit den Operatoren `set_color`, `set_colored`, `set_line_width` und `set_line_style` gesetzt werden können. Die Tangenten und alle Anfasser werden in der zweiten Farbe gezeichnet, die via `set_color` oder `set_colored` gesetzt wurde. Ihre Linienbreite ist fest auf 1 gesetzt, außerdem werden ihre Linien durchgezogen gezeichnet.

Achtung

Im Gegensatz zu `draw_nurbs` hat jeder Punkt, der vom Nutzer vorgegeben wird, Einfluss auf die ganze Kurve. Wird ein Punkt bewegt, kann und wird sich die ganze Kurve ändern. Um diesen Effekt zu minimieren, empfehlen wir die Verwendung eines niedrigen Grades (3-5). Im Allgemeinen sind die Ergebnisse bei ungeradem Grad etwas besser als bei geradem. Außerdem ist es vorteilhaft, die Abstände aufeinanderfolgender Punkte ungefähr gleich zu wählen.

Parameter

▷ **`ContOut`** (output_object) . . . . . xld_cont ⇝ *object*
Kontur der Kurve.

▷ **`WindowHandle`** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **`Rotate`** (input_control) . . . . . string ⇝ *string*
Rotation ermöglichen?
Defaultwert: 'true'
Werteliste: `Rotate` ∈ {'true', 'false'}

▷ **`Move`** (input_control) . . . . . string ⇝ *string*
Verschieben ermöglichen?
Defaultwert: 'true'
Werteliste: `Move` ∈ {'true', 'false'}

▷ **`Scale`** (input_control) . . . . . string ⇝ *string*
Skalieren ermöglichen?
Defaultwert: 'true'
Werteliste: `Scale` ∈ {'true', 'false'}

▷ **`KeepRatio`** (input_control) . . . . . string ⇝ *string*
Seitenverhältnis beim Skalieren beibehalten?
Defaultwert: 'true'
Werteliste: `KeepRatio` ∈ {'true', 'false'}

▷ **`Degree`** (input_control) . . . . . integer ⇝ *integer*
Grad p der Kurve. Vernünftige Werte liegen zwischen 3 und 5.
Defaultwert: 3
Wertevorschläge: `Degree` ∈ {2, 3, 4, 5}
Restriktion: Degree >= 2 && Degree <= 9

▷ **`ControlRows`** (output_control) . . . . . coordinates.y-array ⇝ *real*
Zeilenkoordinaten des Kontrollpolygons.

▷ **ControlCols** (output_control) ..... coordinates.x-array ⇝ *real*
Spaltenkoordinaten des Kontrollpolygons

▷ **Knots** (output_control) ..... real-array ⇝ *real*
Knotenvektor.

▷ **Rows** (output_control) ..... coordinates.y-array ⇝ *real*
Zeilenkoordinaten der vom Benutzer bestimmten Punkte.

▷ **Cols** (output_control) ..... coordinates.x-array ⇝ *real*
Spaltenkoordinaten der vom Benutzer bestimmten Punkte.

▷ **Tangents** (output_control) ..... real-array ⇝ *real*
Vom Benutzer bestimmte Tangenten.

Ergebnis

`draw_nurbs_interp` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_xld`, `draw_nurbs`

Siehe auch

`draw_nurbs_interp_mod`, `draw_nurbs`, `gen_contour_nurbs_xld`

Modul

Foundation

```
draw_nurbs_interp_mod ( : ContOut : WindowHandle, Rotate, Move,
    Scale, KeepRatio, Edit, Degree, RowsIn, ColsIn,
    TangentsIn : ControlRows, ControlCols, Knots, Rows, Cols,
    Tangents )
```

Interaktives Modifizieren einer NURBS-Kurve durch Interpolation.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit `dev_open_window` geöffnet wurden.

`draw_nurbs_interp_mod` liefert die Kontur `ContOut` einer NURBS-Kurve, die interaktiv durch den Benutzer im Fenster `WindowHandle` durch Interpolation modifiziert worden ist.

Zusätzlich zu `ContOut` werden die Kontrollinformation der Kurve (`ControlRows`,`ControlCols` und `Knots`), die vom Benutzer bestimmten Interpolationspunkte (`Rows` und `Cols`) und die Tangenten im ersten und letzten Punkt der Kurve (`Tangents`) zurückgegeben. `Tangents` besteht aus den Y- (Zeilen) und X- (Spalten) Werten der Tangente im Startpunkt, gefolgt von den beiden Werten der Tangente im Endpunkt der Kurve.

Ein Gewichtsvektor wird nicht zurückgegeben, da dieser aus gleichen Einträgen besteht. Das heißt, dass *'auto'* als Gewichtsvektor verwendet werden kann, wenn die Kontrollinformationen im Operator `gen_contour_nurbs_xld` verwendet werden. Mehr Informationen über NURBS-Kurven befinden sich in der Dokumentation des Operators `gen_contour_nurbs_xld`.

Die Eingabekurve wird durch die Interpolationspunkte `RowsIn`, `ColsIn`, den Grad `Degree` und die Tangenten `TangentsIn` beschrieben, so dass `draw_nurbs_interp_mod` mit den Ausgabedaten des Operators `draw_nurbs_interp` verwendet werden kann.

Die Kurve kann auf zwei Arten modifiziert werden: Durch Editieren, zum Beispiel durch Einfügen oder Verschieben von Punkten, oder durch das Transformieren der Kurve als Ganzes, zum Beispiel durch Rotieren, Verschieben

oder Skalieren. Die Kontur kann jedoch nur editiert werden, wenn der Parameter `Edit` auf *'true'* gesetzt wurde. Die Parameter `Rotate`, `Move` und `Scale` müssen auf *'true'* gesetzt werden, damit die entsprechende Bearbeitung durchgeführt werden kann. `draw_nurbs_interp_mod` beginnt im Modus zum Transformieren. Durch Drücken der Umschalttaste kann in den Modus zum Editieren gewechselt werden; durch erneutes Drücken der Taste wechselt man zurück in den Modus zum Transformieren.

Transformieren

In diesem Modus kann die Kurve rotiert, verschoben und skaliert werden, jedoch nur, wenn der entsprechenden Parameter (`Rotate`, `Move`, `Scale`) auf *'true'* gesetzt wurde. Die Kurve wird zusammen mit einem Kreuz in der Mitte, einem Pfeil nach rechts, falls `Rotate` auf *'true'* gesetzt wurde, und einem Doppelpfeil, falls `Scale` auf *'true'* gesetzt wurde, dargestellt.

- Zum Verschieben der Kurve klickt man mit der linken Maustaste auf das Kreuz in der Mitte und bewegt die Maus dann mit gedrückter Taste zur neuen Position.
- Zum Rotieren der Kurve klickt man mit der linke Maustaste auf den Pfeil und bewegt die Maus anschließend mit gedrückter Taste solange, bis sich die Kurve in der gewünschten Lage befindet.
- Zum Skalieren der Kurve muss der Doppelpfeil bewegt werden. Um das Seitenverhältnis beim Skalieren beizubehalten, muss der Parameter `KeepRatio` auf *'true'* gesetzt werden.

Editieren

In diesem Modus wird die Kurve zusammen mit den Interpolationspunkten, der Starttangente und der Endtangente dargestellt. Start- und Endpunkt sind mit einem zusätzlichen Quadrat markiert. Folgende Modifikationen können vorgenommen werden.

- Um Punkte anzuhängen, muss mit der linken Maustaste ins Fenster geklickt werden. Hierdurch wird ein neuer Punkt an dieser Position eingefügt.
- Der zuletzt angehängte Punkt kann durch Drücken der Steuerungstaste (Strg, Ctrl) gelöscht werden.
- Punkte können durch Ziehen mit der Maus verschoben werden.
- Neue Punkte auf der Kurve werden eingefügt, indem auf die gewünschte Stelle auf der Kurve geklickt wird.
- Um die Kurve zu schließen bzw. zu öffnen, muss auf den ersten oder letzten Punkt geklickt werden.
- Die Tangenten können verändert werden, indem ihr Ende mit der Maus verschoben wird.

Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_nurbs_interp_mod` auch durch Drücken der Escape-Taste beendet werden.

Das Aussehen der Kurve beim Zeichnen wird durch die Linienbreite, den Linienstil und die Farbe bestimmt, die mit den Operatoren `set_color`, `set_colored`, `set_line_width` und `set_line_style` gesetzt werden können. Die Tangenten und alle Anfasser werden in der zweiten Farbe gezeichnet, die via `set_color` oder `set_colored` gesetzt wurde. Ihre Linienbreite ist fest auf 1 gesetzt, außerdem werden ihre Linien durchgezogen gezeichnet.

Achtung

Im Gegensatz zu `draw_nurbs` hat jeder Punkt, der vom Nutzer vorgegeben wird, Einfluss auf die ganze Kurve. Wird ein Punkt bewegt, kann und wird sich die ganze Kurve ändern. Um diesen Effekt zu minimieren, empfehlen wir die Verwendung eines niedrigen Grades (3-5). Im Allgemeinen sind die Ergebnisse bei ungeradem Grad etwas besser als bei geradem. Außerdem ist es vorteilhaft, die Abstände aufeinanderfolgender Punkte ungefähr gleich zu wählen.

Parameter

▷ **`ContOut`** (output_object) ..... xld_cont ⇝ *object*
Kontur der modifizierten Kurve.

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Rotate`** (input_control) ..... string ⇝ *string*
Rotation ermöglichen?
Defaultwert: 'true'
Werteliste: `Rotate` ∈ {'true', 'false'}

▷ **Move** (input_control) ..... string ⇝ *string*
Verschieben ermöglichen?
Defaultwert: 'true'
Werteliste: `Move` ∈ {'true', 'false'}

▷ **Scale** (input_control) ..... string ⇝ *string*
Skalieren ermöglichen?
Defaultwert: 'true'
Werteliste: `Scale` ∈ {'true', 'false'}

▷ **KeepRatio** (input_control) ..... string ⇝ *string*
Seitenverhältnis beim Skalieren beibehalten?
Defaultwert: 'true'
Werteliste: `KeepRatio` ∈ {'true', 'false'}

▷ **Edit** (input_control) ..... string ⇝ *string*
Editieren ermöglichen?
Defaultwert: 'true'
Werteliste: `Edit` ∈ {'true', 'false'}

▷ **Degree** (input_control) ..... integer ⇝ *integer*
Grad p der Kurve. Vernünftige Werte liegen zwischen 3 und 5.
Defaultwert: 3
Wertevorschläge: `Degree` ∈ {2, 3, 4, 5}
Restriktion: Degree >= 2 && Degree <= 9

▷ **RowsIn** (input_control) ..... coordinates.y-array ⇝ *real*
Zeilenkoordinaten der Eingabeinterpolationspunkte.

▷ **ColsIn** (input_control) ..... coordinates.x-array ⇝ *real*
Spaltenkoordinaten der Eingabeinterpolationspunkte.

▷ **TangentsIn** (input_control) ..... real-array ⇝ *real*
Eingabetangenten.

▷ **ControlRows** (output_control) ..... coordinates.y-array ⇝ *real*
Zeilenkoordinaten des Kontrollpolygons.

▷ **ControlCols** (output_control) ..... coordinates.x-array ⇝ *real*
Spaltenkoordinaten des Kontrollpolygons

▷ **Knots** (output_control) ..... real-array ⇝ *real*
Knotenvektor.

▷ **Rows** (output_control) ..... coordinates.y-array ⇝ *real*
Zeilenkoordinaten der vom Benutzer bestimmten Punkte.

▷ **Cols** (output_control) ..... coordinates.x-array ⇝ *real*
Spaltenkoordinaten der vom Benutzer bestimmten Punkte.

▷ **Tangents** (output_control) ..... real-array ⇝ *real*
Vom Benutzer bestimmte Tangenten.

Ergebnis

`draw_nurbs_interp_mod` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_xld_mod`, `draw_nurbs_mod`

Siehe auch

draw_nurbs_interp, gen_contour_nurbs_xld

Modul

Foundation

draw_nurbs_mod (: ContOut : WindowHandle, Rotate, Move, Scale, KeepRatio, Edit, Degree, RowsIn, ColsIn, WeightsIn : Rows, Cols, Weights)

Interaktives Modifizieren einer NURBS-Kurve.

Dieser Operator kann nicht in HDevelop-Grafikfenstern verwendet werden, die mit dev_open_window geöffnet wurden.

draw_nurbs_mod liefert die Kontur ContOut und die Kontrollinformationen (Rows, Cols, and Weights) der NURBS-Kurve mit dem Grad Degree, die interaktiv durch den Benutzer im Fenster WindowHandle modifiziert worden ist. Zusätzliche Informationen über NURBS-Kurven finden sich in der Dokumentation des Operators gen_contour_nurbs_xld. Um die Kontrollinformationen Rows, Cols und Weights in einem Aufruf des Operators gen_contour_nurbs_xld zu verwenden, sollte dort der Knotenvektor Knots auf *'auto'* gesetzt werden.

Die Eingabekurve wird durch ihr Kontrollpolygon RowsIn, ColsIn, ihren Gewichtsvektor WeightsIn und ihren Grad Degree beschrieben. Es wird angenommen, dass ihr Knotenvektor uniform ist (das entspricht *'auto'* in gen_contour_nurbs_xld).

Die Kurve kann auf zwei Arten modifiziert werden: Durch Editieren, zum Beispiel durch Einfügen oder Verschieben von Kontrollpunkten, oder durch das Transformieren der Kurve als Ganzes, zum Beispiel durch Rotieren, Verschieben oder Skalieren. Die Kurve kann jedoch nur editiert werden, wenn der Parameter Edit auf *'true'* gesetzt wurde. Die Parameter Rotate, Move und Scale müssen auf *'true'* gesetzt werden, damit die entsprechende Bearbeitung durchgeführt werden kann. draw_nurbs_mod beginnt im Modus zum Transformieren. Durch Drücken der Umschalttaste kann in den Modus zum Editieren gewechselt werden; durch erneutes Drücken der Taste wechselt man zurück in den Modus zum Transformieren.

Transformieren

In diesem Modus kann die Kurve rotiert, verschoben und skaliert werden, jedoch nur, wenn der entsprechenden Parameter Rotate, Move und Scale auf *'true'* gesetzt wurde. Die Kurve wird zusammen mit einem Kreuz in der Mitte, einem Pfeil nach rechts, falls Rotate auf *'true'* gesetzt wurde, und einem Doppelpfeil, falls Scale auf *'true'* gesetzt wurde, dargestellt.

- Zum Verschieben der Kurve klickt man mit der linken Maustaste auf das Kreuz in der Mitte und bewegt die Maus dann mit gedrückter Taste zur neuen Position.
- Zum Rotieren der Kurve klickt man mit der linken Maustaste auf den Pfeil und bewegt die Maus anschließend mit gedrückter Taste solange, bis sich die Kurve in der gewünschten Lage befindet.
- Zum Skalieren der Kurve muss der Doppelpfeil bewegt werden. Um das Seitenverhältnis beim Skalieren beizubehalten, muss der Parameter KeepRatio auf *'true'* gesetzt werden.

Editieren

In diesem Modus wird die Kontur zusammen mit ihrem Kontrollpolygon dargestellt. Start- und Endpunkt sind mit einem zusätzlichen Quadrat markiert. Außerdem ist der Punkt, der als letztes angefasst wurde, von einem Kreis umgeben, der das Gewicht dieses Punktes visualisiert. Folgende Modifikationen können vorgenommen werden:

- Um Punkte an das Kontrollpolygon anzuhängen, muss mit der linken Maustaste ins Fenster geklickt werden. Hierdurch wird ein neuer Punkt an dieser Position eingefügt.
- Der zuletzt angehängte Punkt kann durch Drücken der Steuerungstaste (Strg, Ctrl) gelöscht werden.
- Kontrollpunkte können durch Ziehen mit der Maus verschoben werden
- Punkte auf dem Kontrollpolygon werden eingefügt, indem auf die gewünschte Stelle auf dem Kontrollpolygon geklickt wird.
- Um die Kurve zu schließen bzw. zu öffnen muss auf den ersten oder letzten Punkt des Kontrollpolygons geklickt werden.

- Das Gewicht eines Kontrollpunktes kann verändert werden, indem zuerst auf den Punkt geklickt wird (wenn der Gewichtskreis noch nicht angezeigt wird) und anschließend die Größe des Kreises um den Punkt durch Ziehen mit der Maus verändert wird.

Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_nurbs_mod` auch durch Drücken der Escape-Taste beendet werden.

Das Aussehen der Kurve beim Zeichnen wird durch die Linienbreite, den Linienstil und die Farbe bestimmt, die mit den Operatoren `set_color`, `set_colored`, `set_line_width` und `set_line_style` gesetzt werden können. Das Kontrollpolygon und alle Anfasser werden in der zweiten Farbe gezeichnet, die via `set_color` oder `set_colored` gesetzt wurde. Ihre Linienbreite ist fest auf 1 gesetzt, außerdem werden ihre Linien durchgezogen gezeichnet.

Parameter

▷ **ContOut** (output_object) ..... xld_cont ⇝ *object*
Kontur der modifizierten Kurve.

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Rotate** (input_control) ..... string ⇝ *string*
Rotation ermöglichen?
Defaultwert: 'true'
Werteliste: `Rotate` ∈ {'true', 'false'}

▷ **Move** (input_control) ..... string ⇝ *string*
Verschieben ermöglichen?
Defaultwert: 'true'
Werteliste: `Move` ∈ {'true', 'false'}

▷ **Scale** (input_control) ..... string ⇝ *string*
Skalieren ermöglichen?
Defaultwert: 'true'
Werteliste: `Scale` ∈ {'true', 'false'}

▷ **KeepRatio** (input_control) ..... string ⇝ *string*
Seitenverhältnis beim Skalieren beibehalten?
Defaultwert: 'true'
Werteliste: `KeepRatio` ∈ {'true', 'false'}

▷ **Edit** (input_control) ..... string ⇝ *string*
Editieren ermöglichen?
Defaultwert: 'true'
Werteliste: `Edit` ∈ {'true', 'false'}

▷ **Degree** (input_control) ..... integer ⇝ *integer*
Grad p der Kurve. Vernünftige Werte liegen zwischen 3 und 25.
Defaultwert: 3
Wertevorschläge: `Degree` ∈ {2, 3, 4, 5}
Restriktion: Degree >= 2

▷ **RowsIn** (input_control) ..... coordinates.y-array ⇝ *real*
Zeilenkoordinaten des Eingabekontrollpolygons.

▷ **ColsIn** (input_control) ..... coordinates.x-array ⇝ *real*
Spaltenkoordinaten des Eingabekontrollpolygons.

▷ **WeightsIn** (input_control) ..... real-array ⇝ *real*
Eingabegewichtsvektor.

▷ **Rows** (output_control) ..... coordinates.y-array ⇝ *real*
Zeilen Koordinaten des Kontrollpolygons.

▷ **Cols** (output_control) ..... coordinates.x-array ⇝ *real*
Spalten Koordinaten des Kontrollpolygons.

▷ **Weights** (output_control) ..... real-array ⇝ *real*
Gewichtsvektor.

Ergebnis

`draw_nurbs_mod` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_nurbs_interp_mod`, `draw_xld_mod`

Siehe auch

`draw_nurbs`, `draw_nurbs_interp`, `gen_contour_nurbs_xld`

Modul

Foundation

draw_point (: : WindowHandle : Row, Column)

Zeichnen eines Punktes.

`draw_point` liefert die Parameter für einen Punkt, der interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Punktes wird mit der linken Maustaste geklickt. Während die Taste gedrückt bleibt, kann der Punkt in jede beliebige Richtung „verschoben“ werden. Nach einem erneuten Mausklick kann dieses beliebig verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_point` auch durch Drücken der Escape-Taste beendet werden.

Der Punkt ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenindex des Punktes.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenindex des Punktes.

Beispiel

```
draw_point(WindowHandle,Row,Column)
gen_cross_contour_xld (Cross, Row, Column, 6, 0)
```

Ergebnis

`draw_point` liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_line, set_colored, set_line_width, set_draw, set_insert

Siehe auch

draw_point_mod, draw_circle, draw_ellipse, set_insert

Modul

Foundation

draw_point_mod (: : WindowHandle, RowIn, ColumnIn : Row, Column)

Zeichnen eines Punktes.

draw_point_mod liefert die Parameter für einen Punkt, der interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Punktes werden die Koordinaten RowIn und ColumnIn erwartet. Nach einem Mausklick kann dieses beliebig verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann draw_point_mod auch durch Drücken der Escape-Taste beendet werden.

Der Punkt ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **RowIn** (input_control) ..... point.y ⇝ *real*
  Zeilenindex des Punktes.
- ▷ **ColumnIn** (input_control) ..... point.x ⇝ *real*
  Spaltenindex des Punktes.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenindex des Punktes.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenindex des Punktes.

Beispiel

```
draw_point_mod (WindowHandle, 100, 100, Row,Column)
gen_cross_contour_xld (Cross, Row, Column, 6, 0)
```

Ergebnis

draw_point_mod liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_line, set_colored, set_line_width, set_draw, set_insert

Siehe auch

draw_point, draw_circle, draw_ellipse, set_insert

Modul

Foundation

draw_polygon (: PolygonRegion : WindowHandle :)

Interaktives Erstellen eines Polygonzuges.

draw_polygon liefert ein Bild, dessen Region genau die interaktiv mit der Maus eingegebene Bildpunkte umfasst (die Grauwerte bleiben undefiniert).

Gezeichnet wird im Ausgabefenster mit gedrückter linker Maustaste. Wird die linke Maustaste losgelassen und an einer anderen Stelle wieder gedrückt, dann wird zwischen diesen beiden Punkten ein Geradenstück gezogen. Ein Druck auf die rechte Maustaste beendet die Eingabe. Unter macOS kann draw_polygon auch durch Drücken der Escape-Taste beendet werden. Für das Zeichnen wird die Farbe verwendet die mit set_color, set_rgb, etc. eingestellt wurde.

Um das erzeugte PolygonRegion für die weitere Verarbeitung mit Grauwerten zu unterlegen, kann der Operator reduce_domain verwendet werden.

Achtung

Die gezeichnete Kontur wird nicht automatisch geschlossen, insbesondere auch nicht „aufgefüllt“.

Die Grauwerte des Ausgabeobjektes sind undefiniert.

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **PolygonRegion** (output_object) ..... region ⇝ *object*
  Region, die alle gezeichneten Punkte umfasst.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.

Beispiel

```
draw_polygon(Polygon,WindowHandle)
shape_trans(Polygon,Filled,'convex')
dev_display (Filled)
```

Ergebnis

draw_polygon liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_region, set_colored, set_line_width, set_draw

Alternativen

draw_region, draw_circle, draw_rectangle1, draw_rectangle2, boundary

Siehe auch

reduce_domain, fill_up, set_color

Modul

Foundation

draw_rectangle1 (: : WindowHandle : Row1, Column1, Row2, Column2)

Zeichnen eines Rechtecks parallel zu den Koordinatenachsen.

draw_rectangle1 liefert die Parameter für ein Rechteck parallel zu den Koordinatenachsen, das interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Rechtecks wird mit der linken Maustaste für eine Ecke des Rechtecks geklickt. Während die Taste gedrückt bleibt, kann das Rechteck in jede beliebige Richtung „aufgezogen“ werden. Nach einem erneuten Mausklick in die Mitte des erzeugten Rechtecks, kann dieses beliebig verschoben werden. Mit einem Klick in die Nähe einer Seite kann diese „angefasst“ werden, um die Ausdehnung des Rechtecks in senkrechter Richtung zu dieser Seite zu verändern. Klickt man auf eine Ecke des erzeugten Rechtecks, kann diese Ecke verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann draw_rectangle1 auch durch Drücken der Escape-Taste beendet werden.

Das Rechteck ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row1** (output_control) ..... rectangle.origin.y ⇝ *real*
  Zeilenindex der linken oberen Ecke.
- ▷ **Column1** (output_control) ..... rectangle.origin.x ⇝ *real*
  Spaltenindex der linken oberen Ecke.
- ▷ **Row2** (output_control) ..... rectangle.corner.y ⇝ *real*
  Zeilenindex der rechten unteren Ecke.
- ▷ **Column2** (output_control) ..... rectangle.corner.x ⇝ *real*
  Spaltenindex der rechten unteren Ecke.

Beispiel

```
read_image(Image,'monkey')
get_image_size (Image, Width, Height)
dev_display (Image)
draw_rectangle1(WindowHandle,Row1,Column1,Row2,Column2)
dev_set_part (Row1, Column1, Row2, Column2)
dev_display (Image)
```

Ergebnis

draw_rectangle1 liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_rectangle1_mod, draw_rectangle2, draw_region

Siehe auch

gen_rectangle1, draw_circle, draw_ellipse, set_insert

Modul

Foundation

draw_rectangle1_mod (: : WindowHandle, Row1In, Column1In, Row2In, Column2In : Row1, Column1, Row2, Column2)

Zeichnen eines Rechtecks parallel zu den Koordinatenachsen.

draw_rectangle1_mod liefert die Parameter für ein Rechteck parallel zu den Koordinatenachsen, das interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Rechtecks werden die Parametern Row1In, Column1In,Row2In und Column2In erwartet. Nach einem Mausklick in die Mitte des erzeugten Rechtecks, kann dieses beliebig verschoben werden. Mit einem Klick in die Nähe einer Seite kann diese „angefasst“ werden, um die Ausdehnung des Rechtecks in senkrechter Richtung zu dieser Seite zu verändern. Klickt man auf eine Ecke des erzeugten Rechtecks, kann diese Ecke verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann draw_rectangle1_mod auch durch Drücken der Escape-Taste beendet werden.

Das Rechteck ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row1In** (input_control) ..... rectangle.origin.y ⇝ *real*
  Zeilenindex der linken oberen Ecke.
- ▷ **Column1In** (input_control) ..... rectangle.origin.x ⇝ *real*
  Spaltenindex der linken oberen Ecke.
- ▷ **Row2In** (input_control) ..... rectangle.corner.y ⇝ *real*
  Zeilenindex der rechten unteren Ecke.
- ▷ **Column2In** (input_control) ..... rectangle.corner.x ⇝ *real*
  Spaltenindex der rechten unteren Ecke.
- ▷ **Row1** (output_control) ..... rectangle.origin.y ⇝ *real*
  Zeilenindex der linken oberen Ecke.
- ▷ **Column1** (output_control) ..... rectangle.origin.x ⇝ *real*
  Spaltenindex der linken oberen Ecke.
- ▷ **Row2** (output_control) ..... rectangle.corner.y ⇝ *real*
  Zeilenindex der rechten unteren Ecke.
- ▷ **Column2** (output_control) ..... rectangle.corner.x ⇝ *real*
  Spaltenindex der rechten unteren Ecke.

Beispiel

```
read_image(Image,'monkey')
get_image_size (Image, Width, Height)
dev_display (Image)
draw_rectangle1_mod(WindowHandle,100,100,200,300,\
                    Row1,Column1,Row2,Column2)
dev_set_part (Row1, Column1, Row2, Column2)
dev_display (Image)
```

Ergebnis

draw_rectangle1_mod liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_rectangle1`, `draw_rectangle2`, `draw_region`

Siehe auch

`gen_rectangle1`, `draw_circle`, `draw_ellipse`, `set_insert`

Modul

Foundation

draw_rectangle2 (: : WindowHandle : Row, Column, Phi, Length1, Length2)

Interaktives Erstellen eines beliebig orientierten Rechtecks.

`draw_rectangle2` liefert die Parameter für ein beliebig orientiertes Rechteck, das interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Das zu erzeugende Rechteck wird durch seinen Mittelpunkt, seine zwei Halbachsen und den Winkel der ersten Halbachse zur horizontalen Koordinatenachse beschrieben.

Für die Erzeugung des Rechtecks wird mit der linken Maustaste der Mittelpunkt des Rechtecks festgelegt. Während die Taste gedrückt bleibt, kann die Länge (`Length1`) und die Orientierung (`Phi`) der ersten Halbachse festgelegt werden. Dabei wird für die zweite Halbachse vorläufig eine Default-Länge verwendet, die anschließend aber noch verändert werden kann. Nach einem erneuten Mausklick in die Mitte des erzeugten Rechtecks, kann dieses beliebig verschoben werden. Mit einem Klick in die Nähe einer Seite kann diese „angefasst“ werden, um die Ausdehnung des Rechtecks in senkrechter Richtung zu dieser Seite zu verändern. Die Orientierung kann nur verändert werden, wenn eine Seite senkrecht zur ersten Halbachse angefasst wird. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_rectangle2` auch durch Drücken der Escape-Taste beendet werden.

Das Rechteck ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Row** (output_control) ..... rectangle2.center.y ⇝ *real*
  Zeilenindex des Mittelpunkts.
- ▷ **Column** (output_control) ..... rectangle2.center.x ⇝ *real*
  Spaltenindex des Mittelpunkts.
- ▷ **Phi** (output_control) ..... rectangle2.angle.rad ⇝ *real*
  Orientierung der größeren Halbachse in Bogenmaß.
- ▷ **Length1** (output_control) ..... rectangle2.hwidth ⇝ *real*
  Größere Halbachse.
- ▷ **Length2** (output_control) ..... rectangle2.hheight ⇝ *real*
  Kleinere Halbachse.

Ergebnis

`draw_rectangle2` liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_rectangle2_mod`, `draw_rectangle1`, `draw_region`

Siehe auch

`gen_rectangle2`, `draw_circle`, `draw_ellipse`, `set_insert`

Modul

Foundation

draw_rectangle2_mod (: : WindowHandle, RowIn, ColumnIn, PhiIn, Length1In, Length2In : Row, Column, Phi, Length1, Length2)

Interaktives Erstellen eines beliebig orientierten Rechtecks.

`draw_rectangle2_mod` liefert die Parameter für ein beliebig orientiertes Rechteck, das interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Das zu erzeugende Rechteck wird durch seinen Mittelpunkt, seine zwei Halbachsen und den Winkel der ersten Halbachse zur horizontalen Koordinatenachse beschrieben.

Für die Erzeugung des Rechtecks werden die Parametern `RowIn`, `ColumnIn`,`PhiIn`,`Length1In`,`Length2In` erwartet. Mit einem Klick in die Nähe einer Seite kann diese „angefasst“ werden, um die Ausdehnung des Rechtecks in senkrechter Richtung (`Length2`) zu dieser Seite zu verändern. Die Orientierung (`Phi`) kann nur verändert werden, wenn eine Seite senkrecht zur ersten Halbachse angefasst wird. Nach einem erneuten Mausklick in die Mitte des erzeugten Rechtecks, kann dieses beliebig verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_rectangle2_mod` auch durch Drücken der Escape-Taste beendet werden.

Das Rechteck ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **RowIn** (input_control) ..... rectangle2.center.y ⇝ *real*
  Zeilenindex des Mittelpunkts.
- ▷ **ColumnIn** (input_control) ..... rectangle2.center.x ⇝ *real*
  Spaltenindex des Mittelpunkts.
- ▷ **PhiIn** (input_control) ..... rectangle2.angle.rad ⇝ *real*
  Orientierung der größeren Halbachse in Bogenmaß.
- ▷ **Length1In** (input_control) ..... rectangle2.hwidth ⇝ *real*
  Größere Halbachse.
- ▷ **Length2In** (input_control) ..... rectangle2.hheight ⇝ *real*
  Kleinere Halbachse.
- ▷ **Row** (output_control) ..... rectangle2.center.y ⇝ *real*
  Zeilenindex des Mittelpunkts.
- ▷ **Column** (output_control) ..... rectangle2.center.x ⇝ *real*
  Spaltenindex des Mittelpunkts.
- ▷ **Phi** (output_control) ..... rectangle2.angle.rad ⇝ *real*
  Orientierung der größeren Halbachse in Bogenmaß.

▷ **Length1** (output_control) . . . . . rectangle2.hwidth ⇝ *real*
Größere Halbachse.

▷ **Length2** (output_control) . . . . . rectangle2.hheight ⇝ *real*
Kleinere Halbachse.

Ergebnis

draw_rectangle2_mod liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_rectangle2, draw_rectangle1, draw_rectangle2, draw_region

Siehe auch

gen_rectangle2, draw_circle, draw_ellipse, set_insert

Modul

Foundation

draw_region (: Region : WindowHandle :)

Interaktives Erstellen einer geschlossenen Region.

draw_region liefert ein Bild, dessen Region die interaktiv mit der Maus eingegebene Bildregion umfasst (die Grauwerte bleiben undefiniert). Gezeichnet wird im Ausgabefenster mit gedrückter linker Maustaste. Die linke Maustaste kann auch punktweise betätigt werden; hierdurch wird eine Gerade zwischen den angeklickten Punkten gezogen. Ein Druck auf die rechte Maustaste beendet die Eingabe und schließt die Kontur. Unter macOS kann draw_region auch durch Drücken der Escape-Taste beendet werden. Anschließend wird das Bild „aufgefüllt“, enthält also den gesamten mit der Maus umfahrenen Bildbereich.

Für das Zeichnen wird die Farbe verwendet die mit set_color, set_rgb, etc. eingestellt wurde.

Achtung

Die Grauwerte des Ausgabeobjektes sind undefiniert.

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

▷ **Region** (output_object) . . . . . region ⇝ *object*
Interaktiv erstellte Region.

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

Beispiel

```
read_image(Image,'fabrik')
dev_display (Image)
draw_region(Region,WindowHandle)
reduce_domain(Image,Region,ImageReduced)
invert_image (ImageReduced, ImageInvert)
dev_display (ImageInvert)
```

Ergebnis

`draw_region` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`

Alternativen

`draw_circle`, `draw_ellipse`, `draw_rectangle1`, `draw_rectangle2`

Siehe auch

`draw_polygon`, `reduce_domain`, `fill_up`, `set_color`

Modul

Foundation

draw_xld (: ContOut : WindowHandle, Rotate, Move, Scale, KeepRatio :)

Interaktives Erstellen einer Kontur.

`draw_xld` liefert eine Kontur, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Direkt nach dem Aufruf von `draw_xld` kann man, durch Drücken der linken Maustaste, Punkte zur Kontur hinzufügen. Der zuletzt angehängte Punkt kann durch Drücken der Steuerungstaste (Strg, Ctrl) gelöscht werden. Sobald die Kontur aus 3 Punkten besteht, werden 5 Pickpunkte dargestellt, einer in der Mitte und die anderen an den 4 Ecken des umschließenden Rechtecks. Durch Klicken auf einen dieser Pickpunkte kann die Kontur geschlossen und wieder geöffnet werden. Bei einer geschlossenen Kontur werden die Pickpunkte als Rechtecke dargestellt, ansonsten gleicht ihre Form einem 'u'.

Bei einer geschlossene Kontur kann man

- Konturpunkte verschieben, indem man mit der linken Maustaste auf einen Punkt, markiert durch ein Rechteck, klickt und bei gedrückter Maustaste die Maus bewegt,
- Konturpunkte einfügen, indem man mit der linken Maustaste in die Nähe einer Linie klickt, und dann die Maus bei gedrückter Maustaste an die gewünschte Position bewegt, und
- Konturpunkte löschen, indem man einen Punkt über die linke Maustaste selektiert und dann die Steuerungstaste (Strg, Ctrl) drückt.

Durch Drücken der Umschalttaste kann in den Modus zum Transformieren der Kontur gewechselt werden. In diesem Modus kann die Kontur rotiert, verschoben und skaliert werden, jedoch nur wenn der entsprechenden Parameter `Rotate`, `Move` und `Scale` auf *'true'* gesetzt wurde. Die Kontur wird zusammen mit einem Kreuz in der Mitte und einem Pfeil nach rechts, falls `Rotate` auf *'true'* gesetzt wurde, und einem Doppelpfeil, falls `Scale` auf *'true'* gesetzt wurde, dargestellt.

Die Kontur kann

- verschoben werden, indem man mit der linken Maustaste auf das Kreuz im Mittelpunkt klickt und dann die Kontur an die neue Position schiebt,
- rotiert werden, indem man mit der linken Maustaste auf den Pfeil klickt und die Maus bewegt bis die Kontur die gewünschte Lage eingenommen hat, und
- skaliert werden, indem der Doppelpfeil verschoben wird. Um das Seitenverhältnis beim Skalieren beizubehalten, muss der Parameter `KeepRatio` auf *'true'* gesetzt werden.

Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_xld` auch durch Drücken der Escape-Taste beendet werden.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

- ▷ **`ContOut`** (output_object) ..... xld_cont ⇝ *object*
  Modifizierte Kontur.
- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`Rotate`** (input_control) ..... string ⇝ *string*
  Rotation ermöglichen?
  Defaultwert: 'true'
  Werteliste: `Rotate` ∈ {'true', 'false'}
- ▷ **`Move`** (input_control) ..... string ⇝ *string*
  Verschieben ermöglichen?
  Defaultwert: 'true'
  Werteliste: `Move` ∈ {'true', 'false'}
- ▷ **`Scale`** (input_control) ..... string ⇝ *string*
  Skalieren ermöglichen?
  Defaultwert: 'true'
  Werteliste: `Scale` ∈ {'true', 'false'}
- ▷ **`KeepRatio`** (input_control) ..... string ⇝ *string*
  Seitenverhältnis beim Skalieren beibehalten?
  Defaultwert: 'true'
  Werteliste: `KeepRatio` ∈ {'true', 'false'}

Ergebnis

`draw_xld` liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Nachfolger

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_rectangle2`, `draw_rectangle1`, `draw_rectangle2`, `draw_region`

Siehe auch

`gen_rectangle2`, `draw_circle`, `draw_ellipse`, `set_insert`

Modul

Foundation

```
draw_xld_mod ( ContIn : ContOut : WindowHandle, Rotate, Move, Scale,
    KeepRatio, Edit : )
```

Interaktives Modifizieren einer Kontur.

`draw_xld_mod` liefert eine Kontur, die interaktiv durch den Benutzer im Fenster modifiziert worden ist.

Die Kontur kann auf zwei Arten modifiziert werden: Durch Editieren, zum Beispiel durch Einfügen oder Verschieben von Punkten, oder durch das Transformieren der Kontur als Ganzes, zum Beispiel durch Rotation, Verschieben

oder Skalieren. Die Kontur kann jedoch nur editiert werden, wenn der Parameter `Edit` auf *'true'* gesetzt wurde. Die Parameter `Rotate`, `Move` und `Scale` müssen auf *'true'* gesetzt werden, damit die entsprechende Bearbeitung durchgeführt werden kann. `draw_xld_mod` beginnt im Modus zum Transformieren. Durch Drücken der Umschalttaste kann in den Modus zum Editieren gewechselt werden; durch erneutes Drücken der Taste wechselt man zurück in den Modus zum Transformieren.

Transformieren In diesem Modus kann die Kontur rotiert, verschoben und skaliert werden, jedoch nur wenn der entsprechenden Parameter `Rotate`, `Move` und `Scale` auf *'true'* gesetzt wurde. Die Kontur wird zusammen mit einem Kreuz in der Mitte und einem Pfeil nach rechts, falls `Rotate` auf *'true'* gesetzt wurde, und einem Doppelpfeil, falls `Scale` auf *'true'* gesetzt wurde, dargestellt.

- Zum Verschieben der Kontur klickt man mit der linken Maustaste auf das Kreuz in der Mitte und bewegt die Maus dann mit gedrückter Taste zur neuen Position.
- Zum Rotieren der Kontur klickt man mit der linken Maustaste auf den Pfeil und bewegt die Maus anschließend mit gedrückter Taste solange, bis sich die Kontur in der gewünschten Lage befindet.
- Zum Skalieren der Kontur muss der Doppelpfeil bewegt werden. Um das Seitenverhältnis beim Skalieren beizubehalten, muss der Parameter `KeepRatio` auf *'true'* gesetzt werden.

Editieren In diesem Modus wird die Kontur zusammen mit 5 sogenannten Pickpunkten dargestellt. Diese befinden sich in der Mitte und an den vier Ecken des umschließenden Rechtecks. Bei geschlossener Kontur werden diese Pickpunkte als Rechtecke dargestellt, ansonsten gleicht deren Form einem 'u'. Durch Drücken auf diese Pickpunkte kann eine geöffnete Kontur geschlossen werden und umgekehrt. Abhängig vom Zustand der Kontur können unterschiedliche Modifikationen vorgenommen werden. Geöffnete Kontur (Pickpunkte gleichen einem 'u')

- Um Punkte an die Kontur anzuhängen, muss mit der linken Maustaste ins Fenster geklickt werden. Hierdurch wird ein neuer Punkt an dieser Position eingefügt.
- Der zuletzt angehängte Punkt kann durch Drücken der Steuerungstaste (Strg, Ctrl) gelöscht werden.
- Zum Verschieben oder Einfügen von Punkten muss zunächst die Kontur geschlossen werden. Hierzu muss auf einen der Pickpunkte geklickt werden.

Geschlossene Kontur (rechteckige Pickpunkte)

- Zum Verschieben eines Punktes muss mit der linke Maustaste auf einen Punkt, markiert durch ein Rechteck, geklickt werden. Während man nun die Maustaste gedrückt hält, kann der Punkt verschoben werden.
- Wird die linke Maustaste in der Nähe einer Linie geklickt, so wird ein neuer Punkt eingefügt. Dieser kann, solange die linke Maustaste gedrückt wird, an die gewünschte Position verschoben werden.
- Zum Löschen eines Punktes muss ein Punkt mit der linken Maustaste selektiert werden und anschließend die Steuerungstaste (Strg, Ctrl) gedrückt werden.

Ein Klick mit der rechten Maustaste beendet die Prozedur. Unter macOS kann `draw_xld_mod` auch durch Drücken der Escape-Taste beendet werden.

Achtung

Für die Nutzung des Zeichenoperators in einem Puffer-Fenster muss die Anwendung Maus-Events senden. Der Zeichenoperator muss dabei in einem anderen Thread ausgeführt werden.

Parameter

▷ **`ContIn`** (input_object) ..... xld_cont ⇝ *object*
Eingabekontur

▷ **`ContOut`** (output_object) ..... xld_cont ⇝ *object*
Modifizierte Kontur.

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Rotate`** (input_control) ..... string ⇝ *string*
Rotation ermöglichen?
Defaultwert: 'true'
Werteliste: `Rotate` ∈ {'true', 'false'}

▷ **`Move`** (input_control) ..... string ⇝ *string*
Verschieben ermöglichen?
Defaultwert: 'true'
Werteliste: `Move` ∈ {'true', 'false'}

▷ **Scale** (input_control) ..... string ⇝ *string*
Skalieren ermöglichen?
Defaultwert: 'true'
Werteliste: Scale ∈ {'true', 'false'}

▷ **KeepRatio** (input_control) ..... string ⇝ *string*
Seitenverhältnis beim Skalieren beibehalten?
Defaultwert: 'true'
Werteliste: KeepRatio ∈ {'true', 'false'}

▷ **Edit** (input_control) ..... string ⇝ *string*
Editieren ermöglichen?
Defaultwert: 'true'
Werteliste: Edit ∈ {'true', 'false'}

Ergebnis

draw_xld_mod liefert den Wert TRUE, falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_rectangle2, draw_rectangle1, draw_rectangle2, draw_region

Siehe auch

gen_rectangle2, draw_circle, draw_ellipse, set_insert

Modul

Foundation

Kapitel 12

Identifizierung

12.1 Barcode

Dieses Kapitel gibt eine Einführung zum Thema Barcodes und den wichtigsten Begriffen beim Finden und Dekodieren von Barcodes in HALCON.

Aufbau von Linearen (1D) Barcodes

Ein Barcode besteht aus mehreren dunklen Balken und hellen Leerzeichen. Dabei gibt es ein kleinstmögliches Element, also der schmalst-mögliche Balken, bzw. das schmalst-mögliche Leerzeichen des Barcodes, welches Modul genannt wird. Die Breite der Balken und Leerzeichen ist im Allgemeinen unterschiedlich, aber stets ein vielfaches des Moduls. Mehrere Balken und Leerzeichen ergeben ein Muster. Dieses Muster kodiert ein Symbol, welches in einer Dekodiertabelle festgehalten ist. Die Konventionen, wie ein solches binäres Muster ein Symbol kodiert und wie die Dekodiertabelle aussieht, führen zum jeweiligen Barcode-Typen. Nebst den Mustern zum Kodieren der Symbole gibt es noch Muster mit speziellen Aufgaben. Je nach Typ gibt es weitere Muster, die vorkommen können oder müssen:

- Ruhezone (quiet-zone): Zone in der keine Kanten auftreten dürfen. Sie wird vor dem Start- und nach dem Stoppzeichen benötigt, um den Code zu finden.



Schematische Darstellung der Ruhezone (orange) anhand eines Beispiel-Barcodes des Typs Code 128.

- Start- und Stoppzeichen: Legen Start, Ende und Richtung fest. Damit kann der Code sowohl von links nach rechts als auch von rechts nach links (und somit auf dem Kopf stehend) gelesen und dekodiert werden.



Schematische Darstellung der Start- und Stoppzeichen (orange) anhand eines Beispiel-Barcodes des Typs Code 128.

- Prüfziffer: Ziffer, die als Sicherheitscheck dient. Mit ihr wird überprüft, ob der Code richtig gelesen wurde.



Schematische Darstellung der Prüfziffer (orange) anhand eines Beispiel-Barcodes des Typs Code 128.

- Guard-Pattern: Spezielle Muster, die links, rechts, in der Mitte oder als Zusatzsymbol vorkommen können.



Schematische Darstellung des Guard-Patterns (orange) anhand eines Beispiel-Barcodes des Typs UPC-A.

- Finder-Pattern: Spezielle Suchmuster zur Lokalisierung des Barcodes.



Schematische Darstellung des Finder-Patterns (orange) anhand eines Beispiel-Barcodes des Typ GS1 Databar.

- Add-on Symbol: Zusätzliche Symbole zur Kodierung ergänzender Informationen. Diese Symbole sind als Barcodes begrenzter Länge kodiert.



Schematische Darstellung zusätzlicher Symbole (orange) anhand eines Beispiel-Barcodes des Typs EAN-8 Add-On 2.

Aufbau von zusammengesetzten Barcodes

Zusammengesetzte Barcodes (composite bar codes) bestehen aus folgenden Elementen:

- Linearer Barcode: Wie oben beschrieben.
- 2D Code: Für Erklärungen zu den 2D Codes wird auf den „`Solution Guide II-C - 2D Data Codes`“ verwiesen.
- Verbindendes Muster: Verbindet die beiden vorher gelisteten Codes miteinander.

Die folgende Abbildung zeigt ein Schema eines zusammengesetzten Barcodes.



Schematische Darstellung eines zusammengesetzten Barcodes anhand eines Beispiel-Barcodes des Typs UPC-A Composite: Linearer Barcode (schwarz), 2D Code (hell-blau) und das verbindende Muster (orange).

Lesen eines Barcodes

Um auf einem Bild einen Barcode lesen zu können, muss dieser zuerst gefunden werden. Die Regionen, in welchen HALCON einen Barcode vermutet, werden Kandidaten genannt.

Zum Lesen eines Kandidaten werden verschiedene Linien durch die Region gelegt, sogenannte Scanlinien. Obwohl theoretisch eine einzelne Scanlinie reichen würde, wird entlang jeder Scanlinie versucht, den Barcode zu dekodieren. Dadurch steigt die Chance, den Code richtig zu dekodieren. Zudem ermöglichen mehrere erfolgreich dekodierte Scanlinien zusätzliche Plausibilitätschecks, d.h., werden mehrere Scanlinien gleich gelesen, so sinkt das Risiko, den Barcode falsch zu dekodieren.



Schematische Darstellung verschiedener Scanlinien eines Kandidaten. Dabei wurden gewisse Scanlinien erfolgreich gelesen (grün), während andere fehlschlugen (rot).

Der generelle Ablauf und verschiedene Beispiele, wie Barcodes in HALCON gelesen werden können, finden sich im „Solution Guide I - Basics.

clear_bar_code_model (: : BarCodeHandle :)

Löschen eines Barcode-Modells und Freigabe des verwendeten Speichers.

Der Operator clear_bar_code_model löscht ein mit create_bar_code_model angelegtes Barcode-Modell und gibt den gesamten darin referenzierten Speicherplatz wieder frei. Der Handle des zu löschenden 2D-Barcode-Modells muss in BarCodeHandle übergeben werden. Nach dem Aufruf ist dieser ungültig.

Parameter

▷ **BarCodeHandle** (input_control) ..... barcode(-array) ⇝ *handle*
Handle des zu löschenden Barcode-Modells.

Ergebnis

clear_bar_code_model liefert den Wert TRUE, falls der übergebene Handle gültig ist, und das darüber referenzierte Modell vollständig freigegeben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BarCodeHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

find_bar_code

Modul

Bar Code

create_bar_code_model (: : GenParamName,
GenParamValue : BarCodeHandle)

Anlegen eines generischen Modells für einen Barcode-Leser.

Mit Hilfe von create_bar_code_model wird ein generisches Modell für alle unterstützen Typen von Barcodes angelegt. Als Ergebnis liefert der Operator in BarCodeHandle einen Handle zurück, der für alle weiteren Barcode-Operationen verwendet wird, beispielsweise um das Modell zu modifizieren, um im Bild nach Symbolen zu suchen und diese zu lesen, oder um später auf die Ergebnisse der Symbolsuche zuzugreifen. Welcher Typ von Barcode in einem entsprechenden Bild gelesen werden möchte, wird erst mit dem Aufruf von find_bar_code festgelegt.

Im allgemeinen werden Barcodes ohne besondere Einstellung von Parametern erfolgreich in Bildern gefunden. Die Parameter GenParamName und GenParamValue werden daher standardmäßig als leere Tupel übergeben. Für den Fall, dass der Barcode oder die verwendeten Bilder Besonderheiten aufweisen, welche eine spezielle

Einstellung der möglichen Parameter erfordern, können diese bereits beim Anlegen des Modells gesetzt werden. Alternativ dazu ist auch das spätere Verändern von Parametern mit Hilfe des Operators `set_bar_code_param` bzw. `set_bar_code_param_specific` möglich. Siehe auch dort für eine detaillierte Beschreibung zur Einstellung von Parametern.

Parameter

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für das Barcode-Modell eingestellt werden sollen.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'check_char', 'composite_code', 'barcode_height_min', 'barcode_width_min', 'element_size_max', 'element_size_min', 'meas_thresh', 'meas_thresh_abs', 'min_identical_scanlines', 'num_scanlines', 'orientation', 'orientation_tol', 'persistence', 'start_stop_tolerance', 'stop_after_result_num', 'upce_encodation', 'timeout', 'train', 'quiet_zone', 'element_size_variable', 'min_code_length'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value(-array) ⇝ *real* / integer / string
Werte der generischen Parameter, die für das Barcode-Modell eingestellt werden sollen.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {0, 0.1, 1, 1.5, 2, 8, 32, 45, 'present', 'absent', 'none', 'CC-A/B', 'auto', 'high', 'low', 'true', 'false'}

▷ **`BarCodeHandle`** (output_control) . . . . . barcode ⇝ *handle*
Handle für den Zugriff auf das Barcode-Modell.

Ergebnis

`create_bar_code_model` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind und das Modell erzeugt werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`find_bar_code`

Siehe auch

`clear_bar_code_model`

Literatur

International Standard ISO/IEC 15420: „Information technology - Automatic identification and data capture techniques - Bar code symbology specification - EAN/UPC“; Reference number ISO/IEC 15420:2000 (E); ISO/IEC 2000.

International Standard ISO/IEC 16390: „Information technology - Automatic identification and data capture techniques - Bar code symbology specification - Interleaved 2 of 5“; Reference number ISO/IEC 16390:1999 (E); ISO/IEC 1999.

International Standard ISO/IEC 16388: „Information technology - Automatic identification and data capture techniques - Bar code symbology specification - Code 39“; Reference number ISO/IEC 16388:1999 (E); ISO/IEC 1999.

American National Standards Institute, Inc.(ANSI): „Uniform Symbology Specification Code 93“; ANSI/AIM-BC5-2000; AIM 2000.

International Standard ISO/IEC 15417: „Information technology - Automatic identification and data capture techniques - Bar code symbology specification - Code 128“; Reference number ISO/IEC 15417:2000 (E); ISO/IEC 2000.

American National Standards Institute, Inc.(ANSI): „Uniform Symbology Specification Codabar“; ANSI/AIM-BC3-2000; AIM 2000.

International Standard ISO/IEC 24724: „Information technology - Automatic identification and data capture techniques - Reduced Space Symbology (RSS) bar code symbology specification“; Reference number ISO/IEC 24724: 2006 (E); ISO/IEC 2006.

International Standard ISO/IEC 24723: „Information technology - Automatic identification and data capture techniques - EAN.UCC Composite bar code symbology specification“; Reference number ISO/IEC 24723:2006 (E); ISO/IEC 2006.

Modul

Bar Code

decode_bar_code_rectangle2 (Image : : BarCodeHandle, CodeType, Row, Column, Phi, Length1, Length2 : DecodedDataStrings)

Lesen von Barcode-Symbolen innerhalb eines Rechtecks.

Der Operator decode_bar_code_rectangle2 benutzt das durch BarCodeHandle angegebene Barcode-Modell um einen Barcode an einer vorgegebenen Position innerhalb des Bildes Image zu dekodieren. BarCodeHandle muss mit create_bar_code_model erstellt worden sein. Dessen Parameter können mit set_bar_code_param gesetzt werden. Die Position des Barcodes wird als beliebig orientiertes Rechteck übergeben. Im Gegensatz zu find_bar_code, wo der Dekodierung eine zeitaufwändige Suche nach Kandidatenregionen vorangeht, durchsucht decode_bar_code_rectangle2 die übergebene Region direkt nach Barcodes. Die rechteckige Region wird definiert mit den Parametern Row und Column für den Schwerpunkt, Phi für die Orientierung und Length1 und Length2 für die halbe Länge der Rechteckseiten (vergleiche gen_rectangle2). Der Winkel Phi legt auch die Leserichtung des Barcodes fest. Die Orientierung eines Barcodes ist der Winkel zwischen der Leserichtung des Barcodes und der horizontalen Bildachse. Der Winkel ist positiv bei Drehung nach links und wird wie bei gen_rectangle2 in Bogenmaß angegeben. Dies ist entgegengesetzt zu den Operatoren get_bar_code_result und set_bar_code_param bei denen Winkelangaben in Grad angegeben werden. Phi kann im Wertebereich von $[-\pi, +\pi]$ liegen. Die Leserichtung ist senkrecht zu den Kanten der einzelnen Barcodestriche. Es werden auch Barcodes mit einer Leserichtung von Phi $+ \pi$ zurückgegeben. Mehrere Regionen zur Dekodierung können als Tupel in Row, Column, Phi, Length1 und Length2 übergeben werden.

Das Rechteck sollte den Barcode und die Ruhezonen vollständig überdecken. Zu groß gewählte Regionen werden dekodiert, falls in dieser Region keine störenden Muster auftauchen und die Höhe (Length2) klein genug ist, dass die Scanlinien genügend dicht gelegt werden können. Rechtecke, die in Leserichtung (Length1) zu kurz gewählt sind, können nicht dekodiert werden.

decode_bar_code_rectangle2 kann dann verwendet werden, wenn die Position des Barcodes bereits im Voraus bekannt ist. Beispielsweise könnten durch find_bar_code und get_bar_code_object bereits Kandidatenregionen für Barcodes vorliegen (siehe folgendes Beispiel).

Für den Parameter CodeType können mehrere Barcode-Typen angegeben werden, vergleiche Abschnitt Autodiskriminierung für find_bar_code.

Die weiteren Aspekte der tatsächlichen Dekodierung werden bei dem Operator find_bar_code beschrieben.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object* : byte / uint2
  Eingabebild.
- ▷ **BarCodeHandle** (input_control) ..... barcode ⇝ *handle*
  Handle des zu verwendenden Barcode-Modells.
- ▷ **CodeType** (input_control) ..... string(-array) ⇝ *string*
  Typ des zu findenden Barcodes.
  Defaultwert: 'EAN-13'
  Werteliste: CodeType ∈ {'2/5 Industrial', '2/5 Interleaved', 'Codabar', 'Code 39', 'Code 93', 'Code 128', 'EAN-13', 'EAN-13 Add-On 2', 'EAN-13 Add-On 5', 'EAN-8', 'EAN-8 Add-On 2', 'EAN-8 Add-On 5', 'UPC-A', 'UPC-A Add-On 2', 'UPC-A Add-On 5', 'UPC-E', 'UPC-E Add-On 2', 'UPC-E Add-On 5', 'MSI', 'PharmaCode', 'GS1 DataBar Omnidir', 'GS1 DataBar Truncated', 'GS1 DataBar Stacked', 'GS1 DataBar Stacked Omnidir', 'GS1 DataBar Limited', 'GS1 DataBar Expanded', 'GS1 DataBar Expanded Stacked', 'GS1-128', 'auto'}

▷ **Row** (input_control) . . . . . rectangle2.center.y(-array) ⇝ *real* / integer
Zeilenindex des Schwerpunktes.
Defaultwert: 50.0

▷ **Column** (input_control) . . . . . rectangle2.center.x(-array) ⇝ *real* / integer
Spaltenindex des Schwerpunktes.
Defaultwert: 100.0

▷ **Phi** (input_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real* / integer
Orientierung des Rechtecks in Bogenmaß.
Defaultwert: 0.0
Wertevorschläge: `Phi` ∈ {0.0, 0.785398, 1.570796, 3.1415926}

▷ **Length1** (input_control) . . . . . rectangle2.hwidth(-array) ⇝ *real* / integer
Hälfte der Seitenlänge des Rechtecks entlang der Leserichtung des Barcodes
Defaultwert: 200.0

▷ **Length2** (input_control) . . . . . rectangle2.hheight(-array) ⇝ *real* / integer
Hälfte der Seitenlänge des Rechtecks senkrecht zur Leserichtung des Barcodes.
Defaultwert: 100.0

▷ **DecodedDataStrings** (output_control) . . . . . string(-array) ⇝ *string*
Die Datenstrings der erfolgreich dekodierten Barcodes.

Beispiel

```
read_image (Image, 'barcode/ean13/ean1301.png')
create_bar_code_model ([], [], BHandle)
find_bar_code (Image, SymReg, BHandle, '2/5 Industrial', Dec)
if (|Dec| == 0)
*     A 2/5 Industrial code wasn't found. Try decoding an EAN-13 code.
    get_bar_code_object (CandReg, BHandle, 'all', 'candidate_regions')
    smallest_rectangle2 (CandReg, R, C, Phi, L1, L2)
    decode_bar_code_rectangle2 (Image, BHandle, 'EAN-13', R, C, Phi, \
                                L1, L2, Dec)
endif
```

Ergebnis

`decode_bar_code_rectangle2` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BarCodeHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_bar_code_model`, `set_bar_code_param`, `smallest_rectangle2`

Nachfolger

`get_bar_code_result`, `get_bar_code_object`, `clear_bar_code_model`

Alternativen

`find_bar_code`

Siehe auch

`gen_rectangle2`

Modul

Bar Code

deserialize_bar_code_model (
: : SerializedItemHandle : BarCodeHandle)

Deserialisieren eines Barcode-Modells.

deserialize_bar_code_model deserialisiert ein Barcode-Modell, welches mit serialize_bar_code_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Modell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Barcode-Modell mit dem Handle BarCodeHandle gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **BarCodeHandle** (output_control) ..... barcode ⇝ *handle*
  Handle für das deserialisierte Barcode-Modell.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_bar_code_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_bar_code_model

Nachfolger

find_bar_code

Alternativen

create_bar_code_model

Siehe auch

serialize_bar_code_model, clear_bar_code_model

Modul

Bar Code

find_bar_code (Image : SymbolRegions : BarCodeHandle,
CodeType : DecodedDataStrings)

Suchen und Lesen von Barcode-Symbolen im Eingabebild.

Der Operator find_bar_code sucht im Eingabebild (Image) nach Barcode-Symbolen und dekodiert die darin enthaltenen Zeichenfolgen. In einem Bild kann eine beliebige Anzahl an Barcodes eines Typs gefunden und dekodiert werden, wobei der Typ des gesuchten Barcodes mit CodeType ausgewählt wird. Die gelesenen Zeichenfolgen werden in DecodedDataStrings und die dazu korrespondierenden Barcode-Regionen in SymbolRegions zurück gegeben. Werden insgesamt n Barcodes erfolgreich gelesen, so können die Indizes von 0 bis (n-1) in den Operatoren get_bar_code_result und get_bar_code_object verwendet werden, um die gewünschte Information eines speziellen Ergebnisse zu erhalten.

Es wird angenommen, dass die Barcodes dunkel auf einem hellen Hintergrund erscheinen. Um helle Barcodes auf einem dunklen Hintergrund lesen zu können, muss das Eingabebild vorher mit dem Operator invert_image invertiert werden.

Vor dem Aufruf des Operators find_bar_code muss mit Hilfe von create_bar_code_model ein Barcode-Modell erstellt und der dabei erzeugte Handle in BarCodeHandle an find_bar_code übergeben werden.

Der Ausgabewert DecodedDataStrings gibt für jedes Ergebnis die dekodierte Zeichenfolge zurück. Der Inhalt dieser Zeichenfolge wird durch den Standard des entsprechenden Codetyps bestimmt. Typischerweise enthält DecodedDataStrings nur die Zeichen mit Dateninformation. Bei Codes mit obligatorischem Prüfzeichen wird dieses nicht mit ausgegeben. Bei Codes mit optionalem Prüfzeichen, z.B. Code 39, Codabar, 2/5 Industrial oder 2/5 Interleaved, hängt das Ergebnis von der Einstellung des Parameters *'check_char'* ab, welcher in create_bar_code_model, set_bar_code_param oder set_bar_code_param_specific einstellbar ist: Der Standardwert von *'absent'* gibt an, dass kein Prüfzeichen vorliegt. In diesem Fall wird keine Prüfung durchgeführt, und alle Zeichen werden als Daten zurückgeliefert. Ist dieser Parameter auf *'present'* gesetzt, so wird ein Prüfzeichen erwartet und benutzt um die Korrektheit des Barcodes zu prüfen. Ist die Prüfsumme fehlerhaft, so wird der Code als nicht lesbar eingestuft und dementsprechend auch nicht als Ergebnis in DecodedDataStrings und SymbolRegions zurückgeliefert. Das Prüfzeichen selbst wird aus den Nutzdaten entfernt. Falls dies nicht erwünscht ist, kann der Parameterwert *'preserved'* eingestellt werden um das Prüfzeichen trotz Prüfung zu erhalten.

Die zu Grunde liegenden Referenz-Zeichen, einschließlich möglicher Start- oder Stopzeichen sowie Prüfzeichen, können mit get_bar_code_result und *'decoded_reference'* abgefragt werden.

Folgende Barcode-Symbole werden unterstützt:

2/5 Industrial	EAN-8	GS1-128
2/5 Interleaved	EAN-8 Add-On 2	GS1 DataBar Omnidirectional
Codabar	EAN-8 Add-On 5	GS1 DataBar Truncated
Code 39	EAN-13	GS1 DataBar Stacked
Code 32 (converted from Code 39)	EAN-13 Add-On 2	GS1 DataBar Stacked Omnidirectional
Code 93	EAN-13 Add-On 5	GS1 DataBar Limited
Code 128	UPC-A	GS1 DataBar Expanded
MSI	UPC-A Add-On 2	GS1 DataBar Expanded Stacked
PharmaCode	UPC-A Add-On 5	
	UPC-E	
	UPC-E Add-On 2	
	UPC-E Add-On 5	

Anmerkung: Da die Leserichtung beim PharmaCode nicht bestimmt werden kann, werden auch die beiden Zeichenketten für jede Leserichtung zurückgegeben. Beide Zeichenketten werden in einer Zeichenkette durch ein Komma getrennt aneinandergehängt.

Barcodes vom Typ GS1 DataBar können zusätzlich eine Composite-Code-Komponente enthalten. Um diese Komponente finden und dekodieren zu können, muss mittels des Operators set_bar_code_param der Parameter *'composite_code'* auf *'CC-A/B'* gesetzt werden.

Es können auch Barcodes vom Typ 'Code 32' gelesen werden, indem der Operator find_bar_code mit CodeType *'Code 39'* in Kombination mit der externen Prozedur convert_decoded_string_code39_to_code32 verwendet wird.

Autodiskriminierung

Autodiskriminierung beschreibt das gleichzeitige Lesen mehrerer Barcode-Typen in einem Aufruf von find_bar_code. Dazu wird dem Parameter CodeType ein Tupel von möglichen Barcode-Typen übergeben. Mit dem speziellen Wert *'auto'* werden alle Barcode-Typen dekodiert, mit Ausnahme von *'PharmaCode'* und *'MSI'*, da diese zu wenige Merkmale bieten um von anderen Barcode-Typen abgegrenzt werden zu können. In dem Tupel können auch Barcode-Typen mit vorangestellter Tilde (~) angegeben werden, welche dann nicht dekodiert werden. So bezeichnet beispielsweise

```
['auto', '~EAN-8', '~EAN-8 Add-On 2', '~EAN-8 Add-On 5']
```

alle Barcode-Typen ohne *'PharmaCode'*, *'MSI'* und *'EAN-8'*-Arten. Zu beachten ist, dass jeder zusätzlich erlaubte Barcode-Typ die Laufzeit des Operators verlängert. Da eventuell nicht alle Barcodes definitiv voneinander unterschieden werden können nimmt ggf. die Zuverlässigkeit der Dekodierung mit jedem zusätzlichen Barcode-Typ ab. Um die Kompatibilität der Autodiskriminierung zu erhöhen sollten falls möglich immer Barcodes mit Prüfzeichen bzw. Prüfsumme verwendet werden.

Der Barcode-Leser versucht die Barcode-Typen in der folgenden Reihenfolge zu dekodieren:

'GS1 DataBar Omnidir', *'GS1 DataBar Truncated'*, *'GS1 DataBar Stacked'*, *'GS1 DataBar Stacked Omnidir'*, *'GS1 DataBar Limited'*, *'GS1 DataBar Expanded'*, *'GS1 DataBar Expanded Stacked'*, *'GS1-128'*, *'Code 128'*, *'EAN-13 Add-On 5'*, *'EAN-13 Add-On 2'*, *'EAN-13'*, *'UPC-A Add-On 5'*, *'UPC-A Add-On 2'*, *'UPC-A'*, *'EAN-8 Add-On 5'*, *'EAN-8 Add-On 2'*, *'EAN-8'*, *'UPC-E Add-On 5'*, *'UPC-E Add-On 2'*, *'UPC-E'*, *'Code 93'*, *'Code 39'*, *'Codabar'*, *'2/5 Interleaved'*, *'2/5 Industrial'*

Deshalb sollten mindestens alle nicht auftretenden Barcode-Typen ausgeschlossen werden die vor dem ersten der erwarteten Barcode-Typen dekodiert werden oder gleich nur die explizite Liste aller Barcode-Typen die erwartet werden.

Insbesondere für den Fall der Autodiskriminierung existiert der Operator `set_bar_code_param_specific`. Einige Parameter des Barcode-Modells können damit spezifisch für verschiedene Barcode-Typen festgelegt werden.

Training

Falls sich der Barcode-Leser im Trainingsmodus befindet führt der Operator `find_bar_code` einen Trainingslauf durch. Der Trainingmodus ist bei dem Operator `set_bar_code_param` beschrieben.

Timeout und Abbruch

Der Operator `find_bar_code` kann sowohl mit einem Timeout sowie auch manuell abgebrochen werden. Mit dem Operator `set_bar_code_param` kann man einen Timeout festlegen. Falls `find_bar_code` in einen Timeout läuft, werden alle bis dahin erkannten Codes zurückgegeben. Zusätzlich gibt es die Möglichkeit `find_bar_code` manuell abzubrechen, indem man `set_bar_code_param` mit *'abort'* von einem anderen Thread aufruft.

In beiden Fällen, kann mit `get_bar_code_result` und dem Parameter *'aborted'* abgefragt werden, ob der Operator abgebrochen wurde.

Des Weiteren kann der Operator `find_bar_code` mittels `set_operator_timeout` oder `interrupt_operator` abgebrochen werden. Dabei werden keine Ergebnisse, aber eine Fehlermeldung zurückgegeben. Wenn `find_bar_code` mit `interrupt_operator` abgebrochen wurde, wird H_ERR_CANCEL (22) zurückgegeben. Wenn `find_bar_code` mit `set_operator_timeout` abgebrochen wurde, wird H_ERR_TIMEOUT (9400) zurückgegeben. Das Abbrechen durch einen dieser Operatoren wird nur im *'cancel'*-Modus unterstützt.

Hinweise zur Verwendung von niedrig aufgelösten Bildern

Dieser Hinweis betrifft Barcode-Anwendungen, in denen niedrig aufgelöste Bilder verwendet werden, d.h. Bilder in denen ein einzelner Strich des Barcodes nur zwischen 0.6 und 2 Pixeln breit ist.

In diesen Fällen sollte man bei der Bildaufnahme darauf achten, dass die Bilder möglichst scharf sind und gleichzeitig möglichst wenig Rauschen enthalten, da diese Fehler zu Problemen bei der Dekodierung von niedrig aufgelösten Barcodes führen können. Im Abschnitt *'small_elements_robustness'* des Operators `set_bar_code_param` finden sich weitere wichtige Hinweise.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object* : byte / uint2
  Eingabebild. Falls das Bild eine reduzierte Domäne besitzen sollte, so wird der Bereich eingeschränkt in dem nach Barcodes gesucht wird. In der Regel führt dies zu einer Verbesserung der Laufzeit. Allerdings sollte man beachten, dass ein Barcode nicht mehr gefunden wird könnte, wenn er nicht vollständig in der Domäne des Bildes liegt.
- ▷ **`SymbolRegions`** (output_object) ..... region(-array) ⇝ *object*
  Regionen der erfolgreich dekodierten Barcode-Symbole.
- ▷ **`BarCodeHandle`** (input_control) ..... barcode ⇝ *handle*
  Handle des zu verwendenden Barcode-Modells.
- ▷ **`CodeType`** (input_control) ..... string(-array) ⇝ *string*
  Typ des zu findenden Barcodes.
  Defaultwert: 'auto'
  Werteliste: `CodeType` ∈ {'auto', '2/5 Industrial', '2/5 Interleaved', 'Codabar', 'Code 39', 'Code 93', 'Code 128', 'EAN-13', 'EAN-13 Add-On 2', 'EAN-13 Add-On 5', 'EAN-8', 'EAN-8 Add-On 2', 'EAN-8 Add-On 5', 'UPC-A', 'UPC-A Add-On 2', 'UPC-A Add-On 5', 'UPC-E', 'UPC-E Add-On 2', 'UPC-E Add-On 5', 'MSI', 'PharmaCode', 'GS1 DataBar Omnidir', 'GS1 DataBar Truncated', 'GS1 DataBar Stacked', 'GS1 DataBar Stacked Omnidir', 'GS1 DataBar Limited', 'GS1 DataBar Expanded', 'GS1 DataBar Expanded Stacked', 'GS1-128'}

▷ **DecodedDataStrings** (output_control) . . . . . string(-array) ⇝ *string*
Die Datenstrings der erfolgreich dekodierten Barcodes.

Beispiel

```
* Small example for aborting bar code search:
read_image (Image, 'barcode/ean13/tea_box_10.png')

* Decode normally:
create_bar_code_model ([], [], BarCodeHandle)
find_bar_code (Image, SymbolRegions, BarCodeHandle, 'auto', \
    DecodedDataStrings)
stop ()

* Start bar code search in separate thread.
par_start<ThreadID>: find_bar_code (Image, SymbolRegions1, BarCodeHandle, \
    'auto', DecodedDataStrings1)

* Let some time pass.
wait_seconds (0.002)

* Abort the data code search.
set_bar_code_param (BarCodeHandle, 'abort', 'true')

* Wait for the thread, it should finish very fast.
par_join(ThreadID)

* We can determine that the search in fact was aborted.
get_bar_code_result (BarCodeHandle, 'all', 'aborted', Aborted)
```

Ergebnis

`find_bar_code` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BarCodeHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_bar_code_model`, `set_bar_code_param`

Nachfolger

`get_bar_code_result`, `get_bar_code_object`, `clear_bar_code_model`

Alternativen

`decode_bar_code_rectangle2`

Modul

Bar Code

get_bar_code_object (: BarCodeObjects : BarCodeHandle, CandidateHandle, ObjectName :)

Zugriff auf ikonische Objekte, die bei der Suche und der Dekodierung von Barcodes angelegt werden bzw. diese beschreiben.

Der Operator get_bar_code_object erlaubt den Zugriff auf ikonische Objekte, die beim letzten Aufruf von find_bar_code während der Suche nach Barcodes oder beim Lesen der gefundenen Symbole angelegt wurden. Für den Zugriff muss neben dem Namen des gewünschten Bildobjekts (ObjectName) auch das zugrundeliegende Barcode-Modell (BarCodeHandle) angegeben werden. Außerdem muss in CandidateHandle ein Index auf ein dekodiertes Symbol oder einen Symbolkandidaten, bzw. *'all'* für alle dekodierten Symbole oder Symbolkandidaten übergeben werden.

Je nachdem wie der Parameter ObjectName ausgewählt ist, werden die Ausgabeobjekten in BarCodeObjects zurückgeliefert, wie folgend:

'symbol_regions': Die Regionen von erfolgreich dekodierten Symbolen werden zurückgegeben. Die Reihenfolge der Objekte entspricht der Reihenfolge der dekodierten Zeichenfolgen im Operator get_bar_code_result.

'candidate_regions': Die potentiellen Barcode-Regionen werden ausgegeben. Gibt es insgesamt m Kandidaten und n erfolgreiche Ergebnisse, so können mit CandidateHandle gleich 0 bis (n-1) die ursprünglichen Regionen der entsprechenden Ergebnisse und mit CandidateHandle gleich n bis (m-1) die Regionen der erfolglosen Kandidaten abgerufen werden.

'scanlines_all', *'scanlines_valid'*, *'scanlines_all_plain'*, *'scanlines_valid_plain'*, *'scanlines_merged_edges'*: Es werden XLD Konturen ausgegeben, welche die einzelnen Striche innerhalb der Scanlinien darstellen. Das Auslesen der Scanlinien kann nur erfolgen, wenn der Barcode mit find_bar_code oder decode_bar_code_rectangle2 im *'persistence'*-Modus gelesen wurde (siehe set_bar_code_param). *'scanlines_all'* stellt alle Scanlinien dar, die von find_bar_code möglicherweise angelegt werden, um einen Barcode zu dekodieren. *'scanlines_valid'* stellt nur diejenige *'scanlines_all'* dar, die erfolgreich dekodiert werden können. Bei einzeiligen Barcodes gibt es mindestens eine *'scanlines_valid'* je erfolgreich dekodiertem Barcodesymbol. Im Gegensatz dazu liefert *'scanlines_valid'* leere Objekte wenn die Dekodierung des Symbols erfolglos war. Diese Regel ist für gestapelte Codes (wie z.B. *'GS1 DataBar Stacked'* und *'GS1 DataBar Expanded Stacked'*) ähnlich, allerdings bezieht sie sich auf die einzelnen Zeilen, anstatt auf das ganze Barcodesymbol. *'scanlines_merged_edges'* zeigt die Scanlinien und Kanten, die für die Dekodierung der vereinigten Scanlinie verwendet wurden (Siehe *'merge_scanlines'* in set_bar_code_param).

'scanlines_all' stellt alle Scanlinien dar, die von find_bar_code möglicherweise angelegt werden, um einen Barcode zu dekodieren.

'scanlines_valid' stellt nur diejenigen Scanlinien dar, die erfolgreich dekodiert werden können. Die Mindestanzahl an Scanlinien, die erfolgreich gelesen werden können müssen, kann mit set_bar_code_param mit dem Parameter *'min_identical_scanlines'* angepasst werden. Wenn die Dekodierung des Symbols erfolglos war, liefert *'scanlines_valid'* leere Objekte. Diese Regel ist für gestapelte Codes (wie z.B. *'GS1 DataBar Stacked'* und *'GS1 DataBar Expanded Stacked'*) ähnlich, allerdings bezieht sie sich auf die einzelnen Zeilen, anstatt auf das ganze Barcodesymbol.

'scanlines_merged_edges' zeigt die Scanlinien und Kanten, die für die Dekodierung der vereinigten Scanlinien verwendet wurden (siehe 'merge_scanlines' in set_bar_code_param).

get_bar_code_object gibt jede Scanlinie als eigene XLD Kontur zurück. Die XLD Konturen, die von *'scanlines_all'* und *'scanlines_valid'* zurückgegeben werden, sind nur zur Visualisierung gedacht und bestehen deshalb aus vielen XLD Punkten. Doppelte Linien werden zwischen Kanten benutzt, die als Striche erkannt wurden, während einfache Linien für weiße Lücken benutzt werden. Zusätzliche Informationen über jede Scanlinie können mit get_bar_code_result mit dem Parameter *'status'* erhalten werden.

Die *'_plain'*-Varianten *'scanlines_all_plain'* und *'scanlines_valid_plain'* geben die o.g. Scanlinien in vereinfachter Form zurück. Jede gefundene Kante wird hier als ein Punkt einer XLD-Kontur zurückgegeben, wobei für jede Scanlinie eine neue XLD-Kontur begonnen wird. Scanlinien, für die keine Kante gefunden wurde, werden nicht zurückgegeben.

Parameter

▷ **BarCodeObjects** (output_object) . . . . . object(-array) ⇝ *object*
Ikonische Objekte, die als Zwischenergebnisse bei der Barcode-Suche oder deren Auswertung anfallen.

▷ **BarCodeHandle** (input_control) . . . . . barcode ⇝ *handle*
Handle des zu verwendenden Barcode-Modells.

▷ **CandidateHandle** (input_control) . . . . . integer ⇝ *string* / integer
Auswahl der Barcode-Ergebnisse oder Kandidaten, für die eine Information abgefragt werden soll.
Defaultwert: 'all'
Wertevorschläge: `CandidateHandle` ∈ {0, 1, 2, 'all'}

▷ **ObjectName** (input_control) . . . . . string ⇝ *string*
Name des ikonischen Objekts, das abgefragt werden soll.
Defaultwert: 'candidate_regions'
Werteliste: `ObjectName` ∈ {'symbol_regions', 'candidate_regions', 'scanlines_all', 'scanlines_valid', 'scanlines_merged_edges', 'scanlines_all_plain', 'scanlines_valid_plain'}

Ergebnis

`get_bar_code_object` liefert den Wert TRUE, falls die übergebenen Parameter korrekt und die angeforderten Objekte für die letzte Symbolsuche verfügbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_bar_code`

Siehe auch

`get_bar_code_result`

Modul

Bar Code

get_bar_code_param (: : BarCodeHandle,
GenParamName : GenParamValue)

Abfrage von einem oder mehreren Parameter, die das Barcode-Modell beschreiben.

Mit Hilfe des Operators `get_bar_code_param` lassen sich die Parameter, durch die das Barcode-Modell beschrieben wird und die für eine erfolgreiche Suche und Dekodierung eines Barcodes benötigt werden, abfragen.

Die Namen der Parameter, die abgefragt werden sollen, werden in `GenParamName` übergeben, die korrespondierenden Werte werden in `GenParamValue` zurückgeliefert. Sämtliche Parameter können mit `set_bar_code_param` gesetzt und jederzeit verändert werden. Siehe auch dort für eine detaillierte Beschreibung der Bedeutung der Parameter.

Einige Parameter können durch `set_bar_code_param_specific` oder während des automatischen Parametertrainings spezifisch für einen Barcode- Typ gesetzt werden. Diese Parameter können nicht mit `get_bar_code_param`, sondern müssen mit `get_bar_code_param_specific` ausgelesen werden. Alle Parameter, die in `get_bar_code_param_specific` aufgelistet sind, können prinzipiell spezifisch gesetzt sein. Welche Parameter zur Laufzeit tatsächlich spezifisch gesetzt sind, kann mit dem Operator `query_bar_code_params` bestimmt werden.

Im Einzelnen kann auf die folgenden Parameter – gruppiert nach verschiedenen Kategorien – zugegriffen werden:

Größe der Barcode Elemente:

'element_size_min': Die minimale Größe eines Basiselements, d.h. die minimale Dicke der schmalsten Striche und Zwischenräume des speziellen Barcodetyps.

'element_size_max': Die maximale Größe eines Basiselements, d.h. die maximale Dicke der schmalsten Striche und Zwischenräume des speziellen Barcodetyps.

'element_size_variable': Gibt an, ob die Größe eines Elements innerhalb eines Barcodes variiert.

'barcode_height_min': Die minimale Höhe des Barcodes.

'barcode_width_min': Die minimal Breite des Barcodes.

Scannen:

'num_scanlines': Maximale Anzahl von Scanlinien per Kandidatregion.

'min_identical_scanlines': Minimale Anzahl von Scanlinien, welche identische Ergebnisse liefern müssen, damit die Dekodierung eines Symbols akzeptiert wird.

'majority_voting': Steuert das Verhalten bei der Auswahl unterschiedlicher Dekodierergebnisse pro Scanlinie. Dieser Parameter spezifiziert ob ein Mehrheitsvotum durchgeführt werden soll.

'stop_after_result_num': Anzahl erfolgreich dekodierter Barcodes nach der die Dekodierung angehalten wird.

'orientation': Zugelassene Orientierung der dekodierten Barcodes.

'orientation_tol': Toleranz der zugelassenen Orientierung.

'quiet_zone': Verifikation der Ruhezonen.

'start_stop_tolerance': Toleranz des Matchingkriteriums zum Start-, bzw. Stoppmuster.

'min_code_length': Minimale Anzahl dekodierter Zeichen.

'merge_scanlines': Steuert, ob vereinigte Scanlinie berechnet werden sollen, wenn nicht genügend einzelne Scanlinien erfolgreich dekodiert werden konnten. Somit können verdeckte oder beschädigte Barcodes gelesen werden.

Erscheinungsbild des Barcodes:

'meas_thresh': Relativer Schwellenwert zur Detektion von Kanten in der Barcode Region.

'meas_thresh_abs': Absoluter Schwellenwert zur Detektion von Kanten in der Barcode Region.

'contrast_min': Minimaler Kontrast zwischen dem Vorder- und Hintergrund der Barcode-Elemente.

Barcode spezifische Parameter:

'check_char': Berücksichtigung eines optionalen Prüfzeichens.

'composite_code': Vorhandensein und Typ eines an den Barcode angehängten 2D GS1-Composite Code.

'upce_encodation': Ausgabeformat für UPC-E-Barcodes.

Sonstiges:

'timeout': Vorzeitiger Abbruch von `find_bar_code` bei Zeitüberschreitung.

'persistence': Steuert, ob bestimmte Zwischenergebnisse bei der Symbolsuche mit `find_bar_code` temporär oder persistent im Modell gespeichert werden sollen.

'train': Gibt die *Namen* der momentan trainierten Parameter zurück. Das zurückgelieferte Tupel kann anschließend als `GenParamName` an `get_bar_code_param` übergeben werden, um die trainierten *Werte* der trainierten Parameter zu bekommen.

Für weitere Erklärungen, siehe `set_bar_code_param`.

Parameter

▷ **`BarCodeHandle`** (input_control) . . . . . barcode ⇝ *handle*
Handle des zu verwendenden Barcode-Modells.

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für den Barcode abgefragt werden sollen.
Defaultwert: 'element_size_min'
Werteliste: `GenParamName` ∈ {'check_char', 'composite_code', 'barcode_height_min', 'barcode_width_min', 'element_size_max', 'element_size_min', 'contrast_min', 'meas_thresh', 'meas_thresh_abs', 'min_identical_scanlines', 'majority_voting', 'num_scanlines', 'orientation', 'orientation_tol', 'persistence', 'start_stop_tolerance', 'stop_after_result_num', 'timeout', 'train', 'upce_encodation', 'quiet_zone', 'element_size_variable', 'min_code_length', 'merge_scanlines', 'small_elements_robustness'}

▷ **GenParamValue** (output_control) . . . . . attribute.name(-array) ⇝ *real* / integer / string
Werte der generischen Parameter.

Ergebnis

`get_bar_code_param` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_bar_code_model`, `set_bar_code_param`

Nachfolger

`set_bar_code_param`

Alternativen

`get_bar_code_param_specific`

Modul

Bar Code

get_bar_code_param_specific (: : BarCodeHandle, CodeTypes, GenParamName : GenParamValue)

Abfrage von Parametern die zum Lesen eines Barcodes benutzt werden falls ein spezifischer Barcodetyp verarbeitet wird.

Mit Hilfe des Operators `get_bar_code_param_specific` lassen sich die Parameter, durch die das Barcode-Modell beschrieben wird, abfragen. Im Gegensatz zu `get_bar_code_param` lassen sich mit `get_bar_code_param_specific` Parameter spezifisch für angegebene Barcodetypen auslesen. Dies ist nützlich, wenn nach unterschiedlichen Barcodetypen innerhalb eines Bildes gesucht wird.

Die Barcodetypen werden in `CodeTypes` übergeben. Die Namen der Parameter, die abgefragt werden sollen, werden in `GenParamName` übergeben, die korrespondierenden Werte werden in `GenParamValue` zurückgeliefert. Sämtliche Parameter können selektiv mit `set_bar_code_param_specific` gesetzt und verändert werden oder für alle Barcodetypen mit `set_bar_code_param`. Siehe `set_bar_code_param` für eine detaillierte Beschreibung der Bedeutung der Parameter.

Welche Parameter zur Laufzeit spezifisch gesetzt sind, kann mit dem Operator `query_bar_code_params` bestimmt werden. Diese Parameter können nicht mit `get_bar_code_param`, sondern müssen mit `get_bar_code_param_specific` ausgelesen werden.

Im Einzelnen kann auf die folgenden Parameter zugegriffen werden:

'num_scanlines': Maximale Anzahl von Scanlinien per Kandidatregion.

'min_identical_scanlines': Minimale Anzahl von Scanlinien, welche identische Ergebnisse liefern müssen, damit die Dekodierung eines Symbols akzeptiert wird.

'stop_after_result_num': Anzahl erfolgreich dekodierter Barcodes nach der die Dekodierung angehalten wird.

'orientation': Zugelassene Orientierung der dekodierten Barcodes.

'orientation_tol': Toleranz der zugelassenen Orientierung.

'quiet_zone': Verifikation der Ruhezonen des Barcodesymbols.

'start_stop_tolerance': Toleranz des Matchingkriteriums zum Start-, bzw. Stoppmuster.

'check_char': Berücksichtigung eines optionalen Prüfzeichens.

'composite_code': Vorhandensein und Typ eines an den Barcode angehängten 2D GS1-Composite Code.

'min_code_length': Minimale Anzahl dekodierter Zeichen.

Für weitere Erklärungen, siehe `set_bar_code_param`.

Parameter

- ▷ **BarCodeHandle** (input_control) . . . . . barcode ⇝ *handle*
  Handle des zu verwendenden Barcode-Modells.
- ▷ **CodeTypes** (input_control) . . . . . string(-array) ⇝ *string*
  Namen der Barcodetypen für die Parameter abgefragt werden sollen.
  Defaultwert: 'EAN-13'
  Werteliste: `CodeTypes` ∈ {'2/5 Industrial', '2/5 Interleaved', 'Codabar', 'Code 39', 'Code 93', 'Code 128', 'EAN-13', 'EAN-13 Add-On 2', 'EAN-13 Add-On 5', 'EAN-8', 'EAN-8 Add-On 2', 'EAN-8 Add-On 5', 'UPC-A', 'UPC-A Add-On 2', 'UPC-A Add-On 5', 'UPC-E', 'UPC-E Add-On 2', 'UPC-E Add-On 5', 'MSI', 'PharmaCode', 'GS1 DataBar Omnidir', 'GS1 DataBar Truncated', 'GS1 DataBar Stacked', 'GS1 DataBar Stacked Omnidir', 'GS1 DataBar Limited', 'GS1 DataBar Expanded', 'GS1 DataBar Expanded Stacked', 'GS1-128'}
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Namen der generischen Parameter, die für den Barcode abgefragt werden sollen.
  Defaultwert: 'check_char'
  Werteliste: `GenParamName` ∈ {'check_char', 'composite_code', 'min_identical_scanlines', 'num_scanlines', 'orientation', 'orientation_tol', 'start_stop_tolerance', 'stop_after_result_num', 'quiet_zone', 'min_code_length'}
- ▷ **GenParamValue** (output_control) . . . . . attribute.name(-array) ⇝ *real* / integer / string
  Werte der generischen Parameter.

Ergebnis

`get_bar_code_param_specific` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_bar_code_model`, `set_bar_code_param`, `set_bar_code_param_specific`

Alternativen

`get_bar_code_param`

Modul

Bar Code

get_bar_code_result (: : BarCodeHandle, CandidateHandle, ResultName : BarCodeResults)

Abfrage von alphanumerischen Ergebnissen, die bei der Dekodierung von Barcode-Symbolen angefallen sind.

Der Operator `get_bar_code_result` gibt alphanumerische Ergebnisse des Lesevorgangs zurück. Um ein bestimmtes Ergebnis zu erhalten, sind das Barcode-Modell (`BarCodeHandle`) und der Index des gelesenen Symbols oder Kandidaten (`CandidateHandle`) notwendig.

`CandidateHandle` bezieht sich auf die gelesenen Kandidaten in der Anordnung wie sie vom Operator `find_bar_code` zurück gegeben werden. Dies bedeutet, die möglichen Werte für `CandidateHandle` hängen vom zurückgegebenen Ergebnis ab. Bezieht es sich auf gelesene Symbole, sind Werte von 0 bis (n-1) möglich (wobei n die Gesamtzahl an erfolgreich gelesener Barcodes ist). Bezieht es sich auf Kandidaten, sind Werte von 0 bis (m-1) möglich (wobei m die Gesamtzahl an Kandidaten ist). Ist das gewünschte Ergebnis ein Einzelwert, kann `CandidateHandle` auch auf *'all'* gesetzt werden, um die Ergebnisse aller Symbole oder Kandidaten zurückzugeben.

In `BarCodeResults` werden Ergebnisse zurückgeliefert, je nachdem was im `ResultName` ausgewählt wurde:

'decoded_strings': Die dekodierte Zeichenfolge des Symbols wird zurückgegeben. Ist `CandidateHandle` zum Beispiel gleich 0 gesetzt, so wird nur die Zeichenfolge für das erste Ergebnis ausgegeben. Steht

CandidateHandle auf *'all'* so gibt der Operator die Zeichenfolgen für alle Ergebnisse zurück. Es werden nur die reinen Datenzeichen zurückgegeben. Start-, Stop- und Prüfzeichen werden entfernt. Für optionale Prüfzeichen hängt das Verhalten vom Wert des Parameters *'check_char'* ab.

'decoded_types': Der Typ des gefundenen Barcodes wird zurückgegeben. Dies ist insbesondere im Zusammenhang mit der Autodiskriminierung (siehe find_bar_code) interessant. CandidateHandle gibt an, für welche gefundenen Barcodes die Typen zurückgegeben werden sollen. Beispielsweise wird CandidateHandle gleich 0 gesetzt, wenn nur der Typ des ersten Ergebnisses verlangt wird, oder CandidateHandle wird auf *'all'* gesetzt, wenn die Typen für alle Ergebnisse zurückgegeben werden sollen.

'decoded_data': Die dekodierte Zeichenfolge des Symbols wird als Tupel von Bytewerten zurückgegeben. Im Gegensatz zu *'decoded_reference'* handelt es sich hier um dieselben aufbereiteten Daten wie sie auch von *'decoded_strings'* geliefert werden. Falls die erwarteten Daten binäre, nicht druckbare Zeichen enthalten, ist es sinnvoll diesen Parameter zu nutzen. Insbesondere können Code 128 oder Code 93 auch NULL-Zeichen kodieren. In diesem Fall können die vollständigen Daten nur über diesen Parameter abgefragt werden, da das NULL-Zeichen ansonsten die Zeichenkette terminiert.

'decoded_reference': Die zu Grunde liegenden Referenz-Zeichen werden ausgegeben. Diese enthalten alle ursprünglichen Zeichen des Barcodes, einschließlich möglicher Start- oder Stopzeichen sowie Prüfzeichen. Das bestimmte Ergebnis hängt vom Typ des Barcodes ab:

- Für Barcodes, welche nur numerische Werte kodieren, wie z.B. die EAN/UPC Codes, die GS1 DataBar Codes (außer der GS1 DataBar Expanded Varianten), sowie die 2/5 Codes, stehen in den Referenz-Zeichen identische Werte wie in der dekodierten Zeichenfolge.
- Für alphanumerische Codes, wie z.B. Code 39 oder Code 128, sind die Referenz-Zeichen die Indizes in der entsprechenden Dekodiertabelle.
- Für GS1-128 Codes sind die Referenz-Zeichen die Indizes in der entsprechenden Dekodiertabelle. Dabei ist das Sonderzeichen FNC1 als 102 dekodiert.
- Für GS1 DataBar Expanded und GS1 DataBar Expanded Stacked Codes stellen die Referenz-Zeichen die ASCII-Werte der dekodierten Daten dar (siehe auch Tupel / String-Operationen). Dabei ist das Sonderzeichen FNC1 als 29 dekodiert.
- Für alle Codes der GS1-DataBar-Familie repräsentiert das erste Referenz-Zeichen das Linkage-Flag. Dieses hat einen Wert von 1, falls das Flag gesetzt ist, bzw. 0 falls es nicht gesetzt ist.

Da die Referenz-Zeichen ein Tupel sind, können diese nur für ein einzelnes Ergebnis abgerufen werden.

'composite_strings': Die dekodierte Zeichenfolge eines am GS1 DataBar Barcode angehängten Composite-Symbols werden ausgegeben. Für weitere Erklärungen siehe die Beschreibung des *'composite_code'* Parameters von set_bar_code_param.

'composite_reference': Die dekodierte Zeichenfolge des GS1 Composite-Symbols wird als Tupel von Bytewerten zurückgegeben. Falls die erwarteten Daten binäre, nicht druckbare Zeichen enthalten, ist es sinnvoll diesen Parameter an Stelle von *'composite_strings'* zu nutzen.

'orientation': Die Orientierung des gelesenen Symbols wird ausgegeben. Die Orientierung eines Barcodes ist der Winkel zwischen der Leserichtung des Barcodes und der horizontalen Bildachse. Der Winkel ist positiv bei Drehung nach links und hat die Einheit Grad. Er kann im Wertebereich von [*-180.0 ... 180.0*] Grad liegen. Die Leserichtung ist senkrecht zu den einzelnen Barcodestrichen. Falls die Orientierung der Barcodestriche nicht eindeutig bestimmt werden kann, wie z.B. für verzerrte Codes, wird die Orientierung der Scanlinien zurückgegeben. Ist ResultName auf *'orientation'* gesetzt, so werden die Orientierungen der Ergebnissen ausgegeben. Es wird ein einzelner Winkelwert ausgegeben, wenn nur ein Ergebnis abgefragt wird, z.B. wenn CandidateHandle auf *0* gesetzt ist. Sonst, wenn CandidateHandle auf *'all'* gesetzt ist, wird ein Tupel mit Winkelwerten für alle Ergebnisse ausgegeben.

'element_size': Die Elementgröße eines gelesenen Symbols wird ausgegeben.

'aborted': Es wird ausgegeben, ob find_bar_code im vorangegangenen Durchlauf vorzeitig abgebrochen wurde oder nicht:

0 - *abgeschlossen* find_bar_code wurde vollständig abgeschlossen.

1 - *abgebrochen* find_bar_code wurde durch ein Timeout abgebrochen, siehe set_bar_code_param.

2 - *abgebrochen* find_bar_code wurde durch set_bar_code_param mit *'abort'* abgebrochen.

'quality_isoiec15416': Ein Tupel mit der Bewertung der Druckqualität gemäß dem internationalen Standard ISO/IEC 15416:2016 wird zurückgegeben. Es ist zu beachten, dass diese Bewertung nur erfolgt, wenn der Barcode mit `find_bar_code` im *'persistence'*-Modus gelesen wurde (siehe `set_bar_code_param`).

Das erste Element des Bewertungs-Tupels enthält immer die Gesamtqualität des Symbols. Die Länge des Tupels und die Bedeutung der restlichen Elemente hängen davon ab, ob ein *einfaches 1D* Barcodesymbol (z.B. EAN-13, Code 128, GS1 DataBar Omnidirectional usw.) oder ein *zusammengesetztes (composite)* Barcodesymbol (z.B. GS1 DataBar Omnidirectional CC-A, GS1 DataBar Limited CC-B usw.) bewertet wird. Die Namen der einzelnen Grade können abgefragt werden, indem `ResultName` auf *'quality_isoiec15416_labels'* gesetzt wird, siehe weiter unten.

Für die Bewertung der Druckqualität werden *Scanlinien* (scan reflectance profiles) durch das Symbol gelegt. Bei jeder dieser Scanlinien wird für jeden der evaluierten Parameter ein (Qualitäts)-Grad zugewiesen.

Aus Kompatibilitätsgründen werden die Grade mit einem Wert von 0 bis 4 bewertet, wobei 0 für den schlechtesten und 4 für den besten Grad steht. Mit *'quality_isoiec15416_float_grades'* werden die einzelnen Grade dem Standard gemäß mit einer Genauigkeit von einer Dezimalstelle zurückgegeben. Der niedrigste Grad von allen zuvor bestimmten Parametergraden einer Scanlinie ist als *Scanlinien-Gütegrad (scan reflectance profile grade)* definiert. Somit besitzt am Ende jede Scanlinie einen Grad, der den minimalen Grad aller Parametergrade der jeweiligen Scanlinie repräsentiert. Der arithmetische Mittelwert dieser Grade wird schließlich als die *Gesamtqualität des Symbols (overall symbol grade)* zurückgegeben.

Es ist zu beachten, dass, obwohl sich diese Implementierung streng an den Standard hält, die Bewertung der Grade von der Barcode-Dekodierungsprozedur abhängt. Dadurch können die Bewertungsergebnisse leicht von den Ergebnissen anderer Barcodeleser (von anderen Anbietern) abweichen.

Das Tupel mit der Bewertung der Druckqualität hängt davon ab, ob es sich um einen einfachen 1D Barcode handelt oder einen zusammengesetzten Barcode:

Einfache 1D Barcodes:

Die Bewertung Druckqualität wird in einem Tupel mit den folgenden neun Elementen zusammengefasst:

0 - *Gesamtwert (overall quality)*: Das arithmetische Mittel der Scanlinien-Gütegrade (scan reflectance profile grades) aller Scanlinien. Man beachte, dass dieser Wert in den meisten Fällen nicht das Minimum der anderen Gradwerte ist, da er von den jeweiligen Scanlinien-Gütegraden abhängt.

1 - *Decode (decode)*: Auf *4* gesetzt, wenn das untersuchte Barcodesymbol entsprechend des Referenzdekodieralgorithmus für das jeweilige Symbol gelesen werden konnte. Andernfalls ist er *0*. Es ist zu beachten, dass HALCONs Dekodieralgorithmus von dem des Referenzdekodieralgorithmus abweicht. So kann es oft sein, dass HALCON ein Symbol lesen kann, obwohl der Wert für *Decode* entsprechend dem Standard *0* ist.

2 - *Symbolkontrast (symbol contrast)*: Die Differenz zwischen maximalem und minimalem Reflexionswert des Grauwertprofils. Ein stärkerer Kontrast ergibt einen besseren Grad.

3 - *Kleinster Reflexionswert (minimal reflectance)*: Auf *4* gesetzt, wenn der minimale Reflexionswert des Grauwertprofils kleiner oder gleich 0.5 des maximalen Reflexionswertes ist, andernfalls auf *0*.

4 - *Kleinster Kantenkontrast (minimale edge contrast)*: Bewertet den minimalen Kontrast zwischen zwei benachbarten Symbolelementen (helles zu dunkles oder dunkles zu helles Element) im Grauwertprofil.

5 - *Modulation (modulation)*: Bewertet die Amplitude zwischen den Symbolelementen. Höhere Amplituden machen die Zuordnung der Elemente zu Strich oder Lücke zuverlässiger und resultiert in besserem Grad.

6 - *Defekte (defects)*: Reflektiert Unregelmäßigkeiten im Grauwertprofil innerhalb der Symbolelemente oder Ruhezonen.

7 - *Dekodierbarkeit (decodability)*: Bezeichnet Abweichungen der Symbolelementbreiten von ihrem Nominalwert, der im entsprechenden Symbologiestandard festgelegt ist.

8 - *Zusätzlichen Anforderungen (additional requirements)*: Ein Grad bezüglich weiterer symbologiespezifischen Anforderungen. Dies betrifft meist die Ruhezonen-Breiten (quiet zone widths), kann aber auch anderes betreffen, wie z.B. das Breit-Schmal-Verhältnis (wide/narrow ratio), die Zwischenzeichen-Lücken (inter character gaps), die umschließenden Muster (guarding patterns). Für die folgenden Codetypen kann es zu Abweichungen zu den Spezifikationen im dazugehörigen ISO/IEC Standard kommen. Dies liegt in der Regel daran, dass zur Berechnung der *Zusätzlichen Anforderungen (additional requirements)* metrische Informationen benötigt werden, welche dem Operator nicht zugänglich sind.

Code-39: der Grad für die *Zwischenzeichen-Lücken (inter character gaps)* hängt von der, im metrischen Maß gegebenen, *schmalsten Elementbreite (narrow element width)* ab. Da in

find_bar_code weder metrische Information eingehen noch berechnet werden können, gibt der Operator get_bar_code_result nur den Grad zurück in der Annahme die schmalste Elementbreite sei kleiner als 0,287 mm.

EAN-13, EAN-8, UPC-A, UPC-E (inklusive Add-On-Varianten): ISO/IEC 15420:2009 spezifiziert als zusätzliches Kriterium den Vergrößerungsfaktor (magnification factors). Die Berechnung dieses Kriteriums ist nicht möglich, da Wissen über die metrische Größe des Barcodes vorhanden sein müsste.

Während *Gesamtwert* den endgültigen Symbolgrad angibt, weisen die restlichen Grade auf mögliche qualitätsverringernde Ursachen hin. Eine Liste mit häufig vorkommenden Defekten und deren Effekt auf die einzelnen Qualitätsgrade kann man im Standard ISO/IEC 15416:2016 finden.

Zusammengesetzten Barcodes (composite bar codes):

Die Druckqualitätsbewertung eines zusammengesetzten Barcodes wird in einem Tupel gegeben, welches 24 Grade umfasst. Zur besseren Übersicht sind die Grade in drei Gruppen eingeteilt: GESAMTWERT (OVERALL), LINEAR (LINEAR) und ZUSAMMENGESETZT (COMPOSITE), wobei letztere eine Untergruppe enthält.
Die Gesamtgrade sind in der Gruppe GESAMTWERT (OVERALL) aufgeführt. Die Grade in den Gruppen LINEAR (LINEAR) und ZUSAMMENGESETZT (COMPOSITE) geben Auskunft über mögliche Ursachen für schlechte Qualitätsbewertungen eines Symbols.
Im folgenden werden gruppenweise die einzelnen Tupelelemente mit jeweiligem Index sowie der Definition aufgelistet.

- Die Gruppe GESAMTWERT (OVERALL):
  Diese Grade geben das Minimum aller Grade der entsprechenden Gruppe.

  0 - *Gesamtwert (overall quality)*: Der endgültige Symbolgrad. Er gibt den schlechtesten aller Grade zurück.

  1 - *Gesamtwert Linear (overall linear)*: Der Gesamtgrad der Gruppe LINEAR, den linearen 1D Symbolen. Dieser Grad gibt den schlechtesten Grad der Gruppe zurück.

  2 - *Gesamtwert Zusammengesetzt (overall composite)*: Der Gesamtgrad der Gruppe ZUSAMMENGESETZT, den zusammengesetzten 2D Symbolen. Dieser Grad gibt den schlechtesten Grad der Gruppe zurück.

- Die Gruppe LINEAR (LINEAR):
  Die Interpretation der Grade dieser Gruppe entspricht jenen für einfache 1D Barcodes weiter oben.

  3 - *Decode (decode)*

  4 - *Symbolkontrast (symbol contrast)*

  5 - *Kleinster Reflexionswert (minimal reflectance)*

  6 - *Kleinster Kantenkontrast (minimal edge contrast)*

  7 - *Modulation (modulation)*

  8 - *Defekte (defects)*

  9 - *Dekodierbarkeit (decodability)*

  10 - *Zusätzliche Anforderungen (additional requirements)*

- Die Gruppe ZUSAMMENGESETZT (COMPOSITE): Diese Grade entsprechen den Qualitätsgraden des PDF 417 Datacodes, den Qualitätsgraden zweidimensionaler Datacodes nach ISO/IEC 15415: 2011 (siehe *'quality_isoiec15415'* in get_data_code_2d_results). Dabei besteht jedoch folgender Unterschied: Für PDF 417 Datacodes wird ein für die Bewertung der Reflexionseigenschaften ein Start-/Stoppmusters verwendet. Für zusammengesetzte Barcodes werden statt des Start-/Stoppmusters so genannte Raw Adress Patterns (RAP) verwendet. Die Grade dieser RAP Elemente sind in der Untergruppe ZUSAMMENGESETZTE RAP (COMPOSITE RAP) aufgeführt und stimmen mit den Qualitätsgraden für einfache 1D Barcodes überein.

  11 - *Decode (decode)*: Grad zur Dekodierbarkeit des zusammengesetzten Symbols. Seine Bedeutung und Festlegung entspricht demjenigen für einfache 1D Barcodes, der oben beschrieben ist.

  12 - *Gesamt-RAP (RAP overall)*: Minimum der Qualitätsgrade in der Untergruppe ZUSAMMENGESETZTE RAP Elemente (COMPOSITE RAP elements).

  13 - *RAP Kontrast (RAP contrast)*: Grad der Untergruppe ZUSAMMENGESETZTE RAP Elemente (COMPOSITE RAP elements), entspricht jenem für einfache 1D Barcodes.

14 - *RAP Kleinster Reflexionswert (RAP minimal reflectance)*: Grad der Untergruppe `ZUSAMMENGESETZTE RAP Elemente (COMPOSITE RAP elements)`, entspricht jenem für einfache 1D Barcodes.

15 - *RAP Kleinster Kantenkontrast (RAP minimal edge contrast)*: Grad der Untergruppe `ZUSAMMENGESETZTE RAP Elemente (COMPOSITE RAP elements)`, entspricht jenem für einfache 1D Barcodes.

16 - *RAP Modulation (RAP modulation)*: Grad der Untergruppe `ZUSAMMENGESETZTE RAP Elemente (COMPOSITE RAP elements)`, entspricht jenem für einfache 1D Barcodes.

17 - *RAP Defekte (RAP defects)*: Grad der Untergruppe `ZUSAMMENGESETZTE RAP Elemente (COMPOSITE RAP elements)`, entspricht jenem für einfache 1D Barcodes.

18 - *RAP Dekodierbarkeit (RAP decodability)*: Grad der Untergruppe `ZUSAMMENGESETZTE RAP Elemente (COMPOSITE RAP elements)`, entspricht jenem für einfache 1D Barcodes.

19 - *Codewort-Ergebnis (codeword yield)*: zählt und berechnet den relativen Anteil der richtig dekodierten Codewörter, die unter den Scans liegen.

20 - *Nicht verwendete Fehlerkorrektur (unused error correction)*: zählt und berechnet den relativen Anteil der falsch dekodierten Codewörter, die innerhalb der Fehlerkorrekturkapazität liegen.

21: *Modulation (modulation)*: bewertet die Gleichmäßigkeit der Reflexion der dunklen und hellen Module.

22 - *Dekodierbarkeit (decodability)*: misst die Abweichung der Strich- und Lückenlängen von ihrem im Symbologiestandard vorgegebenen Werten.

23 - *Defekte (defects)*: bezeichnet Unregelmäßigkeiten im Scanprofil innerhalb einzelner Striche und Lücke.

'quality_isoiec15416_float_grades': Ein Tupel mit der gleichen Bewertung der Druckqualität wie *'quality_isoiec15416'*. Gemäß dem internationalen Standard ISO/IEC 15416:2016 werden die Qualitätsgrade mit einer Dezimalstelle Genauigkeit zurückgegeben. Die Namen der einzelnen Grade können abgefragt werden, indem `ResultName` auf *'quality_isoiec15416_labels'* gesetzt wird, siehe weiter unten.

Es ist zu beachten, dass diese Bewertung nur erfolgt, wenn der Barcode mit `find_bar_code` im *'persistence'*-Modus gelesen wurde (siehe `set_bar_code_param`).

'quality_isoiec15416_values': Ein Tupel aus Rohmesswerten für alle Grade, die *'direkt messbar'* sind. Das sind Grade, deren Definition im Standard ISO/IEC 15416:2016 einer *'direkten Ableitung'* der Reflexionseigenschaften des abgebildeten Symbols entsprechen, oder Grade, welche das Ergebnis eines *'direkten Zählens'* sind. Es ist zu beachten, dass diese Bewertung nur erfolgt, wenn der Barcode mit `find_bar_code` im *'persistence'*-Modus gelesen wurde (siehe `set_bar_code_param`).

Die Namen der einzelnen Tupel-Elemente können abgefragt werden, indem `ResultName` auf *'quality_isoiec15416_labels'* gesetzt wird, siehe weiter unten.

Alle Werte (nicht Grade) sind zwischen *0.0* und *1.0* normiert. Somit entspricht z.B. ein Wert von *0.75* für den Grad *Symbolkontrast* einem Grauwert von *191.25* (für byte Bilder).

Aus welchen Werten sich das Tupel zusammensetzt hängt davon ab, ob es sich um einen einfachen 1D Barcode handelt oder einen zusammengesetzten Barcode:

Einfache 1D Barcodes:

Die folgenden Werte werden unter dem entsprechenden Index zurückgegeben:

0 - *Gesamtwert (overall quality)*

2 - *Symbolkontrast (symbol contrast)*

3 - *Kleinster Reflexionswert (minimal reflectance)*

4 - *Kleinster Kantenkontrast (minimale edge contrast)*

5 - *Modulation (modulation)*

6 - *Defekte (defects)*

7 - *Dekodierbarkeit (decodability)*

Zusammengesetzten Barcodes (composite bar codes):

Die folgenden Werte werden unter dem entsprechenden Index zurückgegeben:

0 - *Gesamtwert (overall quality)*

1 - *Gesamtwert Linear (overall linear)*

2 - *Gesamtwert Zusammengesetzt (overall composite)*
4 - *Symbolkontrast (symbol contrast)*
5 - *Kleinster Reflexionswert (minimal reflectance)*
6 - *Kleinster Kantenkontrast (minimal edge contrast)*
7 - *Modulation (modulation)*
8 - *Defekte (defects)*
9 - *Dekodierbarkeit (decodability)*
13 - *RAP Kontrast (RAP contrast)*
14 - *RAP Kleinster Reflexionswert (RAP minimal reflectance)*
15 - *RAP Kleinster Kantenkontrast (RAP minimal edge contrast)*
16 - *RAP Modulation (RAP modulation)*
17 - *RAP Defekte (RAP defects)*
18 - *RAP Dekodierbarkeit (RAP decodability)*
19 - *Codewort-Ergebnis (codeword yield)*
20 - *Nicht verwendete Fehlerkorrektur) (unused error correction)*

Diese Tupel haben die selbe Länge wie die Grade, die zurückgegeben werden, setzt man `ResultName` auf *'quality_isoiec15416'*, respektive *'quality_isoiec15416_float_grades'*. Für alle nicht aufgelisteten Elemente liefert der Operator den Wert *'N/A'* zurück. Bei einfachen 1D Barcodes haben die Einträge *Decode (decode)* und *Zusätzliche Anforderungen (additional requirements)* keine entsprechenden darunterliegenden Rohmesswerte und der Operator liefert dementsprechend *'N/A'* zurück. Bei zusammengesetzten Barcodes gilt dies für die Grade *Decode (decode)* (`LINEAR (LINEAR)`), *Zusätzliche Anforderungen (additional requirements)* (`LINEAR (LINEAR)`), *Decode (decode)* (`ZUSAMMENGESETZT (COMPOSITE)`) und *Gesamt-RAP (RAP overall)* (`ZUSAMMENGESETZT (COMPOSITE)`). Darüber hinaus ist zu beachten, dass, obwohl die Grade *Modulation (modulation)* (`ZUSAMMENGESETZT (COMPOSITE)`), *Dekodierbarkeit (decodability)* (`ZUSAMMENGESETZT (COMPOSITE)`) und *Defekte (defects)* (`ZUSAMMENGESETZT (COMPOSITE)`) auch das Grauwert-Profil des zusammengesetzten Symbols bewerten, auch für sie *'N/A'* zurückliefert wird. Das liegt daran, dass diese Grade durch ein komplexes Schema, welches die Dekodierungsroutine und den Fehlerkorrekturmechanismus der Symbologie einbezieht, berechnet werden. Demzufolge haben diese Grade keine direkten darunterliegenden Rohmesswerte.

'quality_isoiec15416_labels': Gibt die Namen für die Elemente eines Bewertungstupels nach ISO/IEC 15416: 2016 zurück. Das Bewertungstupel wird von `get_bar_code_result` z.B. mit dem Parameter *'quality_isoiec15416'* ausgegeben.

Es ist zu beachten, dass der Operator den `CandidateHandle` eines erfolgreich gefundenen Ergebnisses braucht, um den *einfachen 1D Barcode* von dem *composite Barcode* Fall unterscheiden zu können.

'status': Berechnet zusätzliche Informationen über Scanlinien einer Kandidatenregion und gibt diese im Klartext zurück. `CandidateHandle` kann hier Werte von 0 bis (m-1) annehmen, wobei m die Gesamtzahl an Kandidaten ist. Der Status kann nur berechnet werden, wenn zuvor `find_bar_code` oder `decode_bar_code_rectangle2` aufgerufen wurde, während der Barcode-Leser im *'persistence'*-Modus war (siehe `set_bar_code_param`). Zwischen dem Aufruf von `find_bar_code` und dieser Statusabfrage dürfen keine Änderungen an Parametern des Barcode-Lesers durchgeführt werden.

Zur Berechnung der zusätzlichen Informationen wird die angegebene Kandidatenregion nochmals gescannt. Dabei beendet sich der Barcode-Leser erst nach Auswertung aller Scanlinien und nicht wie normalerweise nach erfolgreicher Dekodierung. Dies ist rechenintensiv und sollte nur abgefragt werden, falls diese zusätzlichen Informationen benötigt werden. Für weitere Informationen zum Setzen von Scanparametern, welche die Scanlinien betreffen, siehe `set_bar_code_param`.

Der Operator gibt für jede Scanlinie der Kandidatenregion eine Statusmeldung und möglicherweise eine Warnung zurück, welche an dem String der Statusmeldung hinzugefügt wird. Diese Meldungen sind in der gleichen Reihenfolge wie die Scanlinien selbst vom Operator `get_bar_code_object` mit dem Parameter *'scanlines_all'* zurückgegeben werden. Die möglichen Meldungen dieser Kategorien werden unten gelistet.

- Statusmeldungen:
  Die folgende Liste zeigt die möglichen Statusmeldungen, gruppiert nach Arbeitsphasen in denen die Meldung auftreten kann. Die Zahlen sind die zugehörigen Statuscodes die mit *'status_id'* zurückgegeben werden (siehe unten).

  0: *'unknown status.'* Der Status der Scanlinie ist unbekannt. Die Scanlinie wird in diesem Fall ignoriert.

1 - *'success.'* Die Scanlinie konnte erfolgreich dekodiert werden.

2 - *'edges: not enough edges detected.'* Die Anzahl der Kanten in dieser Scanlinie ist zu gering für diesen Barcode-Typ.

3 - *'edges: not enough edges for a start, a stop and at least one data character.'* Die Anzahl der Kanten in dieser Scanlinie ist zu gering um mindestens das Start-, Stop- und ein Zeichenmuster zu finden.

4 - *'edges: too many edges detected.'* Die Anzahl der Kanten in dieser Scanlinie ist zu hoch für diesen Barcode-Typ.

5 - *'edges: center of scanline not within image domain.'* Das Zentrum der Scanlinie liegt nicht innerhalb der Bilddomäne.

6 - *'decoding: could not find stop character.'* Das Stop-Zeichen konnte nicht gefunden werden.

7 - *'decoding: could not find start and stop characters.'* Die Start- und Stop-Zeichen konnten nicht gefunden werden.

8 - *'decoding: internal error when estimating the maximum string length.'* Interner Fehler.

9 - *'decoding: internal error when decoding a single character.'* Interner Fehler.

10 - *'decoding: number of wide bars of a single character is not equal to 2.'* Für Barcode-Typen *'2/5 Industrial'* und *'2/5 Interleaved'* muss die Anzahl breiter Striche in einem Zeichen zwei sein.

11 - *'decoding: invalid encoding pattern.'* Das Kodiermuster entspricht keinem Zeichen in der symbolspezifischen Look-Up-Tabelle.

12 - *'decoding: invalid mix of character sets.'* Für die Barcode-Typen *'EAN-13'* und *'UPC-A'*: Die linke Hälfte des Symbols enthält eine ungültige Kombination der Sets A und B.

13 - *'decoding: error decoding the reference to a human readable string.'* Fehler bei der Dekodierung der Referenzdaten in einen lesbaren String. Dies kann beispielsweise auftreten, wenn (abhängig davon ob ein Prüfzeichen erwartet wird) nicht genügend Zeichen gefunden wurden.

14 - *'decoding: could not detect center guard pattern.'* Für die Barcode-Typen *'EAN-13'*, *'EAN-8'* und *'UPC-A'* (inklusive Add-on-Varianten) konnte das obligatorische Center Guard-Pattern nicht gefunden werden.

15 - *'decoding: could not detect left and/or right guard patterns.'* Für die Barcode-Typen *'EAN-13'*, *'EAN-8'* und *'UPC-A'* (inklusive Add-on-Varianten) konnte entweder das obligatorische linke oder rechte Normal Guard-Pattern nicht gefunden werden. Für die Barcode-Typen *'UPC-E'* (inklusive Add-on-Varianten) konnte entweder das linke Normal Guard-Pattern oder das rechte Special Guard-Pattern nicht gefunden werden.

16 - *'decoding: could not detect add-on guard pattern.'* Für die Barcode-Typen *'EAN-13'*, *'EAN-8'*, *'UPC-A'* und *'UPC-E'* mit Add-on-Symbolen konnte das obligatorische Add-On Guard-Pattern nicht gefunden werden.

17 - *'decoding: could not detect enough finder patterns.'* Für die Barcode-Typen der GS1-DataBar-Familie konnten nicht genug Finder-Pattern gefunden werden.

18 - *'decoding: no segment found.'* Für Stacked Barcode-Typen konnte kein Segment gefunden werden.

19 - *'check: checksum test failed.'* Die Überprüfung des Prüfzeichens ist fehlgeschlagen. Für Barcodes mit einem nicht-verpflichtenden Prüfzeichen kann die Überprüfung mit dem Parameter *'check_char'* im Operator `set_bar_code_param` ausgeschaltet werden.

20 - *'check: check of add-on symbol failed.'* Für die Barcode-Typen *'EAN-13'*, *'EAN-8'*, *'UPC-A'* und *'UPC-E'* mit Add-On-Symbolen entspricht die Kombination der Sets A und B nicht dem impliziten Prüfzeichen.

21 - *'check: detected EAN-13 bar code type instead of specified type.'* Für die Barcode-Typen *'UPC-A'* und dessen Add-on-Varianten muss die linke Hälfte des Symbols aus sechs Zeichen des Sets A bestehen, aber eine *'EAN-13'*-kompatible Kombination der Sets A und B wurde gefunden. Der Barcode sollte als *'EAN-13'* dekodiert werden.

22 - *'check: symbol region overlaps with another symbol region'* Der Barcode konnte dekodiert werden, aber die Symbolregion schneidet sich mit der Symbolregion eines bereits erfolgreich dekodierten Barcodes.

23 - *'decoding: could not detect the bar code type.'* Der Barcodeleser könnte das Barcodetyp nicht bestimmen. Dieser Zustand kann passieren nur wenn `find_bar_code` mit `CodeType`=*'auto'*) oder mit einer Liste von unterschiedlichen Barcodetypen ausgeführt wurde.

24 - *'decoding: quiet zone is disturbed.'* Die Überprüfung der Ruhezone ist fehlgeschlagen. Siehe den Abschnitt zu *'quiet_zone'* in `set_bar_code_param` für weitere Informationen.

25 - *'check: symbol region detected outside of the candidate region.'* Ein Barcode wurde detektiert aber ganz am Rand von oder sogar weit außerhalb der Kandidatregion. Das ist eine Indikation dafür, dass die Scanlinien zufällig Strichsequenzen gelesen haben, welche eher zu einem anderen und nicht zu dem aktuellen Kandidat gehören.

26 - *'decoding: minimum code length not reached (see min_code_length).'* Die Anzahl dekodierter Zeichen im gefundenen Barcode ist kleiner als die erforderliche Anzahl. Siehe den Abschnitt zu *'min_code_length'* in `set_bar_code_param` für weitere Informationen zur minimalen Anzahl dekodierten Zeichen.

- Warnungen:
  Die unten gelisteten Meldungen sind nur Warnungen, keine Fehler. Sie können in Kombination mit einer Statusmeldung auftreten und werden dann an die Statusmeldung angehängt. Diese Meldungen werden zurückgegeben, falls der Barcode-Leser mögliche Qualitätsprobleme im Eingabebild feststellt. Die Meldungen 1000 bis 1003 können nur bei erfolgreich dekodierten Scanlinien auftreten und nur für folgende Barcode-Typen: *'Codabar'*, *'2/5 Industrial'*, *'2/5 Interleaved'*, *'Code 39'*, *'Code 93'*, *'GS1-128'*, *'Code 128'*, *'MSI'*, alle *'EAN'*-, *'UPC-A'*- und *'UPC-E'*-Varianten. Zu beachten ist, dass diese Warnungen in Fällen in denen die Scanlinie bereits inkorrekt dekodiert wurde, aber vom Barcode-Leser nicht als ungültig erkannt wurde, falsch sind und ignoriert werden müssen.

  1000 - *'White spaces too wide.'* Die gemessene Breite der weißen Lücken ist größer als intern erwartet. Dies ist kein Dekodierfehler, sondern eine Warnung, dass es zu Dekodierfehlern kommen könnte, wenn die Breite der weißen Lücken noch mehr zunimmt.

  1001 - *'White spaces too narrow.'* Die gemessene Breite der weißen Lücken ist kleiner als intern erwartet. Dies ist kein Dekodierfehler, sondern eine Warnung, dass es zu Dekodierfehlern kommen könnte, wenn die Breite der weißen Lücken noch mehr abnimmt.

  1002 - *'Bars too wide.'* Die gemessene Breite der Striche ist größer als intern erwartet. Dies ist kein Dekodierfehler, sondern eine Warnung, dass es zu Dekodierfehlern kommen könnte, wenn die Breite der Striche noch mehr zunimmt.

  1003 - *'Bars too narrow.'* Die gemessene Breite der Striche ist kleiner als intern erwartet. Dies ist kein Dekodierfehler, sondern eine Warnung, dass es zu Dekodierfehlern kommen könnte, wenn die Breite der Striche noch mehr abnimmt.

  1004 - *'Possible saturation of gray values.'* Ein interner Algorithmus zeigt, dass die weißen Pixel in der Symbolregion in Sättigung sein könnten. Diese Warnung ist nicht spezifisch für einen Barcode-Typ. Diese Warnung wird für jede Scanlinie ausgegeben. Beispielsweise ist die Kombination der Statusmeldungen 1001/1004 oder 1003/1004 ein Hinweis darauf, dass die Eingabebilder überbelichtet sein könnten.

  1005 - *'No composite component found.'* Das Linkage-Flag in der linearen Barcode-Komponente zeigt an, dass eine Composite-Komponente existiert, aber die Composite-Komponente konnte nicht gefunden werden. Der Dekodierer gab nur die lineare Komponente zurück. Diese Warnung wird für jede Scanlinie in der entsprechenden Kandidatenregion ausgegeben.

  1006 - *'Used for merging.'* Diese Information zeigt an, dass die Kanten der aktuellen Scanlinie benutzt wurden um die vereinigte Scanline zu berechnen.

'status_id': Berechnet zusätzliche Informationen über Scanlinien und gibt diese in einem numerischen Format zurück, das leicht weiterverarbeitet werden kann. Die Beschreibung dieser Funktionalität und die Zahlen der Nachrichten werden bei dem Parameter *'status'* (s.o.) beschrieben. Die Zahlen der Warnungen können zusammen mit anderen Status-Zahlen zurückgegeben werden. Die einzelnen Zahlen der Statusmeldungen und Warnungen werden als String, getrennt durch ein Semikolon, zurückgegeben, beispielsweise als `'1;1000;1004'`.

'status_small_elements_robustness': bestimmt zusätzliche Informationen über einen optionalen Dekodierversuch, der im Abschnitt über *'small_elements_robustness'* des Operators `set_bar_code_param` beschrieben wird. Zusätzlich zu den Statusmeldungen, die im Abschnitt *'status'* beschrieben sind, wird hier die Meldung *'small_elements_robustness: no scan.'* zurückgegeben falls der Algorithmus nicht ausgeführt wurde oder fehlschlug.

Parameter

▷ **`BarCodeHandle`** (input_control) ..... barcode ⇝ *handle*
Handle des zu verwendenden Barcode-Modells.

▷ **CandidateHandle** (input_control) . . . . . integer ⇝ *string* / integer
Auswahl der Barcode-Ergebnisse oder Kandidaten, für die eine Information abgefragt werden soll.
Defaultwert: 'all'
Wertevorschläge: `CandidateHandle` ∈ {0, 1, 2, 'all'}

▷ **ResultName** (input_control) . . . . . attribute.name ⇝ *string*
Namen der Ergebnisse, die vom Barcode-Leser abgefragt werden sollen.
Defaultwert: 'decoded_types'
Wertevorschläge: `ResultName` ∈ {'decoded_types', 'decoded_strings', 'decoded_data', 'decoded_reference', 'element_size', 'orientation', 'composite_strings', 'composite_reference', 'aborted', 'quality_isoiec15416', 'quality_isoiec15416_labels', 'quality_isoiec15416_values', 'quality_isoiec15416_float_grades', 'status', 'status_id', 'status_small_elements_robustness'}

▷ **BarCodeResults** (output_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Liste mit den Ergebnissen.

Ergebnis

`get_bar_code_result` liefert den Wert TRUE, falls die übergebenen Parameter korrekt und die angeforderten Ergebnisse für die letzte Symbolsuche verfügbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_bar_code`

Siehe auch

`get_bar_code_object`

Modul

Bar Code

query_bar_code_params (: : BarCodeHandle,
Properties : GenParamName)

Liefert für das übergebene Barcode-Modell die Namen der Parameter, die in den set_bar_code- und get_bar_code*-Operatoren benutzt werden können.*

Mit Hilfe des Operators `query_bar_code_params` lassen sich die Namen der Parameter des Barcode-Modells finden, die die Eigenschaften erfüllen, die in `Properties` angegeben sind. Um das zu erklären, hier einige Hintergrundinformationen. Nach Erstellen eines Barcode-Modells sind zuerst alle Parameter *'general'*, d.h. sie haben die selben Werte für alle unterstützten Barcode-Typen gesetzt. Während des automatischen Parametertrainings oder nach Aufruf des Operators `set_bar_code_param_specific` können einige Parameter zur Laufzeit Barcodetyp-spezifisch werden. Diese Parameter können nicht mit dem allgemeinen Operator `get_bar_code_param` ausgelesen werden, sondern nur mit `get_bar_code_param_specific`. Um dieses Auslesen einfach und generisch zu machen, kann die Ausgabe `GenParamName` von `query_bar_code_params` als Eingabeparameter `GenParamName` in `get_bar_code_param` oder `get_bar_code_param_specific` verwendet werden, abhängig davon, ob `Properties`='*general*' oder `Properties`='*specific*'.

Bitte beachten Sie, dass Sie alternativ ein statisches Tupel von Parameternamen in `get_bar_code_param` oder `get_bar_code_param_specific` verwenden könnten, z.B. die in der Beschreibung angegebenen. Dieses Vorgehen ist jedoch unflexibel und wird nicht empfohlen.

Mögliche Werte für `Properties` sind:

'all': Alle vom Barcode-Leser unterstützten Parameternamen werden zurückgegeben. Die Namen sind unabhängig vom, durch `BarCodeHandle` angegebenen, Barcode-Modell.

'general': Rückgabe der Namen derjenigen Parameter, die für alle unterstützten Barcode-Typen gleiche Werte gesetzt haben. Nur diese Parameter können mit `get_bar_code_param` ausgelesen werden.

Der Parametername 'train' wird ausgeschlossen, weil dieser nicht zusammen mit anderen möglichen Parameternamen an get_bar_code_param übergeben werden kann, da durch 'train' als Parameter für get_bar_code_param mehrwertige Tupel zurückgegeben werden können.

'specific': Rückgabe der Namen derjenigen Parameter, die spezifisch für bestimmte Barcode-Typen gesetzt sind. Die Werte dieser Parameter müssen über get_bar_code_param_specific statt get_bar_code_param ausgelesen werden. Barcodetyp-spezifische Werte können in Zusammenhang mit der Barcode-Autodiskriminierung auftreten die bei dem Operator find_bar_code erklärt wird oder beim automatischen Parametertraining.

'trained_general': Rückgabe der Namen derjenigen Parameter die bereits trainiert wurden und die für alle unterstützten Barcode-Typen gleiche Werte gesetzt haben. Der Trainingsmodus ist bei dem Operator set_bar_code_param beschrieben.

'trained_specific': Rückgabe der Namen derjenigen Parameter die bereits trainiert wurden und die spezifisch für bestimmte Barcode-Typen gesetzt sind. Der Trainingsmodus ist bei dem Operator set_bar_code_param beschrieben.

Parameter

▷ **BarCodeHandle** (input_control) ..... barcode ⇝ *handle*
Handle des zu verwendenden Barcode-Modells.

▷ **Properties** (input_control) ..... attribute.name ⇝ *string*
Eigenschaften der Parameter.
Defaultwert: 'trained_general'
Werteliste: Properties ∈ {'general', 'specific', 'trained_general', 'trained_specific', 'all'}

▷ **GenParamName** (output_control) ..... attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.

Beispiel

```
TrainParams := ['element_size_min', 'element_size_max', 'orientation']
Train3times := gen_tuple_const(|TrainParams|,'train')
create_bar_code_model (Train3times, TrainParams, BarCodeHandle)
find_bar_code (Image, SymbolRegions, BarCodeHandle, 'EAN-13', \
               DecodedDataStrings)
query_bar_code_params (BarCodeHandle, 'trained_general', NamesGen)
* returns NamesGen = ['element_size_max', 'element_size_min']
get_bar_code_param (BarCodeHandle, NamesGen, ValGen)
* returns e.g. ValGen = [4.0, 1.5]
query_bar_code_params (BarCodeHandle, 'trained_specific', NamesSpec)
* returns NamesSpec = ['orientation', 'orientation_tol']
get_bar_code_param_specific (BarCodeHandle, 'EAN-13', NamesSpec, ValSpec)
* returns e.g. ValSpec = [89.9127, 0.5]
```

Ergebnis

query_bar_code_params liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

find_bar_code, decode_bar_code_rectangle2

Nachfolger

get_bar_code_param, get_bar_code_param_specific

Modul

Bar Code

read_bar_code_model (: : FileName : BarCodeHandle)

Liest ein Barcode-Modell aus einer Datei und legt ein neues Modell an.

Der Operator read_bar_code_model liest aus der Datei FileName ein Barcode-Modell ein und erzeugt aus den eingelesenen Daten ein neues Modell. In BarCodeHandle wird ein Handle auf das neue Modell zurückgeliefert. Die Datei FileName wird mit dem Operator write_bar_code_model erzeugt. Die Dateiendung für das Barcode-Modell ist in HALCON 'bcm'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der einzulesenden Barcode-Datei.
Defaultwert: 'bar_code_model.bcm'
Dateiendung: .bcm

▷ **BarCodeHandle** (output_control) . . . . . barcode ⇝ *handle*
Handle für das neu eingelesene Barcode-Modell.

Ergebnis

read_bar_code_model liefert den Wert TRUE, falls eine Barcode-Datei mit dem angegebenen Namen gefunden wurde und korrekt eingelesen werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

find_bar_code

Alternativen

create_bar_code_model

Siehe auch

write_bar_code_model, clear_bar_code_model

Modul

Bar Code

serialize_bar_code_model (: : BarCodeHandle : SerializedItemHandle)

Serialisieren eines Barcode-Modells.

serialize_bar_code_model serialisiert die Daten eines Barcode-Modells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_bar_code_model in eine Datei geschrieben werden. Das Barcode-Modell wird in dem Handle BarCodeHandle übergeben. Das serialisierte Modell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_bar_code_model wieder deserialisiert werden.

Parameter

▷ **BarCodeHandle** (input_control) . . . . . barcode ⇝ *handle*
Handle des zu serialisierenden Barcode-Modells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_bar_code_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_bar_code_model`

Siehe auch

`create_bar_code_model`, `set_bar_code_param`, `find_bar_code`

Modul

Bar Code

set_bar_code_param (: : BarCodeHandle, GenParamName, GenParamValue :)

Setzen ausgewählter Parameter des Barcode-Modells.

Der Operator `set_bar_code_param` erlaubt es, Parameter, welche Eigenschaften des Barcodes beschreiben oder das Verhalten des Modells bei der Symbolsuche steuern, zu verändern. Damit kann auf Besonderheiten des Barcodes oder dessen Abbildung durch den Bildsensor eingegangen werden. Alle Parameter können auch direkt beim Anlegen des Modells mit `create_bar_code_model` übergeben werden. Die aktuelle Konfiguration kann mit Hilfe von `get_bar_code_param` ausgelesen werden.

Es folgt eine Übersicht über die verschiedenen generischen Parameter mit den jeweiligen Wertebereichen und Defaultwerten:

Größe der Barcode Elemente:

Die beiden ersten Parameter, *'element_size_min'* und *'element_size_max'*, beeinflussen die Suche nach Kandidatenregionen und die Dekodierung der Barcodes. Sie werden aber nicht strikt als Grenzen für die Größe der Elemente gefundener Barcodes verwendet. D.h. es können auch Barcodes gefunden werden, deren Elemente schmaler als *'element_size_min'* oder breiter als *'element_size_max'* sind.

Achtung: Beim Ändern der Größe über den Operator `set_bar_code_param`, werden die Parameterwerte *'element_size_min'* und *'element_size_max'* auf Konsistenz überprüft. Die beiden Parameter sollten im gleichen Aufruf gesetzt werden.

'element_size_min': Die minimale Größe eines Basiselements (auch als 'Modul' oder 'narrow bar' bezeichnet, je nach Barcodetyp), d.h. die minimale Dicke der schmalsten Striche und Zwischenräume des speziellen Barcodetyps. Der Wert dieses Parameters ist in Pixel definiert. Für Barcodes mit niedriger Auflösung sollte der Wert auf *1.5* reduziert werden, ggf. sogar auf bis zu *0.6*. Falls der Wert unter *2.0* gesetzt wird, wird ein zusätzlicher Dekodierversuch unternommen, der im Abschnitt über *'small_elements_robustness'* des Operators `set_bar_code_param` beschrieben wird. Für sehr große Barcodes kann der Wert entsprechend vergrößert werden, was kürzere Laufzeiten mit sich bringt.

Typischer Wertebereich: [*1.2*, ..., *10.0*]

Wertebereich: [*0.6*, ..., *64*]

Standardwert: 2.0

'element_size_max': Die maximale Größe eines Basiselements (auch als 'Modul' oder 'narrow bar' bezeichnet, je nach Barcodetyp), d.h. die maximale Dicke der schmalsten Striche und Zwischenräume des speziellen Barcodetyps. Der Wert dieses Parameters ist in Pixel definiert. Er sollte groß genug sein, damit für das komplette Symbol die Kandidatenregion gefunden wird. Anderseits darf er nicht so groß gewählt werden, dass zwei benachbarte Barcodes zu einem einzelnen Kandidaten verschmelzen.

Typischer Wertebereich: [*4.0*, ..., *60.0*]

Wertebereich: [*1*, ..., *256*]

Standardwert: 8.0

'element_size_variable': In manchen Barcodeaufnahmen variiert die Breite des kleinsten Elements über den Bereich des Barcodes. Diese Veränderungen entstehen durch perspektivische Projektion oder durch Deformation der Oberfläche, auf welcher der Barcode gedruckt ist (z.B. eine Tonnenverzerrung auf einer Flasche). Standardmäßig kann der Barcodeleser solche Verzerrungen nicht ausgleichen. Sollte die Einstellung *'element_size_variable'* auf *'true'* gesetzt sein, wird versucht, solche Deformationen lokal bei der Dekodierung auszugleichen. Dies kann dazu führen, dass Barcodes auch unter solchen Bedingungen gelesen werden können. Man beachte, dass nicht alle Deformationen ausgeglichen werden können.

Der Parameter *'element_size_variable'* ist nur für folgende Barcodetypen implementiert:

- GS1 DataBar Limited
- GS1 DataBar Expanded
- GS1 DataBar Expanded Stacked

Bei anderen Barcodetypen hat dieser Parameter keinen Effekt.

Werte: [*'false'*, *'true'*]

Standardwert: 'false'

'barcode_height_min': Die minimale Höhe des Barcodes. Der Wert dieses Parameters ist in Pixel definiert. Die Voreinstellung mit *-1* bedeutet, dass intern die Höhe des Barcodes automatisch entsprechend den anderen Einstellung gewählt wird. Bei sehr flachen Barcodes mit einer Höhe von weniger als 16 Pixeln ist es sinnvoll die Höhe manuell einzustellen, damit der Barcode gefunden und gelesen werden kann. Die minimale Höhe beträgt 8 Pixel. Bei sehr hohen Barcodes, z.B. mit 70 Pixeln und mehr, kann das manuelle Setzen auf die entsprechende Höhe zu einer Beschleunigung beim Lesen führen.

Typischer Wertebereich: [*-1*, *8*, . . . , *64*]

Standardwert: -1

'barcode_width_min': Die minimale Breite des Barcodes im Bild. Der Wert dieses Parameters ist in Pixel definiert. Die Breite eines Barcodes hängt von mehreren Faktoren ab:

- Auflösung der Kamera
- Entfernung zwischen Barcode und Kamera
- Barcode-Typ
- Anzahl kodierter Zeichen

Falls diese Eigenschaften in einer Anwendung nahezu unverändert bleiben, sollte dieser Parameter gesetzt werden, um die Geschwindigkeit und Robustheit des Barcode-Lesers zu erhöhen. Der Standardwert von *-1* bedeutet, dass der Barcode-Leser einen geeigneten Minimalwert anhand der gesuchten Barcode-Typen und des Parameters *'element_size_min'* schätzt.

Typischer Wertebereich: [*-1*, *40*, *50*, *60*, . . .]

Standardwert: -1

Scanparameter:

'num_scanlines': Die maximale Anzahl von Scanlinien, die während des Scannens eines (Kandidat-)Symbols benutzt werden. Wenn *'num_scanlines'* nicht gesetzt ist (der Parameter hat einen Wert von *0*), wird die Anzahl der Scanlinien nach einer internen Regel bestimmt: *10* für alle einzeiligen Barcodes, *20* für GS1 DataBar Stacked beziehungsweise GS1 DataBar Stacked Omnidirectional und *55* für GS1 DataBar Expanded Stacked. Durch diesen Parameter kann die Geschwindigkeit in zwei Fällen gesteigert werden. Im ersten Fall beinhaltet das Bild viele falsche Kandidaten. Während der Barcode selbst normalerweise nach ein oder zwei Scanvorgängen dekodiert werden kann (außer gestapelte Barcodes, siehe unten), wird ein falscher Kandidat mit dem Defaultwert von 10 Scanlinien gescannt, was die Laufzeit unnötig verlängert. Folglich kann die Geschwindigkeit durch eine reduzierte Anzahl der Scanlinien erhöht werden. Generell lässt sich sagen, dass Bilder höherer Qualität weniger Scanlinien benötigen als Bilder von niedrigerer Qualität. Für ein durchschnittliches Bild sollte ein Wert zwischen 2 und 5 ausreichend sein. Sollte ein Barcode jedoch nach Verringerung der Scanlinien nicht mehr gefunden werden, so muss die Anzahl der Scanlinien wieder erhöht werden. Der zweite Fall betrifft gestapelte Barcodes (derzeit GS1 DataBar Stacked, GS1 DataBar Stacked Omnidirectional und GS1 DataBar Expanded Stacked). Hier werden alle Scanlinien bewertet - im Gegensatz zu einzeiligen Barcodes (z.B. Code 128, EAN 13 oder GS1 DataBar Limited), wo das Scannen nach erfolgreicher Dekodierung des Codes beendet wird. Der Scanvorgang ist eine der zeitaufwändigsten Phasen des Operators `find_bar_code`. Deswegen kann die Anpassung des Parameters *'num_scanlines'* große Geschwindigkeitsvorteile bringen. Das gilt insbesondere für GS1 DataBar Expanded Stacked. Ein GS1 DataBar

Expanded Stacked Symbol kann in der Regel aus bis zu 11 Zeilen bestehen. Damit jede Zeile robust von jeweils 5 Scanlinien gelesen wird, sind im Operator 55 Scanlinien für den allgemeinen Fall vorgesehen. Wenn nur Symbole mit geringer Zeilenanzahl zu erwarten sind, sollte *'num_scanlines'* auf 1.5 bis 5 Scanlinien pro erwartete Zeile reduziert werden.

Falls dieser Parameter auf einen Wert gesetzt wird, der kleiner ist als die Einstellung für den Parameter *'min_identical_scanlines'*, so wird der Wert für *'min_identical_scanlines'* automatisch auf den Wert von *'num_scanlines'* reduziert.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden.

Typische Werte: [*0*, *5*, *10*, *20*, ...]

Standardwert: 0

'min_identical_scanlines': Mit diesem Parameter lässt sich die Wahrscheinlichkeit verringern, dass Barcodes falsch gelesen werden oder an Stellen gefunden werden, die keine Barcodes darstellen. Der Parameter legt die minimale Anzahl von Scanlinien fest, welche identische Ergebnisse liefern müssen, damit die Dekodierung eines Symbols akzeptiert wird.

Wenn *'min_identical_scanlines'* auf den Wert 0 gesetzt ist (welches der Standardwert für alle Barcodetypen außer '2/5 Industrial' und '2/5 Interleaved' ist), wird der Barcode dekodiert, sobald eine Scanlinie erfolgreich dekodiert wurde. Falls die Einstellung *'majority_voting'* aktiviert ist, wird ein anderes Verfahren angewandt (siehe Abschnitt über *'majority_voting'*). Analog hierzu reicht bei gestapelten Barcodes eine erfolgreich dekodierte Scanlinie pro Barcodezeile. Dieser Parameter kann auf 2 oder höher gesetzt werden, um zu verhindern, dass Barcodes unbeabsichtigt gefunden werden. Dies kann insbesondere dann hilfreich sein, wenn schlechte Bildqualität oder verrauschte Bildregionen dazu führen, dass eine Scanlinie falsche Strichkanten findet. Dieser Parameter kann auch die Anzahl falsch gefundener Barcodes in Bildern verringern, in welchen Symbole unterschiedlichen Barcodetyps gleichzeitig erscheinen.

Man beachte, dass der Standardwert für diesen Parameter für die meisten Barcodetypen *0* ist. Lediglich ist der Standardwert *2* für die Barcodetypen '2/5 Industrial' und '2/5 Interleaved'. Der Grund dafür ist, dass diese Barcodetypen leicht fälschlicher Weise in Text und anderen Störmustern erkannt werden, wenn der Parameter *'min_identical_scanlines'* auf *0* oder *1* gesetzt ist.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden.

Wenn vereinigte Scanlinien (siehe *'merge_scanlines'*)verwendet werden, drückt *'min_identical_scanlines'* aus, in wie vielen Scanlinien eine Kante erfolgreich erkannt werden muss (siehe `get_bar_code_object`, *'scanlines_merged_edges'*).

Typische Werte: [*0*, *2*, *3*, ...]

Standardwert:

Barcodetyp	min_identical_scanlines
2/5 Industrial	2
2/5 Interleaved	2
alle Anderen	0

'majority_voting': Dieser Parameter steuert das Verhalten bei der Auswahl unterschiedlicher Dekodierergebnisse pro Scanlinie. Falls der Parameter auf *'false'* gesetzt ist, wird ein erfolgreiches Dekodierergebnis zurückgegeben, sobald eine minimale Anzahl an identisch dekodierten Scanlinien gefunden wurde (siehe Abschnitt über *'min_identical_scanlines'* für weitere Details). Setzt man diesen Parameter auf *'true'*, so wird ein Mehrheitsvotum unter Einbeziehung aller Scanlinien durchgeführt. Das zurückgegebene Endergebnis wird dann jenes sein, welches von den meisten Scanlinien dekodiert wurde. Man beachte, dass die Einstellung *'true'* zu leicht erhöhten Laufzeiten führt, denn es müssen fast alle Scanlinien ausgewertet werden anstatt nur die minimal identischen Scanlinien.

Um die Robustheit gegenüber falschen Dekodierungen zu erhöhen, sollte dieser Parameter auf *'true'* gesetzt werden.

Dieser Parameter wird nur für Barcodetypen unterstützt, die nicht gestapelt sind.

Werte: [*'false'*, *'true'*]

Standardwert: 'false'

'stop_after_result_num': Anzahl erfolgreich dekodierter Barcodes nach der die Dekodierung angehalten wird. Falls dieser Parameter nicht gesetzt ist (der Parameter hat einen Wert von *0*), wird versucht alle Kandidaten zu dekodieren. Dieser Parameter wird typischerweise gesetzt, wenn die erwartete Anzahl von Barcodes im Voraus bekannt ist. Dann kann der Barcode-Leser die weitere Dekodierung von Kandidaten abbrechen und die Gesamtdekodierzeit wird reduziert.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden.

Typische Werte: [*0*, *1*, *2*, ...]

Standardwert: 0

'orientation': Erwarteter Barcode Orientierungswinkel. Ein potentieller (Kandidat-) Barcode hat Striche ähnlicher Orientierung. Man kann die Parameter *'orientation'* und *'orientation_tol'* anpassen, um den Wertebereich [*'orientation'*-*'orientation_tol'*, *'orientation'*+*'orientation_tol'*] zu definieren. `find_bar_code` bearbeitet nur Kandidatenregionen, deren Striche einen durchschnittlichen Orientierungswinkel haben, der in dem obigen Wertebereich liegt. Falls die Barcodes nur mit einer bestimmter Orientierung in den bearbeiteten Bildern erscheinen, so kann man den Wertebereich entsprechend reduzieren, sodass falsche Kandidaten früher erkannt werden und sich so die Ausführungszeit des Operator verkürzt. Diese Strategie bringt vor allem dann Vorteile, wenn die bearbeiteten Bilder viel Hintergrundtextur mit falsch orientierten Kandidaten enthalten.

Ausführlichere Informationen zu dem Orientierungswinkel können beim Operator `get_bar_code_result`(...,*'orientation'*,...) nachgelesen werden. Im Gegensatz zu `get_bar_code_result`, wird hier die eigentliche Leserichtung nicht berücksichtigt, und es sind deswegen nur Winkel in dem Wertebereich [*-90.0 ... 90.0*] von Interesse. Die einzige Ausnahme dazu sind Barcodes vom Typ PharmaCode, welche keine eindeutige Leserichtung haben und somit auch in der Regel zwei dekodierte Ergebnisse zurückliefern: mit einem Wertebereich von [*-180.0 ... 180.0*] kann eine spezielle Leserichtung und damit auch ein einzelnes Ergebnis gewählt werden.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden.

Typischer Wertebereich: [*-90.0*, ..., *90.0*]

Standardwert: 0.0

'orientation_tol': Toleranz der Orientierung. Siehe *'orientation'* für weitere Erklärungen. Wie bereits erklärt wurde, ist nur der Wertebereich [*-90.0 ... 90.0*] berücksichtigt, was mit einem *'orientation_tol'* von *90.0* ganz abgedeckt wird. Deswegen sind die Werte von *'orientation_tol'* auf den Wertebereich [*0.0 ... 90.0*] eingeschränkt. Dabei stellt ein Wert von *90.0* die Orientierungsbeschränkung grundsätzlich aus.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden.

Typischer Wertebereich: [*0.0*, ..., *90.0*]

Standardwert: 90.0

'quiet_zone': Steuert die Verifikation der Ruhezonen eines Barcodesymbols. Wenn diese eingeschaltet ist, werden Scanlinien verworfen, falls unerwartete Striche in den Ruhezonen gefunden werden. Wenn *'quiet_zone'='true'*, muss die Ruhezone mindestens so breit sein, wie in der entsprechenden Barcode-Norm festgesetzt ist. Diese Werte (in X-Einheiten, wobei X eine "Modulbreite"bezeichnet, d.h. die Breite des schmalsten Striches aus der Strichsequenz des Barcodes) können von der folgenden Tabelle entnommen werden:

Wenn *'quiet_zone'* auf eine Ganzzahl (>= *1*) gesetzt ist, dann muss eine Ruhezone von mindestens *'quiet_zone'*×X eingehalten werden.

Wenn *'quiet_zone'='tolerant'*, ist eine beschränkte Anzahl an Kanten in der Ruhezone erlaubt, aber höchstens eine pro vier Modulbreiten. Das Ziel ist zu verhindern, nur einen Teil von einem Barcode zu erkennen, aber dennoch Codes mit einer einfachen Verletzung der Ruhezone noch lesen zu können. Es ist zu beachten, dass auch für Codetypen mit einer vorgeschriebenen Ruhezone von weniger als vier Modulbreiten die Kanten innerhalb eines Bereichs von vier Modulbreiten um den Code überprüft werden. In diesem Fall kann dieses Kriterium also strenger sein als die Überprüfung mit *'quiet_zone'='true'*.

Wenn *'quiet_zone'='false'*, ist die Verifikation ausgeschaltet.

Die Verifikation der Ruhezone ist insbesondere für das Lesen von Barcodes im 'auto'-Modus wichtig. Sie verhindert, dass einfache Barcodes innerhalb einer Strichsequenz eines längeren und/oder komplexeren Barcodes gefunden werden. Normalerweise liefern Werte zwischen *2* und *4* optimale Ergebnisse, weil dadurch falsche Barcodes unterdrückt werden, während kleine Störungen wie Text, Etikettenkanten, etc. immer noch toleriert werden.

Barcodetyp	Linke RZ	Rechte RZ	Barcodetyp	Linke RZ	Rechte RZ
2/5 Industrial	10	10	UPC-A	9	9
2/5 Interleaved	10	10	UPC-A Add-On 2	9	5
Codabar	10	10	UPC-A Add-On 5	9	5
Code 39	10	10	UPC-E	9	7
Code 93	10	10	UPC-E Add-On 2	9	5
Code 128	10	10	UPC-E Add-On 5	9	5
MSI	10	10	GS1-128	10	10
PharmaCode	5	5	GS1 DataBar Omnidir	1	1
EAN-8	7	7	GS1 DataBar Truncated	1	1
EAN-8 Add-On 2	7	5	GS1 DataBar Stacked	1	1
EAN-8 Add-On 5	7	5	GS1 DataBar Stacked Omnidir	1	1
EAN-13	11	7	GS1 DataBar Limited	1	5
EAN-13 Add-On 2	11	5	GS1 DataBar Expanded	1	1
EAN-13 Add-On 5	11	5	GS1 DataBar Expanded Stacked	1	1

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden.

Wertevorschläge: [*'false'*, *'true'*, *'tolerant'*, *1*, *2*, *3*, *4*, *5*]

Werte: [*'false'*, *'true'*, *'tolerant'*, *1*, ..., *100*]

Standardwert: 'false'

'start_stop_tolerance': Fordert ein tolerantes (*'high'*) oder ein strenges (*'low'*) Matchingkriterium bei der Suche nach Start-, bzw. Stoppmuster in einer Scanlinie. Ein tolerantes Kriterium erhöht die allgemeine Leserate, insbesondere in Bildern mit schlechtem Kontrast. Anderseits kann diese Einstellung zu ungültigen Dekodierungen in verrauschten Bildern, oder in Bildern mit Symbolen anderer Barcodetyps führen. Ein strenges Kriterium erhöht die Robustheit gegenüber falscher Dekodierung, kann aber auch die allgemeine Leserate mindern. Es ist zu beachten, dass momentan zwei unterschiedliche Kriterien nur für Code 128 und GS1-128 implementiert sind.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden. Obwohl man diesen Parameter für alle Barcodetypen setzen kann, hat er derzeit, wie bereits erwähnt, nur Wirkung auf Code 128 und GS1-128.

Werte: [*'high'*, *'low'*]

Standardwert: 'high'

'min_code_length': Minimale Anzahl dekodierter Zeichen. Falls ein Barcode gefunden wird, dessen Länge geringer ist als der Wert der für den Parameter *'min_code_length'* angegeben ist, so wird das Resultat verworfen und nicht von `find_bar_code` oder `decode_bar_code_rectangle2` zurückgegeben. Dieser Parameter ist hilfreich um falsches Dekodieren in Anwendungen zu vermeiden in denen die minimale Anzahl zu dekodierender Zeichen im Voraus bekannt ist.

Man beachte, dass der Standardwert für diesen Parameter für die meisten Barcodetypen *0* ist. Lediglich für die Barcodetypen '2/5 Industrial' und '2/5 Interleaved' ist der Standardwert *3*. Der Grund dafür ist, dass diese Barcodetypen leicht fälschlicher Weise in Text und anderen Störmustern mit einer Länge von *2* Zeichen erkannt werden.

Wertevorschläge: [*0*, *1*, *2*, ...]

Standardwert:

Barcodetyp	min_code_length
2/5 Industrial	3
2/5 Interleaved	3
alle Anderen	0

'merge_scanlines': Wenn nicht genug Scanlinien (siehe *'min_identical_scanlines'*) erfolgreich dekodiert werden können, beispielsweise weil der Barcode teilweise verdeckt oder beschädigt ist, wird versucht, die vorhande-

nen Scanlinien zu vereinigen. Anschließend wird versucht, die vereinigten Scanlinien zu dekodieren. Dieser zusätzliche Dekodierschritt wird nur für Barcodetypen durchgeführt, die nicht gestapelt sind und kann für eine verbesserte Geschwindigkeit ausgeschaltet werden.

Werte: [*'true'*, *'false'*]

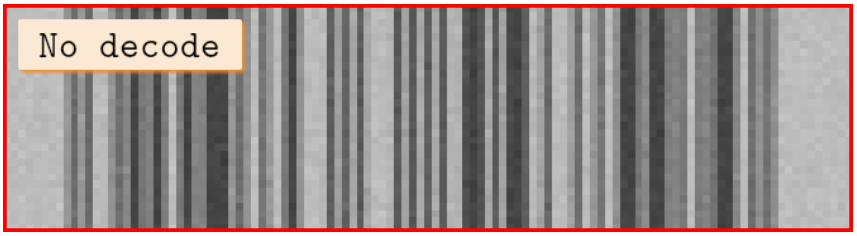
Standardwert: 'true'

'small_elements_robustness': Wenn nicht genug Scanlinien (siehe *'min_identical_scanlines'*) erfolgreich dekodiert werden können, beispielsweise weil der Barcode eine geringe Auflösung hat (kleiner als *2.0* Pixel Elementgröße) wird ein zusätzlicher Dekodierversuch unternommen.

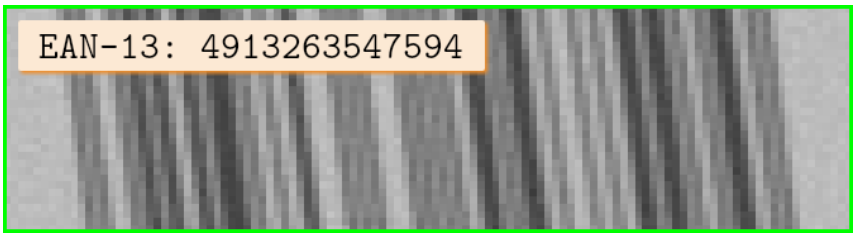
Dieser zusätzliche Dekodierschritt wird nur für Barcodetypen durchgeführt, die nicht gestapelt sind und falls der Parameter *'element_size_min'* kleiner als *2.0* Pixel gesetzt ist.

Zusätzliche Informationen zu diesem Dekodierversuch können über den Operator `get_bar_code_result` abgefragt werden, indem *'status_small_elements_robustness'* für den Parameter `ResultName` gesetzt wird.

Für Anwendungsfälle mit sehr kleinen Elementen (kleiner als *1.3* Pixel) ist es von Vorteil, wenn der Kamerasensor in Bezug auf den Barcode leicht rotiert ist (z.B. um 7 Grad). In dieser niedrigen Auflösung sind wichtige Informationen über den Barcode nur dann über die Zeilen oder Spalten des Sensors verteilt, wenn er rotiert ist. Die beiden folgenden Abbildungen zeigen diesen Zusammenhang.



Ein Barcode mit *0.9* Elementgröße ohne Rotation des Sensors. Die relevanten Informationen des Barcodes sind nicht mehr im Bild vorhanden.



Der gleiche Barcode wie in der vorherigen Abbildung mit Rotation des Sensors. Die Zeilen des Bildes beinhalten mehr Informationen über den Barcode und dieser kann erfolgreich dekodiert werden.

Ein weiterer wichtiger Hinweis ist, dass die Unschärfe durch das optische System möglichst gering gehalten werden muss.

Das Eingabebild sollte nicht vorverarbeitet werden (z.B. über Operatoren wie `emphasize`, `zoom_image_size` oder `mean_image`), da sonst wichtige Informationen verloren gehen würden.

Man beachte, dass es unter optimalen Bedingungen (wenig Unschärfe und leichte Rotation des Sensors) möglich ist, Barcodes mit bis zu *0.6* Pixel Elementgröße zu dekodieren.

Der zusätzliche Dekodierversuch ist nur kompatibel mit Byte-Bildern als Eingabe und kann für eine verbesserte Geschwindigkeit ausgeschaltet werden.

Werte: [*'true'*, *'false'*]

Standardwert: 'true'

Erscheinungsbild der Module im Bild:

'meas_thresh': Innerhalb einer Scanlinie werden Kanten mit Hilfe eines Schwellenwerts gefunden. *'meas_thresh'* bestimmt, wie dieser Schwellenwert relativ zum Dynamikbereich der Grauwerte entlang der Scanlinie berechnet wird. Liegen in dem Barcode Störungen vor oder ist das Rauschen groß, so sollte *'meas_thresh'* auf größere Werte gesetzt werden.

Typischer Wertebereich: [*0.05* ... *0.2*]

Werte: (*0.0*, ..., *1.0*] - ohne *0.0*

Standardwert: 0.05

'meas_thresh_abs': Der Parameter *'meas_thresh_abs'* wird benutzt, um die falsche Kantendetektion zu verhindern. Sollte eine Scanlinie in eine Bildregion mit zu kleinem Dynamikbereich geraten (z.B. eine überwiegend weiße Region mit Grauwerten nahe 255), wird der Schwellenwert zum Kantendetektion unangemessen klein berechnet. Das führt des Öfteren zur Detektion großer Mengen falscher Kanten. Falls der Schwellenwert, der auf dem Parameter *'meas_thresh'* basiert, kleiner als der Wert des Parameters *'meas_thresh_abs'* wird, wird der letztere Wert als Schwellenwert genommen. Standardmäßig ist *'meas_thresh_abs'* auf *5.0* gesetzt. Sollten Bilder mit höherem Rauschpegel behandelt werden, könnte ein Erhöhen des Parameterwertes nützlich sein. Anderseits, wenn rauschfreie Bilder mit niedrigem Kontrast behandelt werden, könnte dieser Parameter die Detektion richtiger Kanten stören. In solchen Fällen ist es empfehlenswert, den Parameterwert zu mindern oder den Parameter sogar zu deaktivieren (auf *0.0* setzen).

Typischer Wertebereich: [*0.0*, . . . , *10.0*]

Werte: [*0.0*, . . . , *65535*]

Standardwert: 5.0

'contrast_min': Minimaler Kontrast zwischen dem Vorder- und Hintergrund der Barcode-Elemente. Wird dieser Parameter auf Werte über 5 gesetzt, optimiert der Operator `find_bar_code` die Kandidatensuche auf den Kontrastbereich oberhalb des *'contrast_min'* Wertes. `find_bar_code` wird dann alle Kandidaten verwerfen, die einen Kontrast unterhalb des Schwellenwertes *'contrast_min'* besitzen. Hohe *'contrast_min'* Werte führen daher in der Regel zu einer verbesserten Laufzeit des Operators `find_bar_code`. Es ist zu beachten, dass Barcodes mit einem geringeren Kontrast als dem Schwellenwert *'contrast_min'* nicht gefunden werden. Aus Effizienzgründen ist der berechnete Kontrast eines Kandidaten nur eine Approximation. Es sollte ein niedrigeren Schwellenwert verwendet werden, falls ein Barcode verworfen wird und dieser einen Kontrast besitzt, der nahe an dem gewählten Schwellenwert *'contrast_min'* liegt.

Typische Werte: [*0*, *40*, *90*, *120*, . . .]

Standardwert: 0

Symbol spezifische Größen:

'check_char': Barcodes mit optionalem Prüfzeichen werden anhand dieses Parameters interpretiert. Darunter fallen z.B. Code 39, Codabar, 2/5 Industrial und 2/5 Interleaved. Der Standardwert von *'absent'* gibt an, dass kein Prüfzeichen vorliegt. In diesem Fall wird keine Prüfung durchgeführt, und alle Zeichen werden als Daten zurückgeliefert.

Ist dieser Parameter auf *'present'* gesetzt, so wird ein Prüfzeichen erwartet und benutzt um die Korrektheit des Barcodes zu prüfen. Ist die Prüfsumme korrekt, so wird das Prüfzeichen aus den Nutzdaten entfernt. Falls dieses Entfernen nicht erwünscht ist, kann der Parameterwert *'preserved'* eingestellt werden um das Prüfzeichen trotz Prüfung zu erhalten. Ist die Prüfsumme jedoch fehlerhaft, so wird der entsprechende Barcode nicht als Ergebnis zurückgeliefert.

Barcodetypen mit obligatorischem Prüfzeichen haben den Parameter *'check_char'* stets auf *'present'* gesetzt. Ob das Prüfzeichen aus den Nutzdaten entfernt wird oder erhalten bleibt, hängt jedoch vom Standard des jeweiligen Codetypen ab. Beispiele für solche Codetypen sind Code 128, EAN-8, EAN-13 und UPC-A.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden.

Werte: [*'absent'*, *'present'*, *'preserved'*]

Standardwert: 'absent'

'composite_code': An den meisten GS1-tauglichen Barcodes kann eine 2D GS1-Composite Code Komponente angehängt sein. Wenn *'composite_code'* auf *'CC-A/B'* gesetzt ist, wird die Composite Komponente lokalisiert und dekodiert. Standardmäßig, ist *'composite_code'* auf *'none'* gesetzt und so wird der Composite Komponente neben dem Barcode ignoriert. Besitzt ein Barcode des gesuchten Typs keine Composite Komponente, so wird beim Aufruf von `find_bar_code` nur das Ergebnis des Barcodes zurückgeliefert. Zur Zeit werden GS1 Composite Codes nur zusammen mit einem Barcode der Familie GS1 DataBar unterstützt.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für unterschiedliche Barcodetypen gesetzt werden.

Werte: [*'none'*, *'CC-A/B'*]

Standardwert: 'none'

'upce_encodation': UPC-E-Barcodes können in verschiedenen Ausgabeformaten zurückgegeben werden. Standardmäßig ist *'upce_encodation'* auf *'ucc-12'* gesetzt und der dekodierte String wird im UCC-12-Format (bestehend aus 12 Ziffern) zurückgegeben. Wird *'upce_encodation'* auf *'zero-suppressed'* gesetzt, so wird

das Ergebnis im Zero-Suppressed-Format (also mit unterdrückten Nullen an definierten Stellen) zurückgegeben. Dieses Format besteht aus führender Null, sechs kodierten Ziffern und einem implizit kodiertem Prüfzeichen. Dies entspricht dem Format, das von ISO/IEC 15420:2009 gefordert wird.

Dieser Parameter kann mit dem Operator `set_bar_code_param_specific` spezifisch für den Barcodetype UPC-E gesetzt werden.

Werte: [*'ucc-12'*, *'zero-suppressed'*]

Standardwert: 'ucc-12'

Training:

Neben dem manuellen Setzen der Modellparameter mit dem Operator `set_bar_code_param` ist es auch möglich, das Modell mit Hilfe von `find_bar_code` anhand eines oder mehrerer Beispielbilder zu trainieren. Dazu muss zuerst ein Barcode-Modell im Trainings-Modus erstellt werden, oder ein bestehendes Barcode-Modell muss in den Trainings-Modus versetzt werden, indem man in `set_bar_code_param` als generischen Parameter `GenParamName` *'train'* übergibt. Der entsprechende Wert bei `GenParamValue` bestimmt, welche Parameter gelernt werden sollen. Im einzelnen können die folgenden Werte gesetzt werden:

'all': Es werden alle trainierbaren Parameter gelernt.

'element_size_min': Minimale Größe eines Basiselements. Diese Einstellung trainiert den Parameter *'element_size_min'*.

'element_size_max': Maximale Größe eines Basiselements. Diese Einstellung trainiert den Parameter *'element_size_max'*.

'barcode_width_min': Minimale Barcode Breite. This setting activates the training mode for the parameter Diese Einstellung trainiert den Parameter *'barcode_width_min'*.

'meas_thresh': Relativer Schwellenwert zur Kantenbestimmung in Scanlinien. Diese Einstellung trainiert den Parameter *'meas_thresh'*.

'meas_thresh_abs': Absoluter Schwellenwert zur Kantenbestimmung in Scanlinien. Diese Einstellung trainiert den Parameter *'meas_thresh_abs'*.

'orientation': Orientierung des Barcodes. Diese Einstellung trainiert die Parameter *'orientation'* und *'orientation_tol'*. Nach Abschluss des Trainings sollte der Wert für *'orientation_tol'* noch um einen Toleranzwert erhöht werden um auch Barcodes zu finden deren Orientierung sich leicht außerhalb des trainierten Bereichs befindet.

Es können mehrere Parameter in einem Aufruf von `find_bar_code` gemeinsam trainiert werden, indem *'train'* in `set_bar_code_param` in einem Tupel mehrfach mit den entsprechenden Werten übergeben wird: z.B. `GenParamName` = *['train','train']* und `GenParamValue` = *['element_size_min','element_size_max']*. Weiterhin können - im Zusammenhang mit *'train'* = *'all'* - einzelne Parameter explizit vom Training ausgeschlossen werden, indem zusätzlich *'train'* mit den entsprechenden Parametern und einem vorangestellten *'~'* gesetzt wird: z.B. `GenParamName` = *['train','train']* und `GenParamValue` = *['all','~orientation']* trainiert alle Parameter außer *'orientation'*.

Anschließend muss für jedes zu trainierende Bild der Operator `find_bar_code` aufgerufen werden.

Die folgenden Hinweise sollten im Zusammenhang mit dem Training beachtet werden:

- Zusätzlich nötige, nicht zu trainierende Parameter (beispielsweise *'check_char'*) müssen vor dem Training manuell festgelegt werden.
- Soll über mehrere Bilder trainiert werden, ist der Operator `find_bar_code` mehrfach nacheinander aufzurufen.
- Pro Bild können mehrere Barcodes des gewählten Typs sichtbar sein. Alle dekodierten Barcodes werden als Trainings-Beispiele behandelt.
- Bitte prüfen Sie die Ausgabe von `find_bar_code` im Trainings-Modus. Während des Trainings sollten nur tatsächlich korrekte Dekodierungen auftreten. Stellen Sie fest, dass eine falsche Dekodierung erfolgt ist, sollten Sie dieses Trainingsbild entweder verwerfen oder entsprechende Parameter zuvor setzen (z.B. können Sie in einem solchen Fall versuchen, den Wert des Parameters *'min_identical_scanlines'* zu erhöhen).
- Befinden sich im Trainingsbild mehrere Barcodes und soll aber nur über einen bestimmten Barcode trainiert werden, ist der Definitionsbereich des Bildes mit `reduce_domain` entsprechend einzuschränken.
- Sind in einer Anwendung alle Bilder sehr ähnlich, reicht i.d.R. *ein* Trainingsbild für die Parameter *'element_size_min'*, *'element_size_max'*, *'meas_thresh'* und *'meas_thresh_abs'*.

- Kann die Elementgröße variieren, sollten nach Möglichkeit im Training auch Bilder mit den kleinsten und den größten vorkommenden Elementen verwendet werden. Sollte dies nicht sichergestellt werden können, müssen die Grenzen für die Elementgröße nach dem Training von Hand gesetzt bzw. entsprechend modifiziert werden, oder die Elementgrößen werden vom Training ganz ausgenommen.
- Die Modellparameter werden beim ersten Training auf die Modelleigenschaften des im Bild gefundenen Barcodes eingeschränkt. Bei jedem weiteren Training werden sie, falls notwendig, so erweitert, dass neben den bisher trainierten Barcodes auch der neue Barcode vom Modell abgedeckt wird.
- Wurden genügend Bilder trainiert, so kann das Training beendet werden. Zuvor sollten jedoch die trainierten Parameter ausgelesen werden. Mit `query_bar_code_params` können die Namen der trainierten Parameter - aufgeteilt in spezifische und allgemeine Parameter - bestimmt werden und anschließend entsprechend mit `get_bar_code_param_specific` und `get_bar_code_param` ausgelesen werden.

  Zum Beenden des Trainings werden alle trainierten Parameter durch einen Aufruf des Operators `set_bar_code_param` mit den Parametern `GenParamName` = *'train'* und `GenParamValue` = *'˜all'* aus dem Training genommen.

  Alternativ kann auch einfach das existierende Barcode-Modell durch den Operator `clear_bar_code_model` geschlossen werden.
- Während des Trainings sollte der Wert eines Trainingsparameters nicht durch einen Aufruf von `set_bar_code_param` oder `set_bar_code_param_specific` explizit gesetzt werden. Falls das gemacht wird, werden die gesamten internen Trainingsdaten zurückgesetzt und das Training startet mit dem nächsten Aufruf von `find_bar_code` von vorne.
- Kann der zu trainierende Barcode nicht gelesen werden, erzeugt dies keine gesonderte Fehlermeldung, sondern es sind lediglich die Ergebnislisten (DecodedDataStrings, SymbolRegions) leer. Die Modellparameter (insbesondere die trainierten Parameter) werden in diesem Fall nicht modifiziert.
- Timeouts sind während des Trainings deaktiviert.

Sonstiges:

'timeout': Mit Hilfe dieses Parameters ist es möglich, dass `find_bar_code` nach einer definierten Zeit (in ms) abbricht. Dies ist vor allem dann nützlich, wenn maximale Zykluszeiten eingehalten werden müssen. Alle bis zu diesem Abbruch gewonnenen Ergebnisse sind anschließend verfügbar und können mit `get_bar_code_result` abgerufen werden. Werden Werte kleiner oder gleich Null übergeben, wird der Timeout deaktiviert (Default).

Die zeitliche Genauigkeit des Timeouts liegt bei ca. 10 ms. Sie hängt von vielen Faktoren ab, u.a. von der Geschwindigkeit des Rechners und dem *'timer_mode'*, der über `set_system` gesetzt wurde.

`find_bar_code` wirft keinen Fehler, falls ein Timeout auftritt. Um zu prüfen, ob `find_bar_code` abgebrochen wurde, kann der Parameter *'aborted'* mit `get_bar_code_result` abgefragt werden.

Im Trainingsmodus von `find_bar_code` wird der Timeout ignoriert.

Typische Werte: [*'false'*, *-1*, *20*, ..., *100*]

Standardwert: 'false'

'abort': Mit dieser Option ist es möglich `find_bar_code` von einem anderen Thread aus abzubrechen. Wenn `set_bar_code_param` mit *'abort'* aufgerufen wird, wird einer Instanz von `find_bar_code` mit dem Model `BarCodeHandle` in einem anderen Thread signalisiert abzubrechen. Falls es keine solche `find_bar_code` Instanz mit diesem Model gibt, passiert nichts.

Es ist möglich, dass der Operator `find_bar_code` nicht sofort abbricht. Er muss zunächst einen interner Abbruchpunkt erreichen, an dem ein sauberes Verlassen des Operators möglich ist. Dies hängt von Rahmenbedingungen wie Rechnerleistung ab und kann ungefähr bis zu 10ms benötigen. Im Trainingsmodus von `find_bar_code` wird der Parameter ignoriert.

Hinweis: Dies ist die einzige Aktion mit Barcode Handles, die von unterschiedlichen Threads genutzt werden kann, ohne dass zusätzliche Synchronisierung nötig ist.

Standardwert: 'true' (Der Wert wird nicht ausgewertet.)

'persistence': Steuert, ob bestimmte Zwischenergebnisse bei der Symbolsuche mit `find_bar_code` temporär oder persistent im Modell gespeichert werden sollen. Zur Zeit ist es erforderlich *'persistence'* einzuschalten (*'persistence'* = *1*), wenn die Barcode Druckqualität bewertet werden muss (`get_bar_code_result` mit Parameter *'quality_isoiec15416'*) oder die Dekodierungs- Scanlinien inspiziert werden müssen (`get_bar_code_object` mit Parameter *'scanlines_all'* oder *'scanlines_valid'*). Mit Einschalten von *'persistence'* steigt der Speicherbedarf für den Barcode-Leser allerdings auch.

Werte: [*0*, *1*]
Standardwert: 0

Parameter

▷ **BarCodeHandle** (input_control) . . . . . barcode ⇝ *handle*
Handle des zu verwendenden Barcode-Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für den Barcode eingestellt werden sollen.
Defaultwert: 'element_size_min'
Werteliste: `GenParamName` ∈ {'check_char', 'composite_code', 'barcode_height_min', 'barcode_width_min', 'element_size_max', 'element_size_min', 'contrast_min', 'meas_thresh', 'meas_thresh_abs', 'min_identical_scanlines', 'majority_voting', 'num_scanlines', 'orientation', 'orientation_tol', 'persistence', 'start_stop_tolerance', 'stop_after_result_num', 'upce_encodation', 'timeout', 'train', 'quiet_zone', 'element_size_variable', 'min_code_length', 'merge_scanlines', 'small_elements_robustness', 'abort'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *real* / integer / string
Werte der generischen Parameter, die für den Barcode eingestellt werden sollen.
Defaultwert: 8
Wertevorschläge: `GenParamValue` ∈ {0, 0.1, 1, 1.5, 2, 8, 32, 45, 'true', 'false', 'present', 'absent', 'none', 'CC-A/B', 'auto', 'high', 'low', 'ucc-12', 'zero-suppressed'}

Ergebnis

`set_bar_code_param` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BarCodeHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_bar_code_model`

Nachfolger

`find_bar_code`

Alternativen

`set_bar_code_param_specific`

Modul

Bar Code

set_bar_code_param_specific (: : BarCodeHandle, CodeTypes, GenParamName, GenParamValue :)

Setzen ausgewählter Parameter des Barcode-Modells für gewählte Barcodetypen

Der Operator `set_bar_code_param_specific` erlaubt es, Parameter, welche Eigenschaften eines spezifisches Barcodetypen beschreiben, zu verändern. Im Gegensatz zu `set_bar_code_param` können mit `set_bar_code_param_specific` spezifische Parameter für angegebene Barcodetypen geändert werden. Dies ist nützlich, wenn nach unterschiedlichen Barcodetypen innerhalb eines Bildes gesucht wird (siehe Abschnitt Autodiskriminierung in `find_bar_code`).

Die aktuelle Konfiguration für einen spezifischen Barcodetyp kann mit Hilfe von get_bar_code_param_specific ausgelesen werden.

Folgende generische Parameter können typspezifisch gesetzt werden:

'num_scanlines': Maximale Anzahl von Scanlinien per Kandidatregion.
Typische Werte: [*0*, *5*, *10*, *20*, ...]
Standardwert: 0

'min_identical_scanlines': Minimale Anzahl von Scanlinien, welche identische Ergebnisse liefern müssen, damit die Dekodierung eines Symbols akzeptiert wird.
Typische Werte: [*0*, *2*, *3*, ...]
Standardwert:

Barcodetyp	min_identical_scanlines
2/5 Industrial	2
2/5 Interleaved	2
alle Anderen	0

'stop_after_result_num': Anzahl erfolgreich dekodierter Barcodes nach der die Dekodierung angehalten wird.
Typische Werte: [*0*, *1*, *2*, ...]
Standardwert: 0

'orientation': Erwarteter Barcode Orientierungswinkel.
Typischer Wertebereich: [*-90.0 ... 90.0*]
Standardwert: 0.0

'orientation_tol': Toleranz der Orientierung.
Typischer Wertebereich: [*0.0 ... 90.0*]
Standardwert: 90.0

'quiet_zone': Verifikation der Ruhezonen des Barcodesymbols.
Typische Werte: 'false', *'true'*, *1*, *2*, *3*, *4*, *5*
Standardwert: 'false'

'start_stop_tolerance': Toleranz des Matchingkriteriums zum Start-, bzw. Stoppmuster. Momentan nur für Code 128 und GS1-128 implementiert.
Werte: [*'high'*, *'low'*]
Standardwert: 'high'

'check_char': Berücksichtigung eines optionalen Prüfzeichens.
Werte: [*'absent'*, *'present'*, *'preserved'*]
Standardwert: 'absent'

'composite_code': GS1-Composite-Komponente finden und dekodieren.
Werte: [*'none'*, *'CC-A/B'*]
Standardwert: 'none'

'upce_encodation': Ausgabeformat für UPC-E-Barcodes.
Werte: [*'ucc-12'*, *'zero-suppressed'*]
Standardwert: 'ucc-12'

'min_code_length': Minimale Anzahl dekodierter Zeichen.
Standardwert:

Barcodetyp	min_code_length
2/5 Industrial	3
2/5 Interleaved	3
alle Anderen	0

Für weitere Erklärungen, siehe set_bar_code_param.

Es ist möglich mehrwertige Tupel für CodeTypes, GenParamName und GenParamValue zu verwenden. Der n-te Eintrag in einem Tupel entspricht dem n-ten Eintrag des anderen mehrwertigen Tupels. Die folgenden Kombinationen sind erlaubt:

CodeTypes	GenParamName	GenParamValue
einzelwertig	einzelwertig	einzelwertig
einzelwertig	mehrwertig	mehrwertig
mehrwertig	einzelwertig	mehrwertig
mehrwertig	einzelwertig	einzelwertig

Parameter

▷ **BarCodeHandle** (input_control) . . . . . barcode ⇝ *handle*
Handle des zu verwendenden Barcode-Modells.

▷ **CodeTypes** (input_control) . . . . . string(-array) ⇝ *string*
Namen der Barcodetypen für die Parameter gesetzt werden sollen.
Defaultwert: 'EAN-13'
Werteliste: CodeTypes ∈ {'2/5 Industrial', '2/5 Interleaved', 'Codabar', 'Code 39', 'Code 93', 'Code 128', 'EAN-13', 'EAN-13 Add-On 2', 'EAN-13 Add-On 5', 'EAN-8', 'EAN-8 Add-On 2', 'EAN-8 Add-On 5', 'UPC-A', 'UPC-A Add-On 2', 'UPC-A Add-On 5', 'UPC-E', 'UPC-E Add-On 2', 'UPC-E Add-On 5', 'MSI', 'PharmaCode', 'GS1 DataBar Omnidir', 'GS1 DataBar Truncated', 'GS1 DataBar Stacked', 'GS1 DataBar Stacked Omnidir', 'GS1 DataBar Limited', 'GS1 DataBar Expanded', 'GS1 DataBar Expanded Stacked', 'GS1-128'}

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für den Barcode eingestellt werden sollen.
Defaultwert: 'check_char'
Werteliste: GenParamName ∈ {'check_char', 'composite_code', 'min_identical_scanlines', 'num_scanlines', 'orientation', 'orientation_tol', 'start_stop_tolerance', 'stop_after_result_num', 'quiet_zone', 'min_code_length'}

▷ **GenParamValue** (input_control) . . . . . attribute.name-array ⇝ *real* / integer / string
Werte der generischen Parameter, die für den Barcode eingestellt werden sollen.
Defaultwert: 'absent'
Wertevorschläge: GenParamValue ∈ {0, 1, 2, 4, 45, 90, 'true', 'false', 'present', 'absent', 'none', 'CC-A/B', 'high', 'low'}

Ergebnis

set_bar_code_param_specific liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BarCodeHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_bar_code_model

Nachfolger

find_bar_code

Alternativen

set_bar_code_param

Modul

Bar Code

write_bar_code_model (: : BarCodeHandle, FileName :)

Schreibt ein Barcode-Modell in eine Datei.

Der Operator write_bar_code_model schreibt ein Barcode-Modell in eine Datei mit dem Namen FileName. Das so gesicherte Modell kann mit read_bar_code_model später wieder eingelesen und wiederverwendet werden. Gespeichert werden alle generischen und spezifischen Parameter des Modells (siehe set_bar_code_param, bzw. set_bar_code_param_specific). Wenn das Modell im Training-Modus läuft (siehe set_bar_code_param mit Parameter *'train'*), wird der Training-Stand des Modells mitgespeichert. Dadurch kann das Training nach dem Einlesen des Modells fortgesetzt werden (siehe Beispiel). Außer dem Training-Stand werden keine weiteren Ergebnisse des Operators find_bar_code in die Datei gespeichert.

Die Dateiendung für das Barcode-Modell ist in HALCON 'bcm'. Der Handle des zu schreibenden Barcode-Modells ist in BarCodeHandle zu übergeben.

Parameter

▷ **BarCodeHandle** (input_control) ..... barcode ⇝ *handle*
Handle des zu schreibenden Barcode-Modells.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der zu schreibenden Barcode-Datei.
Defaultwert: 'bar_code_model.bcm'
Dateiendung: .bcm

Beispiel

```
*
* Create the bar code model in the training mode
create_bar_code_model ('train', 'all', BarCodeHandle)
*
* Set all additional, non-trained parameters in advance:
* Here, we specify that the training images have check characters
set_bar_code_param (BarCodeHandle, 'check_char', 'present')
*
* Train the model with several images
for I := 1 to 7 by 1
    FileName := 'barcode/25interleaved/25interleaved' + I$'.02'
    read_image (Image, FileName)
    *
    * Apply the training
    find_bar_code (Image, SymbolRegion, BarCodeHandle, '2/5 Interleaved', \
                   DecodedDataStrings)
endfor
*
* The training may be interrupted and the intermediate state
* of the model can be stored in a file
write_bar_code_model (BarCodeHandle, 'bar_code_model.bcm')
*
* RESTORE TRAINING:
* Later, when, e.g., new images are available, the training
* may be restored
read_bar_code_model ('bar_code_model.bcm', BarCodeHandle)
FileName := 'barcode/25interleaved/25interleaved08'
read_image (Image, FileName)
*
* Apply the training to the new image
find_bar_code (Image, SymbolRegion, BarCodeHandle, '2/5 Interleaved', \
```

```
                DecodedDataStrings)
*
* Finally, the training can be completed
set_bar_code_param (BarCodeHandle, 'train', '~all')
*
* The trained model can be stored for ONLINE use
write_bar_code_model (BarCodeHandle, 'trained_bar_code_model.bcm')
*
* ONLINE USE:
read_bar_code_model ('trained_bar_code_model.bcm', BarCodeHandle)
* ...
```

Ergebnis

`write_bar_code_model` liefert den Wert TRUE, falls der übergebene Handle gültig ist und das Modell in die Datei mit dem angegebenen Namen geschrieben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_bar_code_model`, `set_bar_code_param`

Siehe auch

`create_bar_code_model`, `set_bar_code_param`, `find_bar_code`

Modul

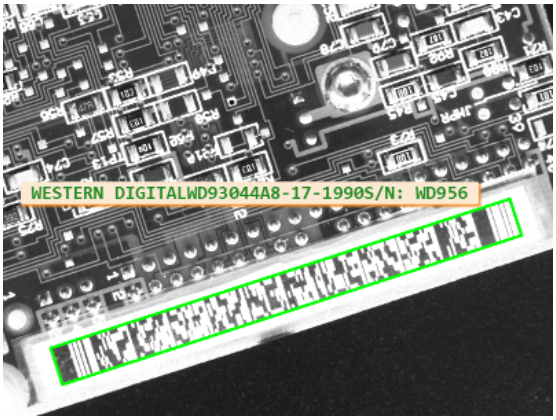
Bar Code

12.2 Datacode

Dieses Kapitel beschreibt Operatoren für das Lesen von 2D-Datacodes.

Konzept des 2D-Datacode-Lesers

2D-Datacode-Symbole stellen eine spezielle Form von zweidimensionalen Mustern dar, die der Kodierung von Text und Zahlen dient. HALCON kann die gängigsten 2D-Datacodes lesen: Data Matrix ECC 200, QR Code, Micro QR Code, Aztec Code, PDF417 und DotCode. Mit Ausnahme des DotCodes enthalten alle diese Codes ein sogenanntes Finder-Pattern und ein Muster für die eigentlichen Daten des Codes. Das Finder-Pattern wird dafür gebraucht, das Muster im Bild zu finden und grundlegende Information über geometrische Eigenschaften, z.B. die Orientierung des Symbols im Bild, zu erhalten. Das Muster für die Daten, also der eigentliche Code, besteht aus vielen punktförmigen, balkenförmigen oder quadratischen Modulen. Durch das spezielle Design der Codes können sie auch dann dekodiert werden, wenn Teile des Symbols Störungen aufweisen.



Lesen eines 2D-Datacodes vom Typ PDF417. Dieses Bild stammt aus dem Beispielprogramm 2d_data_codes_default_settings.hdev.

Im Folgenden werden die einzelnen Schritte, die zum Lesen von 2D-Datacodes benötigt werden, kurz beschrieben.

Anlegen eines 2D-Datacode-Modells: Zuerst muss ein 2D-Datacode-Modell erstellt werden. Dies erfolgt mit dem Operator

- `create_data_code_2d_model`.

Dieses Modell versorgt den 2D-Datacode-Leser mit allen nötigen Informationen bezüglich der Struktur des zu lesenden Codes. Für normal gedruckte Codes reicht es, den Namen des Codes anzugeben. Die passenden Standardparameter wählt HALCON dann automatisch. Für speziellere Fälle können die Modellparameter an ein bestimmtes Symbolerscheinungsbild angepasst werden. Die Parameter können bereits im Rahmen der Erstellung des Modells gesetzt werden oder in einem späteren Schritt angepasst werden.

Ändern der Modellparameter für Nicht-Standard-Codes: Mit den Standardparametern kann der 2D-Datacode-Leser eine große Bandbreite von Codes lesen. Für Nicht-Standard-Codes können die Parameter mit dem folgenden Operator verändert werden:

- `set_data_code_2d_param`.

Hierbei kann entweder ein erweiterter Satz von Standardparametern ausgewählt werden, indem der generische Parameter *'default_parameters'* z.B. auf den Wert *'enhanced_recognition'* gesetzt wird, oder die Parameter werden einzeln gesetzt, um das Modell optimal an die Eigenschaften des genutzten Drucktyps anzupassen. Mit `query_data_code_2d_params` können alle Parameter abgefragt werden, die für einen bestimmten 2D-Datacode-Symbol-Typen verfügbar sind. Die aktuell gesetzten Werte von Parametern können mit `get_data_code_2d_param` abgefragt werden.

Statt die Parameter manuell zu modifizieren, kann das Modell auch automatisch von HALCON trainiert werden. Dazu nutzt man den Operator

- `find_data_code_2d`

mit dem generischen Parameter *'train'*. HALCON sucht dann automatisch nach den Parameterwerten, die am besten geeignet sind, den gegebenen Code zu lesen. Es wird empfohlen, das Training mit vielen Bildern auszuführen, damit alle auftretenden Variationen im Erscheinungsbild der Symbole abgedeckt werden können.

Lesen des 2D-Datacodes: Der 2D-Datacode wird mit dem folgenden Operator gefunden und dekodiert:

- `find_data_code_2d`.

Für jedes erfolgreich dekodierte Symbol liefert der Operator die umgebende XLD- Kontur, einen Kandidaten-Handle, der auf eine Ergebnisstruktur verweist, in der zusätzliche Informationen über das Symbol und den Such- und Leseprozess abgespeichert wurden, sowie den String, der in dem 2D-Datacode-Symbol kodiert ist. Mit den Ergebnis-Handles und den Operatoren `get_data_code_2d_results` und `get_data_code_2d_objects` kann auf zusätzliche Information über den Extraktionsprozess zugegriffen werden. Diese kann sowohl für die Prozessanalyse also auch für Visualisierungszwecke hilfreich sein. Insbesondere erlaubt es `get_data_code_2d_results`, auf verschiedene alphanumerische Ergebnisse zuzugreifen, die während der Suche und dem Lesen des Codes berechnet wurden, und `get_data_code_2d_objects`, auf ikonische Objekte zuzugreifen, die während dem letzten Aufruf von `find_data_code_2d` erzeugt wurden.

Weitere Operatoren

Zusätzlich zu den oben genannten Operatoren erlaubt es `write_data_code_2d_model`, das 2D-Datacode-Modell in eine Datei zu schreiben. Diese kann später z.B. in einer anderen Applikation dazu genutzt werden, eine identische Kopie des Modells zu erzeugen. Solch eine Kopie wird dann direkt über `read_data_code_2d_model` angelegt, also ohne `create_data_code_2d_model` aufrufen zu müssen. Außerdem kann das Modell mit `serialize_data_code_2d_model` und `deserialize_data_code_2d_model` serialisiert bzw. deserialisiert werden.

Glossar

2D-Datacode-Symbol Zweidimensionales grafisches Symbol, das Text und Zahlen kodiert. Es besteht aus dunklen und hellen Punkten, Balken oder Quadraten, die als Module bezeichnet werden. Es gibt verschiedene Arten von 2D-Datacodes. Zwei gebräuchliche Arten sind die Stapelcodes und die Matrixcodes.

Stapelcode 2D-Datacode-Symbol, das einen Stapel aus 1D-Barcodes enthält, die in Zeilen und Spalten angeordnet sind. Um zu gewährleisten, dass der komplette Stapel verarbeitet wird, enthält das Symbol außerdem ein Start- und ein Stoppmuster. Zusätzlich wird das Symbol durch eine Ruhezone umrahmt.

Matrixcode 2D-Datacode-Symbol, das ein zweidimensionales grafisches Muster nutzt, das aus dunklen und hellen Modulen besteht. Das Symbol besteht aus drei Komponenten: dem Finder-Pattern, einem Muster, das die eigentlichen Daten enthält, und einer Ruhezone.

Module Dunkle und helle Punkte, Balken oder Quadrate, aus denen ein 2D-Datacode-Symbol gebildet wird.

Ruhezone Homogener Rahmen um den Rand eines Symbols, das dazu dient, das Symbol im Bild besser von seiner Umgebung abzuheben.

Finder-Pattern Muster, das dazu benutzt wird, das Symbol und seine Orientierung im Bild zu finden. Die Art des Musters hängt vom gewählten 2D-Datacode-Typ ab.

Weiterführende Information

Weitere Details zum Lesen von 2D-Datacodes finden sich im „`Solution Guide Basics`“ und „`Solution Guide on 2D Data Codes`“.

`clear_data_code_2d_model` (: : `DataCodeHandle` :)

Löschen eines 2D-Datacode-Modells und Freigabe des verwendeten Speichers.

Der Operator `clear_data_code_2d_model` löscht ein mit `create_data_code_2d_model` oder `read_data_code_2d_model` angelegtes 2D-Datacode-Modell und gibt den gesamten darin referenzierten Speicherplatz wieder frei. Das Handle des zu löschenden 2D-Datacode-Modells muss in `DataCodeHandle` übergeben werden. Nach dem Aufruf ist es ungültig.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Parameter

▷ **`DataCodeHandle`** (input_control) . . . . . datacode_2d ⇝ *handle*
Handle des zu löschenden 2D-Datacode-Modells.

Ergebnis

`clear_data_code_2d_model` liefert den Wert TRUE, falls das übergebene Handle gültig ist, und das darüber referenzierte Modell vollständig freigegeben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DataCodeHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

`create_data_code_2d_model`, `read_data_code_2d_model`

Modul

Data Code

create_data_code_2d_model (: : SymbolType, GenParamName, GenParamValue : DataCodeHandle)

Anlegen eines generischen Modells für eine Klasse von 2D-Datacodes.

Mit Hilfe von create_data_code_2d_model wird ein generisches Modell für eine Klasse von 2D-Datacodes angelegt. Als Ergebnis liefert der Operator in DataCodeHandle ein Handle zurück, das für alle weiteren Datacode-Operationen verwendet wird, beispielsweise um das Modell zu modifizieren, um im Bild nach Symbolen zu suchen und diese zu lesen, oder um später auf die Ergebnisse der Symbolsuche zuzugreifen.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Unterstützte Symboltypen

Der Parameter SymbolType legt den Typ der zu lesenden 2D-Datacode-Symbole fest. Derzeit werden sechs 2D-Datacode-Typen unterstützt: *'Data Matrix ECC 200'*, *'QR Code'*, *'Micro QR Code'*, *'PDF417'*, *'Aztec Code'* und *'DotCode'*. Zusätzlich werden vier weitere GS1 Typen unterstützt: *'GS1 DataMatrix'*, *'GS1 QR Code'*, *'GS1 Aztec Code'* und *'GS1 DotCode'*. Die Datamatrix-Codes ECC 000-140 werden nicht unterstützt, der QR-Code hingegen kann sowohl in der älteren Model-1-Version als auch in der neuen Model-2-Version gelesen werden. Der PDF417 kann sowohl in seiner herkömmlichen als auch in seiner kompakten Form ('Compact/Truncated PDF417') gelesen werden. Der Aztec Code kann im 'compact', 'full_range' und 'rune' (siehe ISO/IEC 24778: 2008 (E) Annex E) Format gelesen werden. Die Symbolstruktur der vier GS1 Symbologien ist identisch mit der Struktur ihrer nicht-GS1-Varianten - Data Matrix ECC 200, QR Code, Aztec Code bzw. DotCode. Deswegen gelten alle typspezifischen Parameter, Einstellungen und Vorgaben, die Data Matrix ECC 200, QR Code, Aztec Code oder DotCode betreffen, auch für ihre entsprechende GS1-Variante. Die GS1 Symbologien haben lediglich zusätzliche Vorgabe bezüglich des Formats der im jeweiligen Symbol kodierten Daten. Die Daten müssen in sogenannten "GS1 Application Element Strings"organisiert werden, welche in den allgemeinen GS1 Spezifikationen definiert sind.

Alle Datacode-Leser unterstützen das Extended Channel Interpretation (ECI) Protokoll. Wenn das Symbol einen ECI-Code enthält, werden dem Standard entsprechend alle Backslashs ('\', ASCII-Code 92), die im normalen Datenstrom vorkommen, in den Ausgabedaten verdoppelt ('\\'), damit diese von der ECI-Sequenz *'\nnnnnn'* unterschieden werden können.

Ob ein Symbol einen ECI-Code enthält und damit, ob Backslashs verdoppelt wurden oder nicht, kann der Symbology Identifier Kennung entnommen werden. Diese kann mit dem Operator get_data_code_2d_results, mit dem Parameter *'symbology_ident'* für jedes erfolgreich gelesene Symbol abgefragt werden. Wie in der so zurückgelieferten Zahl kodiert ist, ob das Symbol ECI-Codes enthält oder nicht, kann den jeweiligen Symbol-Standards oder der Operatorbeschreibung get_data_code_2d_results entnommen werden.

Der Datacode-Leser stellt den Ausgabedaten in keinem Fall, also auch dann nicht, wenn das Symbol ECI-Codes enthält, automatisch die Symbology Identifier Kennung voran. Wenn ein solcher Datenstrom für die weitere Verarbeitung notwendig sein sollte, muss die Sequenz bestehend aus Identifierflag und Symbolkennung (']d' für Data Matrix ECC 200 Codes, ']Q' für QR-Codes, ']L' für PDF417 Codes, ']z' für Aztec Codes bzw. ']J' für DotCodes) sowie der als *'symbology_ident'* zurückgelieferten Optionsnummer in der Anwendung vor dem Datenstrom manuell ausgegeben werden. Dementsprechend liefert der Parameter *'symbology_ident'* für die GS1 Symbologien die folgenden Werte zurück: 2 für GS1 DataMatrix, 3 für GS1 QR Code und 1 für GS1 Aztec und GS1 DotCode, welche den Identifiern ']d2', ']Q3', bzw. ']z1' und ']J1' entsprechen.

Standardeinstellung des Datacode-Modells

In der Grundeinstellung sind die Modellparameter so gewählt, dass eine relativ große Gruppe von Symbolen in vertretbarer Zeit erkannt wird, wobei in dieser Einstellung aus Laufzeitgründen nur Symbole gelesen werden können, die gewissen Einschränkungen unterliegen (vgl. dazu die folgende Tabelle). Wurde das Modell wie weiter unten beschrieben modifiziert, kann es durch Setzen des generischen Parameters *'default_parameters'* auf *'standard_recognition'* (Operator set_data_code_2d_param) jederzeit wieder in diese Grundeinstellung zurückgesetzt werden.

Für manche Modellparameter ist der Standardwert abhängig vom gewählten Modus (*'standard_recognition'*, *'enhanced_recognition'* oder *'maximum_recognition'*). Diese Parameter sind in der nachfolgenden Tabelle gelistet. Detailliertere Parameterbeschreibungen, sowie die restlichen verfügbaren Modellparameter sind in der Operatorreferenz von set_data_code_2d_param zu finden.

Modellparameter	*'standard_recognition'*	*'enhanced_recognition'*	*'maximum_recognition'*
'polarity'	'dark_on_light' (Dunkle Symbole auf hellem Hintergrund)	'any' (Dunkle Symbole auf hellem Hintergrund und andersherum)	'any' (Dunkle Symbole auf hellem Hintergrund und andersherum)
'contrast_min' (Aztec Code, Micro QR Code, QR Code, PDF417)	30	10	10
'symbol_cols_min', *'symbol_cols_max'* (PDF417)	1 ... 20	1 ... 30	1 ... 30
'symbol_rows_min', *'symbol_rows_max'* (PDF417)	5 ... 45	3 ... 90	3 ... 90
'module_size_min', *'module_size_max'* (Aztec Code, Data Matrix ECC 200, Micro QR Code, QR Code)	6 ... 20 Pixel	4 (2 bei guter Schärfe) ... 100 Pixel	4 (1 bei guter Schärfe) ... 100 Pixel
'module_size_min', *'module_size_max'* (DotCode)	4 ... 100 Pixel	2 ... 100 Pixel	2 ... 100 Pixel
'module_width_min', *'module_width_max'* (PDF417)	3 ... 15 Pixel	3 (2 bei guter Schärfe) ... 100 Pixel	3 (1 bei guter Schärfe) ... 100 Pixel
'module_aspect_min', *'module_aspect_max'* (PDF417)	1 ... 4	1 ... 10	1 ... 10
'small_modules_robustness' (Alle Codetypen außer DotCode)	'low' (niedrig)	'low' (niedrig)	'high' (hoch)
'module_gap_max' (Aztec Code, Micro QR Code, QR Code)	'small' (< 10% der Modulgröße)	'big' (< 50% der Modulgröße)	'big' (< 50% der Modulgröße)
'module_gap_max' (Data Matrix ECC 200)	'no'	'small' (< 10% der Modulgröße)	'big' (< 50% der Modulgröße)
'module_gap_max' (DotCode)	'no'	'small' (< 10% der Modulgröße)	'small' (< 10% der Modulgröße)
'slant_max' (Data Matrix ECC 200)	0.1745 (10°)	0.5235 (30°)	0.5235 (30°)
'module_grid' (Data Matrix ECC 200)	'fixed' (fest)	'any' (fest oder variabel)	'any' (fest oder variabel)
'finder_pattern_tolerance' (Data Matrix ECC 200)	'low' (niedrig)	'low' (niedrig)	'any' (niedrig oder hoch)
'contrast_tolerance' (Data Matrix ECC 200)	'low' (niedrig)	'low' (niedrig)	'any' (niedrig oder hoch)
'candidate_selection' (Data Matrix ECC 200)	'default' (Standard)	'extensive' (umfangreich)	'extensive' (umfangreich)
'candidate_selection' (DotCode)	'default' (Standard)	'extensive' (umfangreich)	'all' (alle)
'finder_pattern_tolerance' (Aztec Code)	'low' (niedrig)	'high' (hoch)	'high' (hoch)
'position_pattern_min' (QR Code)	3	2	2

Ändern der Modelleinstellung

Ist bekannt, dass einzelne Parameter der zu lesenden Symbole diesen Vorgaben (möglicherweise) nicht entsprechen (erscheinen im Bild beispielsweise die Symbole hell vor dunklem Hintergrund oder ist mit einem sehr geringen Kontrast zu rechnen), oder ergeben erste Tests, dass einzelne Symbole in der Grundeinstellung nicht lesbar sind, können entweder die entsprechenden Werte – wenn sie bekannt sind – einzeln angepasst werden, wobei alle anderen Einstellungen unverändert bleiben, oder aber das gesamte Modell wird durch Setzen des generischen Parame-

ters *'default_parameters'* auf den Wert *'enhanced_recognition'* in einem Schritt stark erweitert, so dass eine deutlich größere Gruppe von 2D-Datacode-Symbolen erkannt werden kann. Da die Suche mit einem derart allgemeinen Modell allerdings deutlich aufwendiger ist, kann sich die Rechenzeit des Operators `find_data_code_2d` bei der Symbolsuche z.T. erheblich erhöhen. Dies ist insbesondere dann zu erwarten, wenn in einem Bild *kein* lesbarer Datacode gefunden werden kann, wenn das Symbol heller als der Hintergrund ist oder wenn die Module sehr klein sind.

Durch Setzen des generischen Parameters *'default_parameters'* auf den Wert *'maximum_recognition'* wird das Modell im Vergleich zu *'enhanced_recognition'* nochmals erweitert, so dass Datacodes mit sehr kleiner Modulgröße robuster gelesen werden können (siehe *'small_modules_robustness'*). Für den Fall des Data Matrix ECC 200 unterscheidet sich der Modus *'maximum_recognition'* vom Modus *'enhanced_recognition'* insofern, dass auch Symbole mit einem gestörten oder teilweise fehlenden Finder-Pattern gefunden werden können (siehe *'finder_pattern_tolerance'*). Die Verwendung dieses Modus kann zu einer erhöhten Rechenzeit und einem erhöhten Speicherbedarf des Operators `find_data_code_2d` bei der Symbolsuche und der Dekodierung führen.

Aus diesen Gründen sollte das Modell immer durch Setzen der bekannten Symbolparameter soweit wie möglich eingeschränkt werden. Dies kann hier direkt beim Erzeugen des Modells oder aber später mit dem Operator `set_data_code_2d_param` erfolgen, wofür bei beiden Operatoren die generischen Parameter `GenParamName` und `GenParamValue` zur Verfügung stehen. So sollte z.B. im Falle des Data Matrix ECC 200 bei der Verwendung des Modus *'maximum_recognition'* die Symbolgröße soweit wie möglich eingeschränkt werden. Eine detaillierte Beschreibung der Parameter findet sich beim Operator `set_data_code_2d_param`.

Schließlich steht als eine weitere Möglichkeit, das Modell an eine bestimmte Symbolform anzupassen, auch ein Trainingsmodus zur Verfügung. Der Operator `find_data_code_2d` erlaubt es, mit der Parametereinstellung *'train'* die Symbolparameter aus einem oder mehreren Beispielbildern zu trainieren.

In jedem Fall können die aktuell eingestellten Modellparameter mit `get_data_code_2d_param` ausgelesen werden. Die Namen der Parameter, die für das konkrete Modell gesetzt bzw. abgefragt werden können, liefert `query_data_code_2d_params`.

Parameter

▷ **`SymbolType`** (input_control) . . . . . string ⇝ *string*
Typ des 2D-Datacodes.
Defaultwert: 'Data Matrix ECC 200'
Werteliste: `SymbolType` ∈ {'Data Matrix ECC 200', 'QR Code', 'Micro QR Code', 'PDF417', 'Aztec Code', 'DotCode', 'GS1 DataMatrix', 'GS1 QR Code', 'GS1 Aztec Code', 'GS1 DotCode'}

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für das 2D-Datacode-Modell eingestellt werden sollen.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'default_parameters', 'strict_model', 'persistence', 'polarity', 'mirrored', 'contrast_min', 'candidate_selection', 'model_type', 'version', 'version_min', 'version_max', 'symbol_size', 'symbol_size_min', 'symbol_size_max', 'symbol_cols', 'symbol_cols_min', 'symbol_cols_max', 'symbol_rows', 'symbol_rows_min', 'symbol_rows_max', 'symbol_shape', 'module_size', 'module_size_min', 'module_size_max', 'small_modules_robustness', 'module_width', 'module_width_min', 'module_width_max', 'module_aspect', 'module_aspect_min', 'module_aspect_max', 'module_gap', 'module_gap_min', 'module_gap_max', 'slant_max', 'module_grid', 'position_pattern_min', 'strict_quiet_zone', 'timeout', 'finder_pattern_tolerance', 'contrast_tolerance', 'additional_levels'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Werte der generischen Parameter, die für das 2D-Datacode-Modell eingestellt werden sollen.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'standard_recognition', 'enhanced_recognition', 'maximum_recognition', 'yes', 'no', 'any', 'dark_on_light', 'light_on_dark', 'square', 'rectangle', 'small', 'big', 'fixed', 'variable', 'low', 'high', 'default', 'extensive', 0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 30, 50, 70, 90, 12, 14, 16, 18, 20, 22, 24, 26, 32, 36, 40, 44, 48, 52, 64, 72, 80, 88, 96, 104, 120, 132, 144}

▷ **`DataCodeHandle`** (output_control) . . . . . datacode_2d ⇝ *handle*
Handle für den Zugriff auf das 2D-Datacode-Modell

Beispiel

```
* Two simple examples that show the use of create_data_code_2d_model
* to detect a Data matrix ECC 200 code and a QR Code.
```

```
* (1) Create a model for reading simple QR Codes
*     (only dark symbols on a light background will be read)
create_data_code_2d_model ('QR Code', [], [], DataCodeHandle)
* Read an image
read_image (Image, 'datacode/qrcode/qr_workpiece_01')
* Read the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                   ResultHandles, DecodedDataStrings)
* Clear the model
clear_data_code_2d_model (DataCodeHandle)

* (2) Create a model for reading a wide range of Data matrix ECC 200 codes
*     (this model will also read light symbols on dark background)
create_data_code_2d_model ('Data Matrix ECC 200', 'default_parameters', \
                           'enhanced_recognition', DataCodeHandle)
* Read an image
read_image (Image, 'datacode/ecc200/ecc200_cpu_010')
* Read the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                   ResultHandles, DecodedDataStrings)
* Clear the model
clear_data_code_2d_model (DataCodeHandle)
```

Ergebnis

`create_data_code_2d_model` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind und das Modell erzeugt werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_data_code_2d_param`, `find_data_code_2d`

Alternativen

`read_data_code_2d_model`

Siehe auch

`clear_data_code_2d_model`

Literatur

International Standard ISO/IEC 16022: „Information technology - Automatic identification and data capture techniques - Data Matrix bar code symbology specification“; Reference number ISO/IEC 16022:2006 (E); ISO/IEC 2006.

International Standard ISO/IEC 15438: „Information technology - Automatic identification and data capture techniques - PDF417 bar code symbology specification“; Reference number ISO/IEC 15438:2006 (E); ISO/IEC 2006.

International Standard ISO/IEC 18004: „Information technology - Automatic identification and data capture techniques - QR Code 2005 bar code symbology specification“; Reference number ISO/IEC 18004:2006 (E); ISO/IEC 2006.

International Standard ISO/IEC 24778: „Information technology - Automatic identification and data capture techniques - Aztec Code bar code symbology specification“; Reference number ISO/IEC 24778:2008 (E); ISO/IEC 2008.

GS1 General Specifications; Version 12; Issue 1, Jan-2012; GS1.

Modul

Data Code

deserialize_data_code_2d_model (
: : SerializedItemHandle : DataCodeHandle)

Deserialisieren eines serialisierten 2D-Datacode-Modells.

deserialize_data_code_2d_model deserialisiert ein 2D-Datacode-Modell, welches mit serialize_data_code_2d_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte 2D-Datacode-Modells wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten 2D-Datacode-Modell mit dem Handle DataCodeHandle gespeichert.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **DataCodeHandle** (output_control) ..... datacode_2d ⇝ *handle*
Handle des 2D-Datacode-Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_data_code_2d_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_data_code_2d_model

Nachfolger

find_data_code_2d

Alternativen

create_data_code_2d_model

Siehe auch

serialize_data_code_2d_model, clear_data_code_2d_model

Modul

Data Code

find_data_code_2d (Image : SymbolXLDs : DataCodeHandle, GenParamName,
GenParamValue : ResultHandles, DecodedDataStrings)

Suchen und Lesen von 2D-Datacode-Symbolen im Eingabebild sowie Training des 2D-Datacode-Modells.

Der Operator find_data_code_2d sucht im Eingabebild (Image) nach 2D-Datacode-Symbolen und liest den darin kodierten String aus. Dafür muss zuvor mit Hilfe von create_data_code_2d_model oder read_data_code_2d_model ein Modell für eine Klasse von 2D-Datacodes erstellt und der dabei erzeugte Handle in DataCodeHandle an find_data_code_2d übergeben werden. Um in einem Bild gezielt nach mehr als einem Symbol zu suchen, kann in dem generischen Parameter GenParamName = *'stop_after_result_num'* die Anzahl der gesuchten Datacodes übergeben werden (GenParamValue).

Als Ergebnis liefert der Operator für jedes erfolgreich detektierte Symbol eine XLD-Kontur mit dem umschließenden Viereck (SymbolXLDs), einen Kandidaten-Handle, der auf eine Ergebnisstruktur verweist, in der zusätzliche Informationen über das Symbol und den Such- und Leseprozess abgespeichert wurden (ResultHandles), sowie den String, der in dem Datacode-Symbol kodiert ist (DecodedDataStrings). Ist der String länger als 1024 Zeichen, wird er auf 1020 Zeichen gefolgt von '...' gekürzt. Der Zugriff auf den kompletten String kann in diesem Fall nur mit Hilfe des ResultHandles und dem Operator get_data_code_2d_results mit dem Parameter *'decoded_data'* erfolgen, wobei die einzelnen Zeichen des Strings in einem Tupel zurückgegeben werden.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Die Symbolstruktur von GS1 DataMatrix, GS1 QR Code, GS1 Aztec Code und GS1 DotCode ist identisch der Struktur von Data Matrix ECC 200, QR Code, Aztec Code, bzw. DotCode. Deswegen gelten alle typspezifischen Parameter, Einstellungen und Regel, die Data Matrix ECC 200, QR Code, Aztec oder DotCode betreffen, auch für ihre entsprechende GS1-Variante. Die GS1 Symbologien haben lediglich zusätzliche Vorgaben bezüglich des Formats der im jeweiligen Symbol kodierten Daten. Die Daten müssen in sogenannten "GS1 Application Element Strings"organisiert werden, welche in den allgemeinen GS1 Spezifikationen definiert sind. Wenn zum Beispiel `DataCodeHandle` als GS1 DataMatrix erstellt wurde, dann liest `find_data_code_2d` ein Symbol nur wenn das Symbol ein gültiges Data Matrix ECC 200 Symbol ist und nur wenn seine Daten nach den GS1 Regeln formatiert sind. Von den "GS1 Application Element Strings"wird nur überprüft, ob die gelesenen Application Identifier im Standard "GS1 General Specifications"existieren. Wenn stattdessen ein Data Matrix ECC 200 Symbol nur allgemeine Daten enthält, wird er vom Operator `find_data_code_2d` nicht gelesen. Dasselbe stimmt auch für GS1 QR Code, GS1 Aztec Code und GS1 DotCode.

Anpassen des Modells

Sollte ein im Bild sichtbarer 2D-Datacode nicht erkannt werden, ist zu überprüfen, ob das Symbol die im Modell beschriebenen Eigenschaften besitzt. Dabei ist insbesondere auf

- die richtige Polarität (*'polarity'*, hell auf dunkel oder dunkel auf hell),
- die Symbolgröße (*'symbol_size'* beim Data Matrix ECC 200, Aztec Code und DotCode),
- *'version'* beim QR-Code,
- *'format'* beim Aztec Code,
- die Modulgröße (*'module_size'* beim Data Matrix ECC 200, QR Code, Aztec Code und DotCode, *'module_width'* und *'module_aspect'* beim PDF417),
- eine mögliche Spiegelung des Symbols (*'mirrored'*),
- sowie den eingestellten Mindestkontrast (*'contrast_min'* beim Aztec Code, QR Code und PDF417)

zu achten. Weitere relevante Größen sind beim Data Matrix ECC 200, beim QR Code und beim Aztec Code der Zwischenraum zwischen benachbarten Vordergrundmodulen (*'module_gap'*) und beim Data Matrix ECC 200 der maximal erlaubte Winkel des Finder-Patterns (*'slant_max'*). Die aktuellen Einstellungen für diese Werte lassen sich mit dem Operator `get_data_code_2d_param` abfragen. Gegebenenfalls können die entsprechenden Parameter mit Hilfe des Operators `set_data_code_2d_param` an das 2D-Datacode-Symbol im Bild angepasst werden.

Eine möglichst genaue Anpassung des Modells an die gesuchte Datacode-Klasse empfiehlt sich darüber hinaus auch aus Laufzeitgründen. Im Allgemeinen gilt, dass die Laufzeit von `find_data_code_2d` umso höher ist, je allgemeiner das Modell definiert wurde. Zu beachten ist dabei, dass sich das ungünstige Laufzeitverhalten auf Grund eines zu allgemeinen Modells vor allem dann negativ auswirkt, wenn im Bild *kein* gültiger 2D-Datacode gefunden werden kann.

Trainieren des Modells

Neben dem manuellen Setzen der Modellparameter mit dem Operator `set_data_code_2d_param` ist es auch möglich, das Modell mit Hilfe von `find_data_code_2d` anhand eines oder mehrerer Beispielbilder zu trainieren. Dazu muss als generischer Parameter `GenParamName` *'train'* übergeben werden. Der entsprechende Wert bei `GenParamValue` bestimmt, welche Parameter aus dem aktuellen Bild gelernt werden sollen. Im einzelnen können die folgenden Werte gesetzt werden:

- Alle 2D-Datacode-Modelle:

  'all': Es werden alle trainierbaren Parameter gelernt.

  'symbol_size': Größe (und Form) des Datacode-Symbols.
  Für Data Matrix ECC 200 , PDF417 und DotCode werden die Modellparameter *'symbol_cols_min'*, *'symbol_cols_max'*, *'symbol_rows_min'* und *'symbol_rows_max'* trainiert. Zusätzlich dazu wird für Data Matrix ECC 200 der Modellparameter *'symbol_shape'*, welcher die Form des Symbols (Rechteck oder Quadrat) angibt, trainiert.
  Bei QR-Codes und Micro-QR-Codes kann alternativ als Wert auch *'version'* übergeben werden. Dabei werden die Modellparameter *'symbol_size_min'*, *'symbol_size_max'*, *'version_min'* und *'version_max'* trainiert.
  Für Aztec Codes werden die Modellparameter *'symbol_size_min'* und *'symbol_size_max'* trainiert.

'module_size': Größe der Module. Die Modellparameter, die hierbei trainiert werden, sind *'module_size_min'* und *'module_size_max'*.
Bei PDF417-Codes umfasst die Modulgröße die Modulbreite sowie das Seitenverhältnis der Module. Durch das Training werden dadurch die Modellparameter *'module_width_min'*, *'module_width_max'*, *'module_aspect_min'* und *'module_aspect_max'* beeinflusst.

'polarity': Polarität der Symbole, d.h. sind die Symbole heller oder dunkler als der Hintergrund. Hierbei wird der Modellparameter *'polarity'* trainiert.

'mirrored': Sind die Symbole im Bild gespiegelt oder nicht. Hierbei wird der Modellparameter *'mirrored'* trainiert.

- Data Matrix ECC 200, Aztec Code, PDF417, QR Code, und Micro QR Code:

  'small_modules_robustness': Robustheit der Dekodierung gegenüber Datacodes mit sehr kleiner Modulgröße. Hierbei wird der Modellparameter *'small_modules_robustness'* trainiert.

- nur Aztec Code, PDF417 und QR-Code:

  'contrast': Minimaler Kontrast, der eingestellt werden muss, um die Symbole detektieren zu können. Hierbei wird der Modellparameter *'contrast_min'* trainiert.

- Alle 2D-Datacode-Modelle bis auf PDF417:

  'module_shape': Vorhandensein von Lücken zwischen benachbarten Vordergrundmodulen. Hierbei werden die Modellparameter *'module_gap_min'* und *'module_gap_max'* trainiert.

- nur Aztec Code und Data Matrix ECC 200:

  'finder_pattern_tolerance': Die verwendete Toleranz der Suche gegenüber einem gestörten oder teilweise fehlenden Finder-Pattern. Hierbei wird der Modellparameter *'finder_pattern_tolerance'* trainiert.

- nur Data Matrix ECC 200 und DotCode:

  'candidate_selection': Auswahl der zu verarbeitenden Kandidatenregionen. Hierbei wird der Modellparameter *'candidate_selection'* trainiert.

- nur Data Matrix ECC 200:

  'contrast_tolerance': Die verwendete Toleranz bezüglich lokaler Kontrastvariationen. Hierbei wird der Modellparameter *'contrast_tolerance'* trainiert.

  'module_grid': Algorithmus zur Berechnung der Modulpositionen (festes oder variables Gitter). Hierbei wird der Modellparameter *'module_grid'* trainiert.

  'image_proc': Anpassen verschiedener Bildverarbeitungsparameter. Im Moment wird hier lediglich der maximale Slant beim ECC 200 eingestellt, in Zukunft können aber weitere Parameter folgen. Hierbei wird der Modellparameter *'slant_max'* trainiert.

- nur QR-Code:

  'model_type': Spezifikation des QR-Code-Symbols nach Modelltyp 1 oder 2. Hierbei wird der Modellparameter *'model_type'* trainiert.

- nur Aztec Code:

  'additional_levels': Die Anzahl zusätzlich zu untersuchender Pyramidenebenen bei der Suche. Hierbei wird der Modellparameter *'additional_levels'* trainiert.

Es können mehrere zu trainierende Parameter kombiniert und gemeinsam übergeben werden, indem *'train'* in einem Tupel mehrfach mit den entsprechenden Werten übergeben wird: z.B. `GenParamName` = *['train','train']* und `GenParamValue` = *['polarity','module_size']*. Weiterhin können - im Zusammenhang mit *'train'* = *'all'* einzelne Parameter explizit vom Training ausgeschlossen werden, indem zusätzlich *'train'* mit den entsprechenden Parametern und einem vorangestellten *'~'* gesetzt wird: z.B. `GenParamName` = *['train','train']* und `GenParamValue` = *['all','~contrast']*.

Die folgenden Hinweise sollten im Zusammenhang mit dem Training beachtet werden:

- Soll über mehrere Bilder trainiert werden, ist der Operator `find_data_code_2d` mehrfach nacheinander mit dem gesetzten *'train'*-Parameter und den verschiedenen Trainingsbildern aufzurufen.
- Es kann auch über mehrere Symbole in einem Bild trainiert werden. Dazu ist in `GenParamName` zusätzlich zu *'train'* der generische Parameter *'stop_after_result_num'* zu übergeben. In `GenParamValue` wird neben den zu trainierenden Parametern die Anzahl der Symbole, die sich im Bild befinden, übergeben.

- Befinden sich im Trainingsbild mehrere Symbole und soll aber nur über ein bestimmtes Symbol trainiert werden, ist der Definitionsbereich des Bildes mit reduce_domain entsprechend einzuschränken.
- Sind in einer Anwendung alle Bilder sehr ähnlich, reicht i.d.R. *ein* Trainingsbild, wenn die Symbolgröße (in Modulen) bei allen Bildern identisch ist, die Vorder- und Hintergrundmodule gleich groß sind (d.h. auch keine Lücken zwischen benachbarten Vordergrundmodulen auftreten), der Hintergrund keine signifikante Textur aufweist und der Kontrast in allen Bildern etwa gleich groß ist. In allen anderen Fällen sollten mehrere Bilder zum Training herangezogen werden.
- Kann die Symbolgröße variieren, sollten nach Möglichkeit im Training auch Bilder mit den kleinsten und den größten vorkommenden Symbolen verwendet werden. Sollte dies nicht sichergestellt werden können, müssen die Grenzen für die Symbolgröße nach dem Training von Hand gesetzt bzw. entsprechend modifiziert werden, oder die Symbolgröße wird vom Training ganz ausgenommen.
- Die Modellparameter werden beim ersten Training auf die Modelleigenschaften des im Bild gefundenen Symbols eingeschränkt. Bei jedem weiteren Training werden sie, falls notwendig, so erweitert, dass neben den bisher trainierten Symbolen auch der neue Datacode vom Modell abgedeckt wird. Durch ein Zurücksetzen des Modells mit set_data_code_2d_param auf eine der Standardeinstellung (*'default_parameters'* = *'standard_recognition'*, *'enhanced_recognition'* oder *'maximum_recognition'*) wird neben den Modellparametern auch der Trainingsstatus des Modells in den Grundzustand zurückgesetzt. Mit set_data_code_2d_param und *'trained'* kann der Trainingsstatus einzelner Parameter auf trainiert gesetzt werden. Diese gesetzten Parameter werden beim Training nicht mehr überschrieben. Falls nötig wird nur der Suchraum durch das Training erweitert.
- Kann das zu trainierende Symbol nicht gelesen werden, erzeugt dies keine gesonderte Fehlermeldung, sondern es sind lediglich die Ergebnislisten (SymbolXLDs, ResultHandles, DecodedDataStrings) leer. Das Modell wird in diesem Fall nicht modifiziert.

Es ist zu beachten, dass möglicherweise gesetzte Zeitschranken im Trainingsmodus nicht beachtet werden (siehe *'timeout'* in set_data_code_2d_param).

Übersicht über die Funktionsweise der Symbolsuche

- Alle 2D-Datacode-Modelle:

  Abhängig von den Einstellungen im 2D-Datacode-Modell (vgl. set_data_code_2d_param) absolviert find_data_code_2d intern i.d.R. mehrere Durchläufe bei der Suche nach 2D-Datacode-Symbolen. Begonnen wird die Suche auf der höchsten Pyramidenstufe, auf der die einzelnen Module (entsprechend der vorgegebenen maximalen Modulgröße) noch zu unterscheiden sein müssten. Außerdem kann in jeder Pyramidenstufe die Vorverarbeitung entsprechend der eingestellten Modulform variiert werden. Erlaubt das Modell sowohl helle Symbole vor dunklem Hintergrund als auch dunkle Symbole vor hellem Hintergrund (*'polarity* = *any'*), werden in der jeweiligen Pyramidenstufe zuerst die Durchläufe zum Finden von dunklen Symbolen abgearbeitet, danach wird nach hellen Symbolen gesucht. Ein solcher Durchlauf besteht immer aus zwei Phasen: der Suche, bei der i.d.R. eine Reihe von Symbolkandidaten anhand des Finder-Patterns detektiert werden, und der Auswertephase, in der in einer niedrigeren Pyramidenstufe der Reihe nach alle neu gefundenen Kandidaten auf ihre Lesbarkeit hin untersucht werden. In der Suchphase können in einem entsprechend strukturierten Bild auch eine Reihe von Kandidaten an Stellen generiert werden, die keine Symbole enthalten. Die Bearbeitung dieser Kandidaten wird allerdings in der anschließenden Auswertephase i.d.R. sehr früh abgebrochen.

  Der gesamte Operatoraufruf wird schließlich beendet, wenn entweder die geforderte Anzahl an 2D-Datacode-Symbolen gefunden wurde, alle Suchdurchläufe abgearbeitet wurden oder eine Zeitüberschreitung (siehe set_data_code_2d_param) eingetreten ist. Die Anzahl der zu lesenden Symbole, kann mit dem generischen Parameter GenParamName = *'stop_after_result_num'* gesteuert werden. Der entsprechende Wert ist in GenParamValue zu übergeben. Wird nichts übergeben, wird gestoppt, sobald ein Symbol erfolgreich dekodiert werden konnte.

- Data Matrix ECC 200:

  Für einfache Bilder, d.h. Bilder, die:

  – nur Symbole enthalten, die einen hohen Kontrast und eine großzügige Ruhezone aufweisen und
  – einen homogenen Hintergrund haben,

  kann eine weniger komplexe und (in einfachen Fällen) schnellere Methode zur Symbolsuche verwendet werden. Hierfür muss der generische Parameter GenParamName = *'symbol_search'* auf *'rudimental'* gesetzt

werden. Es ist zu beachten, dass in diesem Fall folgende Parameter keine Auswirkung mehr haben: *'module_gap_min'*, *'module_gap_max'*, *'module_gap'*, *'finder_pattern_tolerance'*, *'contrast_tolerance'* und *'candidate_selection'*. Außerdem können mit dieser Methode keine Parameter trainiert werden. Standardmäßig ist *'symbol_search'* auf *'default'* gesetzt.

Abfrage von Ergebnissen der Symbolsuche

Die Operatoren `get_data_code_2d_results` und `get_data_code_2d_objects` erlauben den Zugriff auf zahlreiche Informationen zur Symbolsuche im Allgemeinen wie z.B. die Anzahl der Durchläufe und der untersuchten Kandidaten, und – zusammen mit den `ResultHandles` – über die Symbolkandidaten im einzelnen, wie die Symbol- und Modulgröße, den Kontrast oder die im Symbol kodierten Rohdaten. Diese Operatoren ermöglichen dabei auch den Zugriff auf all jene Symbolkandidaten, die zwar untersucht aber letztendlich nicht gelesen werden konnten. Insbesondere lässt sich so ermitteln, ob die Vorverarbeitung für ein abgebildetes aber nicht gelesenes 2D-Datacode-Symbol überhaupt einen Kandidaten generiert hat und – anhand einer Statusvariablen – warum dessen Bearbeitung schließlich abgebrochen wurde. Nähere Information finden sich bei den Operatoren `get_data_code_2d_results` und `get_data_code_2d_objects`.

Timeout und Abbruch

Der Operator `find_data_code_2d` kann sowohl mit einem Timeout sowie auch manuell abgebrochen werden. Mit dem Operator `set_data_code_2d_param` kann man einen Timeout festlegen. Falls `find_data_code_2d` in einen Timeout läuft, werden alle bis dahin erkannten Codes zurückgegeben. Zusätzlich gibt es die Möglichkeit `find_data_code_2d` manuell abzubrechen, indem man `set_data_code_2d_param` mit *'abort'* von einem anderen Thread aufruft.

In beiden Fällen, kann mit `get_data_code_2d_results` und dem Parameter *'aborted'* abgefragt werden, ob der Operator abgebrochen wurde.

Des Weiteren kann der Operator `find_data_code_2d` mittels `set_operator_timeout` oder `interrupt_operator` abgebrochen werden. Dabei werden keine Ergebnisse, aber eine Fehlermeldung zurückgegeben. Wenn `find_data_code_2d` mit `interrupt_operator` abgebrochen wurde, wird H_ERR_CANCEL (22) zurückgegeben. Wenn `find_data_code_2d` mit `set_operator_timeout` abgebrochen wurde, wird H_ERR_TIMEOUT (9400) zurückgegeben. Das Abbrechen durch einen dieser Operatoren wird nur im *'cancel'*-Modus unterstützt.

Spezielle Parallelisierung für Data Matrix ECC 200

Der Operator `find_data_code_2d` unterstützt eine spezielle Parallelisierung für Data Matrix ECC 200. Diese zusätzliche interne Parallelisierung kann unter gewissen Umständen zu einer signifikant verkürzten Ausführungszeit führen. Im Allgemeinen sind damit schnellere Rechenzeiten zu erwarten, wenn bspw. mit `set_data_code_2d_param` für einen oder mehrere der Parameter *'polarity'*, *'contrast_tolerance'* und *'finder_pattern_tolerance'* der Wert *'any'* gesetzt wird, oder sich die gesetzten Werte für *'module_gap_min'* und *'module_gap_max'* unterscheiden, somit also eine Reihe von Parameterwerten berücksichtigt werden muss. Dies gilt insbesondere, wenn *'default_parameters'* auf *'enhanced_recognition'* oder *'maximum_recognition'* gesetzt worden ist.

Bitte beachten Sie, dass der Speicherverbrauch mit der Anzahl der parallel laufenden Threads ansteigt. Um den Speicherverbrauch zu reduzieren, kann mit `set_system` entweder die Anzahl der Threads mit dem Parameter *'thread_num'* gesetzt werden, oder der Cache für temporären Speicher mit dem Parameter *'temporary_mem_cache'* ausgeschaltet werden. Die Data Matrix ECC 200 spezifische Parallelisierung kann auch auf einfache Weise ausgeschaltet werden, indem der generische Parameter `GenParamName` = *'specific_parallelization'* auf *'disable'* gesetzt wird. Beachten Sie, dass dadurch nur die zusätzliche interne Parallelisierung für Data Matrix ECC 200 ausgeschaltet wird. Bestimmte Teilaufgaben können immer noch parallel abgearbeitet werden. Weitere Informationen zur Deaktivierung der gesamten Parallelisierung finden Sie in der Operatorreferenz von `set_system` mit den Parametern *'thread_num'* und *'parallelize_operators'*. Generell sollte es jedoch nicht notwendig sein, die spezifische Parallelisierung auszuschalten. Standardmäßig ist *'specific_parallelization'* auf *'enable'* gesetzt.

Chinesische Zeichen

Enthält ein QR Code chinesische Zeichen, die nach dem chinesischen nationalen Standard GBT 18284-2000 codiert sind, gibt `find_data_code_2d` diese Zeichen UTF-8 codiert in `DecodedDataStrings` zurück, sofern der Systemparameter *'filename_encoding'* auf *'utf8'* gesetzt ist. Der Inhalt von *'decoded_data'*, welcher mit `get_data_code_2d_results` abgefragt werden kann, wird nicht nach UTF-8 konvertiert.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
  Eingabebild. Falls das Bild eine reduzierte Domäne besitzen sollte, so wird der Bereich eingeschränkt in dem nach Datacodes gesucht wird. In der Regel führt dies zu einer Verbesserung der Laufzeit. Allerdings sollte man beachten, dass ein Datacode nicht mehr gefunden werden könnte, wenn er nicht vollständig in der Domäne des Bildes liegt. In seltenen Fällen kann es vorkommen, dass Datacodes außerhalb der Domäne gefunden werden. Sind diese Ergebnisse unerwünscht, müssen sie nachträglich aussortiert werden.
- ▷ **SymbolXLDs** (output_object) . . . . . xld_cont(-array) ⇝ *object*
  XLD-Konturen, die die erfolgreich dekodierten Datacode-Symbole umschließen. Die Konturen beginnen in der linken oberen Ecke und folgt im Uhrzeigersinn der Standardausrichtung (siehe *'orientation'* bei `get_data_code_2d_results`)



  Reihenfolge der Punkte der `SymbolXLDs`

- ▷ **DataCodeHandle** (input_control) . . . . . datacode_2d ⇝ *handle*
  Handle des zu verwendenden 2D-Datacode-Modells.
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Namen von (optionalen) Parametern für die Steuerung des Verhaltens des Operators.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'train', 'stop_after_result_num', 'symbol_search', 'specific_parallelization'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *integer* / real / string
  Die zu den optionalen generischen Parametern gehörenden Werte.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {'all', 'model_type', 'symbol_size', 'version', 'module_size', 'small_modules_robustness', 'module_shape', 'polarity', 'mirrored', 'contrast', 'candidate_selection', 'module_grid', 'finder_pattern_tolerance', 'contrast_tolerance', 'additional_levels', 'image_proc', 'rudimental', 'default', 1, 2, 3, 'enable', 'disable'}
- ▷ **ResultHandles** (output_control) . . . . . integer(-array) ⇝ *integer*
  Handles der erfolgreich dekodierten 2D-Datacode-Symbole.
- ▷ **DecodedDataStrings** (output_control) . . . . . string(-array) ⇝ *string*
  Die dekodierten Datenstrings aller im Bild detektierten 2D-Datacode-Symbole.

Beispiel

```
* Examples showing the use of find_data_code_2d.
* First, the operator is used to train the model, afterwards it is used to
* read the symbol in another image.

* Create a model for reading Data matrix ECC 200 codes
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
* Read a training image
read_image (Image, 'datacode/ecc200/ecc200_cpu_007')
* Train the model with the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all', \
                   ResultHandles, DecodedDataStrings)
*
```

```
* End of training / begin of normal application
*

* Read an image
read_image (Image, 'datacode/ecc200/ecc200_cpu_010')
* Read the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                   ResultHandles, DecodedDataStrings)

* Display all symbols, the strings encoded in them, and the module size
dev_set_color ('green')
for i := 0 to |ResultHandles| - 1 by 1
    select_obj (SymbolXLDs, SymbolXLD, i+1)
    dev_display (SymbolXLD)
    get_contour_xld (SymbolXLD, Row, Col)
    set_tposition (WindowHandle, max(Row), min(Col))
    write_string (WindowHandle, DecodedDataStrings[i])
    get_data_code_2d_results (DataCodeHandle, ResultHandles[i], \
                              ['module_height','module_width'], ModuleSize)
    new_line (WindowHandle)
    write_string (WindowHandle, 'module size = ' + ModuleSize[0] + 'x' + \
                                 ModuleSize[1])
endfor

* Clear the model
clear_data_code_2d_model (DataCodeHandle)
```

Ergebnis

`find_data_code_2d` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DataCodeHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_data_code_2d_model`, `read_data_code_2d_model`, `set_data_code_2d_param`

Nachfolger

`get_data_code_2d_results`, `get_data_code_2d_objects`, `write_data_code_2d_model`

Siehe auch

`create_data_code_2d_model`, `set_data_code_2d_param`, `get_data_code_2d_results`, `get_data_code_2d_objects`

Literatur

GS1 General Specifications; Version 12; Issue 1, Jan-2012; GS1.

Modul

Data Code

get_data_code_2d_objects (: DataCodeObjects : DataCodeHandle, CandidateHandle, ObjectName :)

Zugriff auf ikonische Objekte, die bei der Suche nach einem 2D-Datacode angelegt werden bzw. diesen beschreiben.

Der Operator get_data_code_2d_objects erlaubt den Zugriff auf ikonische Objekte, die beim letzten Aufruf von find_data_code_2d während der Suche nach 2D-Datacodes oder beim Lesen der gefundenen Symbole angelegt wurden. Für den Zugriff muss neben dem Namen des gewünschten Bildobjekts (ObjectName) auch das zugrundeliegende 2D-Datacode-Modell (DataCodeHandle) angegeben werden. Außerdem muss in CandidateHandle ein Handle auf einen Symbolkandidaten, wie ihn find_data_code_2d oder get_data_code_2d_results liefern, oder ein String, der eine Gruppe von Kandidaten beschreibt (vgl. get_data_code_2d_results) übergeben werden.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Für den Fall, dass *'discard_undecoded_candidates'* in set_data_code_2d_param auf *'yes'* gesetzt wurde, können Ergebnisse nur zu erfolgreich dekodierten Kandidaten abgefragt werden.

Es ist zu beachten, dass auf einige Objekte nur zugegriffen werden kann, wenn für das Modell die Persistenz auf *1* gesetzt wurde (vgl. set_data_code_2d_param). Das Setzen der Persistenz muss vor dem Aufruf von find_data_code_2d entweder beim Anlegen des Modells mit create_data_code_2d_model oder mit Hilfe von set_data_code_2d_param erfolgen.

Im einzelnen wird der Zugriff auf die folgenden Objekte unterstützt:

Regionen der Module

- Alle 2D-Datacode-Modelle:
  'module_1_rois': Alle Module, die als Vordergrund interpretiert wurden (Modulbit ist gesetzt).
- 2D-Datacode-Modelle bis auf DotCode:
  'module_0_rois': Alle Module, die als Hintergrund interpretiert wurden (Modulbit ist nicht gesetzt).

Die hierbei zurückgelieferten Regionen entsprechen den Bildbereichen, in denen eine Interpretation der Module durchgeführt wurde. Die Abfrage dieser Werte kann, da ein Regionenarray zurückgegeben wird, immer nur für genau ein Ergebnisobjekt erfolgen, d.h. an CandidateHandle darf kein Gruppenbezeichner übergeben werden. Die Modellpersistenz muss auf den Wert *1* gesetzt sein. Und schließlich ist die Abfrage der Modulregionen nur für Symbole sinnvoll, die als 2D-Datacode-Symbole erkannt wurden und für die zumindest eine gültige Symbolgröße ermittelt werden konnte. Für Kandidaten, deren Bearbeitung vorher abgebrochen wurde, stehen die Modulregionen nicht zur Verfügung.

XLD Kontur

'candidate_xld': Die den Kandidaten oder das dekodierte Symbol umgebende XLD-Kontur.

Die Kontur kann für alle Kandidaten abgefragt werden, einzeln oder auch gruppenweise. Die Persistenzeinstellung spielt bei diesem Objekt keine Rolle.

Pyramidenbilder

'search_image': Pyramidenbild, in dem der Kandidat gefunden wurde.

'process_image': Pyramidenbild, in dem die weitere Bearbeitung des Kandidaten erfolgte.

Die Persistenzeinstellung spielt bei diesen Bildern ebenfalls keine Rolle.

Parameter

▷ **DataCodeObjects** (output_object) ..... object(-array) ⇝ *object*
Ikonische Objekte, die als Zwischenergebnisse bei der 2D-Datacode-Suche oder deren Auswertung anfallen.

▷ **DataCodeHandle** (input_control) ..... datacode_2d ⇝ *handle*
Handle des zu verwendenden 2D-Datacode-Modells.

▷ **CandidateHandle** (input_control) ..... integer ⇝ *integer* / string
Handle des 2D-Datacode-Kandidaten. Entweder ein Integer (üblicherweise das ResultHandle von find_data_code_2d) oder ein String, der eine Gruppe von Kandidaten beschreibt.
Defaultwert: 'all_candidates'
Wertevorschläge: CandidateHandle ∈ {0, 1, 2, 'all_candidates', 'all_results', 'all_undecoded', 'all_aborted'}

▷ **ObjectName** (input_control) . . . . . string ⇝ *string*
Name des ikonischen Objekts, das abgefragt werden soll.
Defaultwert: 'candidate_xld'
Werteliste: ObjectName ∈ {'module_1_rois', 'module_0_rois', 'candidate_xld', 'search_image', 'process_image'}

Beispiel

```
* Example demonstrating how to access the iconic objects of the data code
* search.

* Create a model for reading Data matrix ECC 200 codes
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
set_data_code_2d_param (DataCodeHandle, 'default_parameters', 'maximum_recognition')
set_data_code_2d_param (DataCodeHandle, 'persistence', 1)
* Read an image
read_image (Image, 'datacode/ecc200/ecc200_disturbed_012')
* Read the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                   ResultHandles, DecodedDataStrings)

* Get the handles of all candidates that were detected as a symbol but
* could not be read
get_data_code_2d_results (DataCodeHandle, 'all_undecoded', 'handle', \
                          HandlesUndecoded)

* For every undecoded symbol, get the contour and the classified
* module regions
for I := 0 to |HandlesUndecoded| - 1 by 1
  * Display image.
  dev_display (Image)
  dev_set_draw ('margin')
  * Get the contour of the symbol.
  dev_set_color ('blue')
  get_data_code_2d_objects (SymbolXLD, DataCodeHandle, \
                            HandlesUndecoded[I], 'candidate_xld')
  dev_display (SymbolXLD)
  * Get the module regions of the foreground modules
  dev_set_color ('green')
  get_data_code_2d_objects (ModuleFG, DataCodeHandle, \
                            HandlesUndecoded[I], 'module_1_rois')
  dev_display (ModuleFG)
  * Get the module regions of the background modules
  dev_set_color ('red')
  get_data_code_2d_objects (ModuleBG, DataCodeHandle, \
                            HandlesUndecoded[I], 'module_0_rois')
  dev_display (ModuleBG)
  * Stop for inspecting the image.
  stop ()
endfor

* Clear the model
clear_data_code_2d_model (DataCodeHandle)
```

Ergebnis

get_data_code_2d_objects liefert den Wert TRUE, falls die übergebenen Parameter korrekt und die angeforderten Objekte für die letzte Symbolsuche verfügbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_data_code_2d`, `query_data_code_2d_params`

Nachfolger

`get_data_code_2d_results`

Siehe auch

`query_data_code_2d_params`, `get_data_code_2d_results`, `get_data_code_2d_param`, `set_data_code_2d_param`

Modul

Data Code

get_data_code_2d_param (: : DataCodeHandle, GenParamName : GenParamValue)

Abfrage von einem oder mehreren Parametern, die das 2D-Datacode-Modell beschreiben.

Mit Hilfe des Operators `get_data_code_2d_param` lassen sich die Parameter, durch die das 2D-Datacode-Modell beschrieben wird, abfragen. Die Namen der Parameter, die abgefragt werden sollen, werden in `GenParamName` übergeben, die korrespondierenden Werte werden in `GenParamValue` zurückgeliefert. Sämtliche Parameter können mit `set_data_code_2d_param` gesetzt und jederzeit verändert werden. Eine Liste der Parameternamen, die für den ausgewählten 2D-Datacode-Type abgefragt werden können, liefert der Operator `query_data_code_2d_params`.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Es ist zu beachten, dass die Symbolstruktur von GS1 DataMatrix, GS1 QR Code, GS1 Aztec Code und GS1 DotCode der Struktur von Data Matrix ECC 200, QR Code, Aztec Code, bzw. DotCode identisch ist. Deswegen gelten alle typspezifischen Parameter, Einstellungen und Regeln, die Data Matrix ECC 200, QR Code, Aztec Code oder DotCode betreffen, auch für ihre entsprechende GS1-Variante. Im Folgenden wird das ausführliche Auflisten typspezifischer Parameter einzelner GS1-Typen ausgelassen, um die Lesbarkeit der Dokumentation zu erleichtern. Stattdessen können die relevante Parameter eines GS1-Datacodetyps entweder von den Parametern von dessen nicht-GS1-Variante hergeleitet werden oder explizit durch `query_data_code_2d_params` mit dem Parameter *'get_model_params'* abgefragt werden.

Im Einzelnen kann auf die folgenden Parameter – gruppiert nach verschiedenen Kategorien und Datacode-Typen – zugegriffen werden:

Größe und Form des Symbols:

- Data Matrix ECC 200 (inklusive Finder-Pattern):

  'symbol_cols_min': Mindestanzahl der Spalten des Symbols in Modulen.

  'symbol_cols_max': Höchstanzahl der Spalten des Symbols in Modulen.

  'symbol_rows_min': Mindestanzahl der Zeilen des Symbols in Modulen.

  'symbol_rows_max': Höchstanzahl der Zeilen des Symbols in Modulen.

  'symbol_shape': Mögliche Einschränkungen hinsichtlich der Form des Symbols (Rechteck und/oder Quadrat): *'square'*, *'rectangle'* oder *'any'*. Seit HALCON 7.1.1 werden bei der Symbolsuche beide Formen gleich behandelt.

- QR-Code (inklusive Finder-Pattern):

  'model_type': Typ des QR-Code-Modells: *1*, *2*, *0* (für *'any'*)

  'version_min': Kleinste zu lesende Symbol-Version: [*1*...*40*] (Modelltyp 1: [*1*...*14*])

  'version_max': Größte zu lesende Symbol-Version: [*1*...*40*] (Modelltyp 1: [*1*...*14*])

  'symbol_size_min': Kleinste zu lesende Symbolgröße in Modulen (dieser Wert korrespondiert direkt mit *'version_min'*): [*21*...*177*] (Modelltyp 1: [*21*...*73*])

  'symbol_size_max': Größte zu lesende Symbolgröße in Modulen (dieser Wert korrespondiert direkt mit *'version_max'*): [*21*...*177*] (Modelltyp 1: [*21*...*73*])

- PDF417:

  'symbol_cols_min': Mindestanzahl der Datenspalten des Symbols in Codewords (je 17 Module), d.h. exklusive der Codewords der Start/Stop-Patterns sowie der beiden Codewords der Row-Indicators.

  'symbol_cols_max': Höchstanzahl der Datenspalten des Symbols in Codewords (je 17 Module), d.h. exklusive der Codewords der Start/Stop-Patterns sowie der beiden Codewords der Row-Indicators.

  'symbol_rows_min': Mindestanzahl der Zeilen des Symbols in Modulen.

  'symbol_rows_max': Höchstanzahl der Zeilen des Symbols in Modulen.

- Aztec Code (inklusive Finder-Pattern):

  'format': Format des Aztec Codes: Durch Leerzeichen getrennte Liste mit den Werten *'compact'*, *'full_range'* oder *'rune'*.

  'symbol_size_min': Kleinste zu lesende Symbolgröße in Modulen: [*11*...*151*]

  'symbol_size_max': Größte zu lesende Symbolgröße in Modulen: [*11*...*151*]

- DotCode:

  'symbol_cols_min': Mindestanzahl der Spalten des Symbols in Modulen.

  'symbol_cols_max': Höchstanzahl der Spalten des Symbols in Modulen.

  'symbol_rows_min': Mindestanzahl der Zeilen des Symbols in Modulen.

  'symbol_rows_max': Höchstanzahl der Zeilen des Symbols in Modulen.

Erscheinungsbild der Module im Bild:

- Alle 2D-Datacode-Modelle:

  'polarity': Mögliche Einschränkungen hinsichtlich der Polarität der Module, d.h. ob diese im Bild dunkel vor hellem Hintergrund oder hell vor dunklem Hintergrund erscheinen: *'dark_on_light'*, *'light_on_dark'* oder *'any'*.

  'discard_undecoded_candidates' Steuert, ob Kandidaten, die nicht erfolgreich dekodiert werden konnten, beim Modell gespeichert werden: *'yes'*, *'no'*.

  'mirrored': Angabe über eine mögliche Spiegelung des Symbols (entspricht einem Vertauschen von Spalten und Zeilen): *'yes'*, *'no'* oder *'any'*.

- Alle 2D-Datacode-Modelle bis auf Data Matrix ECC 200 und DotCode:

  'contrast_min': Mindestkontrast zwischen Vorder- und Hintergrund des Symbols im Bild (angegeben in Grauwerten). Dieses Maß korrespondiert mit dem Anstieg der Kanten zwischen Vorder- und Hintergrund.

- Alle 2D-Datacode-Modelle bis auf PDF417:

  'module_size_min': Minimale Größe der Module im Bild in Pixel.

  'module_size_max': Maximale Größe der Module im Bild in Pixel.

  Es kann in drei Stufen angegeben werden, ob benachbarte Vordergrundmodule direkt aneinanderstoßen, oder ob es zwischen ihnen eine Lücke geben kann - *'no'* (keine Lücke) < *'small'* < *'big'* (Bei DotCode sind nur *'no'* und *'small'* möglich):

  'module_gap_min': minimale Lücke.

  'module_gap_max': maximale Lücke.

- Alle 2D-Datacode-Modelle bis auf DotCode:

  'small_modules_robustness': Robustheit der Dekodierung gegenüber Datacodes mit sehr kleiner Modulgröße. Wird der Parameter *'small_modules_robustness'* auf *'high'* gesetzt, so erhöht sich die Wahrscheinlichkeit, dass Datacodes mit sehr kleinen Modulen dekodiert werden können. Zusätzlich sollte in diesem Fall auch die minimale Modulgröße entsprechend angepasst werden, d.h. *'module_size_min'* bzw. *'module_width_min'* (PDF417) sollte auf die angenommene minimale Modulgröße bzw. Modulbreite gesetzt werden. Ist *'small_modules_robustness'* auf *'high'* gesetzt, so kann sich der interne Speicherbedarf von `find_data_code_2d` deutlich erhöhen. Folglich sollte *'small_modules_robustness'* im Normalfall auf *'low'* gesetzt werden.
  Werteliste: 'low', *'high'*
  Default: 'low' (enhanced: *'low'*, maximum: *'high'*)

- PDF417:

'module_width_min': Minimale Breite der Module im Bild in Pixel.

'module_width_max': Maximale Breite der Module im Bild in Pixel.

'module_aspect_min': Minimales Seitenverhältnis der Module im Bild (Höhe zu Breite).

'module_aspect_max': Maximales Seitenverhältnis der Module im Bild (Höhe zu Breite).

- Aztec Code:

  'finder_pattern_tolerance': Toleranz der Suche gegenüber einem gestörten oder teilweise fehlenden Finder-Pattern. Abhängig von diesem Parameter werden verschiedene Algorithmen für die Suche in `find_data_code_2d` verwendet. Für *'low'* wird angenommen, dass alle Ringe des Finder-Patterns extrahiert werden können. Für *'high'* wird angenommen, dass mindestens einer der Ringe des Finder-Patterns extrahiert werden kann.

  'additional_levels': Um die Robustheit des Aztec Code Lesers zu erhöhen, kann die Anzahl der Pyramidenebenen, die zusätzlich zu den durch die minimale und maximale Modulgröße bedingten Ebenen untersucht werden sollen, angegeben werden. [*0*...*2*]

- Data Matrix ECC 200:

  'slant_max': Maximale Abweichung des Winkels im L-förmigen Finder-Pattern vom (idealerweise) rechten Winkel (die Angabe erfolgt im Bogenmaß und korrespondiert mit der möglichen Verzerrung, die beim Druck des Symbols oder bei der Bildaufnahme auftreten kann).

  'finder_pattern_tolerance': Toleranz der Suche gegenüber einem gestörten oder teilweise fehlenden Finder-Pattern. Das Finder-Pattern umschließt sowohl die L-förmige als auch die gegenüberliegende alternierende Seite. In Abhängigkeit von diesem Parameter werden verschiedene Algorithmen für die Suche in `find_data_code_2d` verwendet. In einem Fall (*'low'*) wird davon ausgegangen, dass das Finder-Pattern zum großen Teil vorhanden ist und kaum Störungen aufweist. Im anderen Fall (*'high'*) kann das Finder-Pattern gestört oder teilweise fehlend sein, ohne dass die Erkennung beeinträchtigt wird. Zu beachten ist allerdings, dass bei dieser Variante die Parameter für die Symbolsuche möglichst stark mit `set_data_code_2d_param` eingeschränkt werden sollten, da ansonsten eine erhöhte Rechenzeit für `find_data_code_2d` zu erwarten ist. Zu beachten ist außerdem, dass sich beide Algorithmen leicht in ihrer Robustheit voneinander unterscheiden. Dies kann dazu führen, dass es selbst bei Symbolen mit intaktem Finder-Pattern in Abhängigkeit von *'finder_pattern_tolerance'* zu unterschiedlichen Ergebnissen kommt. Wenn *'high'* ausgewählt wird, können z.B. nur Symbole mit festem Gitter gefunden werden (siehe unten), wodurch die Robustheit gegenüber perspektivischen Verzerrungen abnimmt. Im Fall *'any'* werden beide Algorithmen ausgeführt.

  'module_grid': Angabe darüber, ob die Größe der Module in einem gewissen Rahmen variieren darf oder nicht. In Abhängigkeit von diesem Parameter werden verschiedene Algorithmen für die Berechnung der Modulpositionen verwendet. In einem Fall (*'fixed'*) wird ein festes Gitter, bei dem die Abstände zwischen den Modulmittelpunkten alle gleich sind, verwendet. Im anderen Fall (*'variable'*) wird das Gitter an der alternierenden Seite des Finder-Patterns ausgerichtet. Bei *'any'* werden beide Varianten für das Gitter nacheinander ausprobiert. Zu beachten ist, dass der Wert von *'module_grid'* ignoriert wird, wenn *'finder_pattern_tolerance'* auf *'high'* gesetzt ist. In diesem Fall wird immer von einem festen Gitter ausgegangen.

  'contrast_tolerance': Toleranz der Suche beim Auftreten starker lokaler Kontrastvariationen (z.B. bei Reflexionen oder Überbelichtung). Abhängig von diesem Parameter wird zwischen zwei Algorithmen gewählt. Im Falle von *'high'* ist die Robustheit gegenüber starken lokalen Kontrastvariationen erhöht. Für *'low'* wird ein Algorithmus gewählt, der zwar etwas weniger robust bei lokalen Kontrastvariationen ist, dafür aber schneller. Unter normalen Umständen ist daher *'low'* zu bevorzugen. Setzt man den Parameter auf *'any'*, so werden beide Varianten ausgeführt.

- QR Code:

  'position_pattern_min': Anzahl von Position-Detection-Patterns, die im Bild gut sichtbar sein müssen, damit ein Symbolkandidat generiert wird.

Allgemeines Modellverhalten:

- Alle 2D-Datacode-Modelle:

  'persistence': Steuert, ob bestimmte Zwischenergebnisse bei der Symbolsuche mit `find_data_code_2d` temporär oder persistent im Modell gespeichert werden sollen: *0* (temporär) oder *1* (persistent).

'strict_model': Steuert das Verhalten von `find_data_code_2d` bei der Detektion von Symbolen, die hinsichtlich der Symbolgröße nicht den Modellvorgaben entsprechen: *'yes'* (strikt: entsprechende Symbole werden verworfen) oder *'no'* (nicht strikt: Symbole werden, wenn sie gelesen werden können, als Ergebnis zurückgeliefert). Es ist zu beachten, dass für DotCode-Symbole die Modellvorgaben bzgl. der Modulgröße (*'module_size_min'* und *'module_size_max'*) auch dann nicht überprüft werden, wenn *'strict_model'* auf *'yes'* gesetzt ist.

'string_encoding': Gibt die erwartete Zeichencodierung des im Symbol kodierten Strings: *'utf8'*, *'locale'*, *'latin1'* oder *'raw'*.

'symbol_type': Gibt den beim Erzeugen mit `create_data_code_2d_model` angegebenen Datacode-Typ zurück.

'timeout': Mit Hilfe dieses Parameters ist es möglich, dass `find_data_code_2d` nach einer definierten Zeit in ms abbricht: *'false'*, *-1*, *20 ... 100*.

- Alle 2D-Datacode-Modelle bis auf Aztec Code:

  'strict_quiet_zone': Steuert das Verhalten von `find_data_code_2d` bei der Detektion von lesbaren Symbolen deren Ruhezone Defekte aufweist. Mögliche Werte:

  'yes': Symbole mit schlechten Bewertungen für ihre Ruhezone werden verworfen und somit nicht als Resultat zurückgegeben. Die Ruhezonen werden ähnlich der Druckbildkontrolle ausgewertet, dabei jedoch nur als ein Modul groß betrachtet. Verworfene Symbole erhalten als *'status'* den Wert *'quiet zone is missing'*.

  'no' **(Default):** Alle lesbaren Symbole werden als Ergebnis zurückgegeben.

- Alle 2D-Datacode-Modelle bis auf DotCode:

  'quality_isoiec15415_aperture_size': Definiert die simulierte Blendenöffnung für die Druckqualitätsprüfung nach ISO/IEC 15415:2011 als Anteil an der Modulgröße. (Siehe ISO/IEC 15415:2011 Abschnitt 7.3.3).

- Data Matrix ECC 200:

  'candidate_selection': Bestimmt die Auswahl der Kandidatenregionen, die verwendet werden, um Symbole im Bild zu finden. Wird dieser Parameter auf *'extensive'* gesetzt, so erhöht sich die Anzahl der generierten Kandidatenregionen und damit die Wahrscheinlichkeit einen Code zu finden. Wenn *'candidate_selection'* auf *'default'* gesetzt ist, werden weniger Kandidatenregionen verwendet.

- DotCode:

  'candidate_selection': Bestimmt die Auswahl der Kandidatenregionen, die verwendet werden, um Symbole im Bild zu finden. Wird dieser Parameter auf *'extensive'* gesetzt, so erhöht sich die Anzahl der generierten Kandidatenregionen und damit die Wahrscheinlichkeit einen Code zu finden. Wird dieser Parameter auf *'all'* gesetzt, werden alle möglichen Kandidaten genutzt und nicht anhand ihrer Wahrscheinlichkeit ein DotCode zu sein gefiltert. Wenn *'candidate_selection'* auf *'default'* gesetzt ist, werden weniger Kandidatenregionen verwendet.

  'max_allowed_error_correction': Bestimmt die maximal zulässige Fehlerkorrektur. Aufgrund der hohen Fehlerkorrekturkapazität, ist es möglich 'false positive' DotCode Kandidaten erfolgreich zu dekodieren. Besonders Kandidaten, die nur einen kleinen Teil eines echten DotCode-Symbols bedecken, werden aufgrund der Fehlerkorrektur eventuell erfolgreich dekodiert. Der Grund dafür ist, dass DotCode-Symbole so gut wie jede Größe annehmen können. Das macht es schwieriger falsche Kandidaten auszusortieren. Um diesem Problem entgegenzuwirken, kann der Parameter *'max_allowed_error_correction'* verwendet werden, um anzugeben wie viel Prozent der Fehlerkorrekturkapazität genutzt werden dürfen. Standardmäßig ist der Wert auf *'1.0'*, also 100% gesetzt. Wird der Wert z.B. auf *'0.5'* herabgesetzt, werden nur noch die Kandidaten als erfolgreich dekodierte Ergebnisse zurückgegeben, die mit maximal 50% genutzter Fehlerkorrektur dekodiert werden konnten.

Es ist möglich, in einem Operatoraufruf obige Parameter beliebig zu kombinieren und gemeinsam in einem Tupel an `GenParamName` zu übergeben. Als Ergebnis wird in `GenParamValue` ein Tupel gleicher Länge zurückgeliefert, dessen Werte mit den entsprechenden Parameternamen korrespondieren.

Parameter

▷ **`DataCodeHandle`** (input_control) ..... datacode_2d ⇝ *handle*
Handle des zu verwendenden 2D-Datacode-Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für das 2D-Datacode-Modell abgefragt werden sollen.
Defaultwert: 'polarity'
Werteliste: GenParamName ∈ {'strict_model', 'persistence', 'polarity', 'mirrored', 'contrast_min', 'candidate_selection', 'discard_undecoded_candidates', 'model_type', 'version_min', 'version_max', 'format', 'string_encoding', 'symbol_size_min', 'symbol_size_max', 'symbol_cols_min', 'symbol_cols_max', 'symbol_rows_min', 'symbol_rows_max', 'symbol_shape', 'module_size_min', 'module_size_max', 'module_width_min', 'module_width_max', 'small_modules_robustness', 'module_aspect_min', 'module_aspect_max', 'module_gap_min', 'module_gap_max', 'slant_max', 'module_grid', 'position_pattern_min', 'strict_quiet_zone', 'timeout', 'finder_pattern_tolerance', 'additional_levels', 'contrast_tolerance', 'quality_isoiec15415_aperture_size', 'decoding_scheme', 'symbol_type', 'max_allowed_error_correction'}

▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Werte der generischen Parameter.

Ergebnis

get_data_code_2d_param liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

query_data_code_2d_params, set_data_code_2d_param, find_data_code_2d

Nachfolger

find_data_code_2d, write_data_code_2d_model

Alternativen

write_data_code_2d_model

Siehe auch

query_data_code_2d_params, set_data_code_2d_param, get_data_code_2d_results, get_data_code_2d_objects, find_data_code_2d

Modul

Data Code

get_data_code_2d_results (: : DataCodeHandle, CandidateHandle, ResultNames : ResultValues)

Abfrage von alphanumerischen Ergebnissen, die bei der Suche nach 2D-Datacode-Symbolen angefallen sind.

Der Operator get_data_code_2d_results erlaubt den Zugriff auf verschiedene alphanumerische Ergebnisse, die während der Suche nach 2D-Datacodes oder beim Lesen der gefundenen Symbole anfallen und so die Ergebnisobjekte – das sind alle untersuchten Kandidaten, unabhängig davon, ob sie gelesen werden konnten oder nicht – genauer beschreiben. Die Ergebnisse beziehen sich dementsprechend meist nicht auf die höchste Auflösung des Symbols, sondern hängen von der Pyramidenstufe ab, in der der Leseprozess abgebrochen wird. Für den Zugriff auf die Ergebnisse muss neben dem Namen des gewünschten Ergebnisses (ResultNames) auch das zugrundeliegende 2D-Datacode-Modell (DataCodeHandle) sowie bei den Ergebnissen, die sich auf einen konkreten Symbolkandidaten beziehen, ein entsprechendes Handle (CandidateHandle) angegeben werden.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Für den Fall, dass *'discard_undecoded_candidates'* in set_data_code_2d_param auf *'yes'* gesetzt wurde, können Ergebnisse nur zu erfolgreich dekodierten Kandidaten abgefragt werden.

Die meisten der Ergebnisse enthalten genau einen Wert. Diese Ergebnisse können für einen konkreten Kandidaten-Handle in einem Aufruf gemeinsam abgefragt werden. Die Werte des Ergebnistupels ResultValues korrespondieren in diesem Fall mit den entsprechenden Parametern in der Eingangsliste ResultNames. Alternativ dazu

kann, wenn diese Ergebnisse einzeln abgefragt werden sollen, statt eines konkreten Kandidaten-Handles auch ein String, der eine Gruppe von Kandidaten beschreibt, angegeben werden, wodurch der entsprechende Wert für alle Kandidaten dieser Gruppe gemeinsam zurückgegeben wird.

Des Weiteren gibt es Ergebnisse, die nicht nur einen einzelnen Wert enthalten, sondern ein Wertetupel (z.B. *'bin_module_data'*). Diese Ergebnisse müssen stets exklusiv für genau einen Kandidaten abgefragt werden, und die Anfrage kann auch nicht mit anderen Ergebnissen in einem Aufruf kombiniert werden.

Schließlich stehen noch eine Reihe an Information bereit, die sich nicht auf einen konkreten Kandidaten sondern auf die Suche im Allgemeinen beziehen. Für sie wird statt eines Kandidaten-Handles der String *'general'* übergeben.

Es ist zu beachten, dass die Symbolstruktur von GS1 DataMatrix, GS1 QR Code, GS1 Aztec Code und GS1 DotCode mit der Struktur von ECC 200, QR Code, Aztec Code, bzw. DotCode identisch ist. Deswegen können alle Ergebnisse welche typspezifisch für Data Matrix ECC 200, QR Code, Aztec Code oder DotCode sind, auch für ihre entsprechende GS1-Variante abgefragt werden. Im Folgenden wird das ausführliche Auflisten typspezifischer Ergebnisse einzelner GS1-Typen ausgelassen, um die Lesbarkeit der Dokumentation zu erleichtern. Stattdessen können die relevante Ergebnisnamen eines GS1-Datacodetyps entweder von den Ergebnisnamen von dessen nicht-GS1-Variante hergeleitet werden oder explizit durch `query_data_code_2d_params` mit dem Parameter *'get_result_params'* abgefragt werden.

Gruppen von Kandidaten

Die folgenden Gruppen von Kandidaten sind vordefiniert und können anstelle eines einzelnen Handles an `CandidateHandle` übergeben werden:

'general': Dieser `CandidateHandle`-Wert wird für Parameter verwendet, die sich auf keinen konkreten Kandidaten sondern auf den letzten `find_data_code_2d`-Aufruf im Allgemeinen beziehen.

'all_candidates': Alle Ergebniskandidaten (einschließlich der erfolgreich dekodierten Symbole), die während des letzten Aufrufs von `find_data_code_2d` untersucht wurden.

'all_results': Alle Symbole, die während des letzten Aufrufs von `find_data_code_2d` erfolgreich dekodiert wurden.

'all_undecoded': All jene Kandidaten des letzten `find_data_code_2d`-Aufrufs, die als Symbol erkannt aber nicht dekodiert werden konnten. Bei diesen Kandidaten hat die Fehlerkorrektur zu viele Fehler entdeckt. Oder aber das Symbol enthält inkonsistente Daten, sodass beim Dekodieren der bereits fehlerkorrigierten Daten ein Fehler auftrat.

'all_aborted': All jene Kandidaten des letzten `find_data_code_2d`-Aufrufs, bei denen die Identifizierung als gültiges 2D-Datacode-Symbol gescheitert ist und die Suche vorzeitig abgebrochen wurde.

Unterstützte Ergebnisse

Zur Zeit wird über `ResultNames` der Zugriff auf folgende Ergebnisse unterstützt, die in `ResultValues` zurückgeliefert werden:

Allgemeine, kandidatenunabhängige Ergebnisse (Alle 2D-Datacode-Modelle) – ***'general'*****:**

- Alle 2D-Datacode-Modelle:

  'min_search_level': Unterste Pyramidenstufe, in der nach Symbolen gesucht wird. Hierbei entspricht das Originalbild der Pyramidenstufe 0.

  'max_search_level': Oberste Pyramidenstufe, in der nach Symbolen gesucht wird. Hierbei entspricht das Originalbild der Pyramidenstufe 0.

  'result_num': Anzahl der erfolgreich dekodierten Symbole insgesamt.

  'candidate_num': Anzahl der bearbeiteten Kandidaten insgesamt.

  'undecoded_num': Anzahl der Kandidaten, die als Symbol identifiziert, jedoch nicht gelesen werden konnten.

  'aborted_num': Anzahl der Kandidaten, bei denen die Bearbeitung vorzeitig abgebrochen wurde, da sie nicht als ein gültiger 2D-Datacode identifiziert werden konnten.

  'aborted': Gibt zurück, ob `find_data_code_2d` im vorangegangenen Durchlauf vorzeitig abgebrochen wurde. Dies kann durch Setzen eines Timeouts (*'timeout'* in `set_data_code_2d_param` oder durch einen manuellen Abbruch (siehe *'abort'* in `set_data_code_2d_param`) verursacht worden.

- Data Matrix ECC 200, Aztec Code, QR Code und Micro QR Code:

Value	Description
0	`find_data_code_2d` wurde vollständig abgeschlossen
1	`find_data_code_2d` wurde durch ein Timeout abgebrochen
2	`find_data_code_2d` wurde durch `set_data_code_2d_param` mit *'abort'* abgebrochen

'quality_isoiec_tr_29158_labels' **und** *'quality_aimdpm_1_2006_labels'***:** Gibt die Namen der Elemente des Qualitätsgradtupels, das von `get_data_code_2d_results` mit Parameter *'quality_isoiec_tr_29158'* oder *'quality_aimdpm_1_2006'* ausgegeben wird, zurück.

- Data Matrix ECC 200, Aztec Code, QR Code, Micro QR Code und PDF417:

  *'pass_num'***:** Gesamtzahl der absolvierten Such-Durchläufe, siehe auch `find_data_code_2d` (Übersicht über die Funktionsweise der Symbolsuche).

  *'quality_isoiec15415_labels'***:** Gibt die Namen der Elemente des Qualitätsgradtupels, das von `get_data_code_2d_results` mit Parameter *'quality_isoiec15415'* ausgegeben wird, zurück. Die Namen der Qualitätselemente unterscheiden sich je nach Datacodemodell.

- Datamatrix ECC 200:

  *'quality_semi_t10_labels'***:** Gibt die Namen der Elemente des Qualitätstupels, das von `get_data_code_2d_results` mit Parameter *'quality_semi_t10_values'* ausgegeben wird, zurück.

Ergebnisse, die sich auf einen konkreten Kandidaten beziehen - Einzelwert Ergebnisse:

Ergebnisse, die genau **einen Wert** beinhalten und somit sowohl für **einzelne Kandidaten** als auch für **Gruppen von Kandidaten** abgefragt werden können. Bei diesen Ergebnissen ist zu beachten, dass für Kandidaten, deren Bearbeitung vorzeitig abgebrochen wurde (vgl. Bearbeitungsstatus *'status'*), nicht alle Ergebnisse sinnvolle Werte liefern:

- Alle 2D-Datacode-Modelle:

  *'handle'***:** Kandidaten-Handle. Dieser Aufruf dient in erster Linie dazu, sich die Handles aller Kandidaten einer der oben beschriebenen Gruppen geben zu lassen.

  *'status'***:** Der Bearbeitungsstatus zeigt für den Kandidaten an, ob die Dekodierung erfolgreich war oder warum die Bearbeitung abgebrochen wurde.

  *'search_level'***:** Pyramidenstufe, in der der Kandidat mit Hilfe des Finder-Patterns gefunden wurde.

  *'process_level'***:** Pyramidenstufe, in der die weitere Bearbeitung des Kandidaten erfolgte.

  *'polarity'***:** Polarität des Symbols. Das ist die Annahme darüber, ob das Symbol im Bild heller oder dunkler als der Hintergrund erscheint.

  *'mirrored'***:** Angabe, ob ein erfolgreich dekodiertes Symbol gespiegelt ist oder nicht. Für ECC 200, PDF417, QR und MicroQR Codes wird *'no'* oder *'yes'* zurückgegeben. Für Aztec Codes wird *'false'* oder *'true'* zurückgegeben. Bei Kandidaten, die nicht gelesen werden konnten, wird hier die vom Modell vorgegebene Annahme über eine mögliche Spiegelung zurückgegeben.

  *'orientation'***:** Ausrichtung eines erfolgreich dekodierten Symbols. Die Null-Grad-Ausrichtung ist in der Abbildung unten dargestellt. Der Winkel ist positiv bei Drehung nach links und hat die Einheit Grad. Er kann im Wertebereich von [-180.0 .. 180.0] Grad liegen.



(1) (2) (3) (4)

Ausrichtung bei 0° eines Aztec-Symbols (1), ECC200-Symbols (2), QR-Code-Symbols (3) und PDF417-Symbols (4). Die Finder Patterns, die die Ausrichtung erkennbar machen, sind in den Abbildungen hervorgehoben.

'symbol_rows', *'symbol_cols'*: Bei ECC 200, QR-Code und Aztec Code: Ermittelte Symbolgröße in Modulen: Anzahl der Zeilen bzw. der Spalten einschließlich des Finder-Patterns; Bei PDF417: Ermittelte Anzahl der Zeilen und Datenspalten (je 17 Module) im Symbol. Weder die Start/Stop-Patterns noch die Row-Indicators werden dabei mitgezählt.

'module_height', *'module_width'*: Größe der Module in Pixel in Richtung der Symbolzeilen bzw. Symbolspalten.

'decoded_string': Im Symbol kodierter String – die Abfrage ist nur für erfolgreich dekodierte Kandidaten sinnvoll und liefert das gleiche Ergebnis wie `find_data_code_2d`.

'decoding_error': Dekodierungsfehler – bei erfolgreich dekodierten Symbolen steht hier die Anzahl der von der Fehlerkorrektur gefundenen und korrigierten Fehler, d.h. der Codewörter, bei deren Lesen zunächst Probleme auftraten. Für den PDF417 beinhaltet dieser Wert auch Auslöschungen (Erasures), d.h. Fehler deren Position bekannt ist. Für Symbole, bei denen die Fehlerkorrektur scheiterte, steht hier ein negativer Fehlercode.

- Datamatrix ECC 200, Aztec Code, QR-Code, Micro QR Code und DotCode:

  'module_gap': Die dem Durchlauf, in dem der Kandidat gefunden wurde, zugrundeliegende Annahme über die Größe der Lücke zwischen benachbarten Modulen.

- Datamatrix ECC 200, Aztec Code, QR-Code, Micro QR Code und PDF417:

  'pass': Nummer des Durchlaufs, in dem der Kandidat gefunden und bearbeitet wurde, siehe auch `find_data_code_2d` (Übersicht über die Funktionsweise der Symbolsuche).

  'small_modules_robustness': Robustheit der Dekodierung gegenüber Datacodes mit sehr kleiner Modulgröße. Für *'low'* wurde der Datacode mit einem Verfahren dekodiert, das nur eine geringe Robustheit gegenüber sehr kleinen Modulgrößen aufweist. Für *'high'* wurde der Datacode mit einem Verfahren dekodiert, das diesbezüglich eine hohe Robustheit aufweist.

  'contrast': Schätzung des Kontrastes innerhalb des Symbols. Dieser Wert basiert auf den Gradienten der Kanten zwischen Finder-Pattern und Hintergrund.

- Datamatrix ECC 200, Aztec Code, QR-Code, PDF417 und DotCode:

  'symbology_ident': Der *Symbology Identifier* wird verwendet um anzuzeigen, dass der Datacode das FNC1- und/oder ECI-Zeichen enthält.
  Das Zeichen FNC1 (*Function 1 Character*) wird verwendet, wenn die Daten in dem Symbol entsprechend spezieller vordefinierter Industrie- oder Anwendungsnormen formatiert sind.
  Das ECI-Protokoll (*Extended Channel Interpretation*) wird im Zusammenhang mit einem 6-stelligen Code verwendet, wenn die nachfolgenden Zeichen entsprechend spezieller Codetabellen, beispielsweise mit einem bestimmten internationalen Zeichensatz, zu interpretieren sind. Im Ausgabestrom werden sie als *'\nnnnnn'* kodiert. Enthält das Symbol ein oder mehrere ECI-Codes, werden alle Backslashs ('\', ASCII-Code 92), die im normalen Datenstrom auftreten, verdoppelt
  Als Identifier wird hier nur der eigentliche Wert *m* ($m \in [0,6]$ (QR-Code), $m \in [0,12]$ (Data Matrix ECC 200 und Aztec Code), $m \in [0,2]$ (PDF417) bzw. $m \in [0,5]$ (DotCode)) entsprechend der Spezifikationen für Datamatrix-, QR-Codes, Aztec Codes, PDF417 und DotCode zurückgeliefert. Falls erforderlich muss der *Symbology Identifier*, der aus dem Präfix und dem Wert *m* zusammengesetzt wird, manuell dem dekodierten String vorangesetzt werden (normalerweise aber nur falls $m > 1$). (Für Data Matrix ECC 200 und Aztec Code Symbole, müssen die Werte 10, 11 und 12 in A, B und C umgewandelt werden)
  Zu erkennen sind Symbole mit ECI-Codes (und damit verdoppelten Backslashs) an den folgenden Werten: ECC 200: 4, 5, 6, 10, 11 und 12, QR-Code: 2, 4 und 6, Aztec Code: 3, 4, 5, 9, 10 und 11, PDF417: 1, DotCode: 3, 4 und 5. Auf den Identifier-Präfix ']d' für Datamatrix bzw. ']Q' für QR-Code bzw. ']z' für Aztec Code bzw. ']L' für PDF417 bzw. ']J ' für DotCode wird an dieser Stelle verzichtet. Gegebenenfalls muss der *Symbology Identifier* bestehend aus dem Präfix und Wert *m* dem eigentlichen Ausgabedatenstrom in der Anwendung manuell vorangestellt werden.
  GS1 Symbologien haben die folgenden Identifier: ']d2' für GS1 DataMatrix, ']Q3' für GS1 QR Code , ']z1' für GS1 Aztec Code und ']J1' für GS1 DotCode. Dementsprechend liefert *'symbology_ident'* die Werte 2, 3, bzw. 1 zurück.

- Datamatrix ECC 200 und DotCode:

'reader_programming': Enthält das Symbol ein Reader Programming Zeichen (welches signalisiert, dass dieses Symbol eine Nachricht enthält um das Lesesystem zu programmieren) gib dieser Parameter *'yes'* zurück. Andernfalls gibt dieser Parameter *'no'* zurück. Das Reader-Programming-Zeichen ist nicht Bestandteil der dekodierten Nachricht.

- Datamatrix ECC 200:

  'slant': Slant innerhalb des Datamatrix-Symbols im Bogenmaß. Das ist die Differenz zwischen dem Winkel des Finder-Patterns und dem rechten Winkel.

  'finder_pattern_tolerance': Algorithmus, mit dem das Symbol gefunden wurde. Für *'low'* wurde das Symbol mit dem Algorithmus gefunden, der nur eine geringe Toleranz gegenüber einem gestörten oder teilweise fehlenden Finder-Pattern aufweist. Für *'high'* wurde das Symbol mit dem Algorithmus gefunden, der diesbezüglich eine hohe Toleranz aufweist. Zu Beachten ist, dass es bei der Verwendung beider Algorithmen vorkommen kann, dass auch Symbole mit ungestörtem Finder-Pattern vom Algorithmus mit hoher Toleranz gefunden werden.

  'contrast_tolerance': Dieser Parameter gibt Auskunft darüber, welcher Algorithmus zur Korrektur lokaler Kontrastvariationen verwendet wurde. Für *'high'* wurde das Symbol mit einem Algorithmus gefunden, der auch starke lokale Kontrastvariationen korrigieren kann. Für *'low'* wurde das Symbol mit dem schnelleren Algorithmus gefunden, der allerdings etwas weniger robust im Fall lokaler Kontrastvariationen ist.

  'module_grid': Konnte das Symbol gelesen werden, gibt dieser Parameter Auskunft darüber, ob bei der Bestimmung der Modulpositionen ein variables Gitter verwendet wurde (*'variable'*) oder nicht (*'fixed'*). Anderenfalls wird entweder die beim letzten Dekodierversuch verwendete Methode oder, falls der Kandidat bereits vor dem Dekodieren verworfen wurde, die Vorgabe aus dem Modell zurückgeliefert.

- QR-Code:

  'version': Die mit der Symbolgröße korrespondierende Versionsnummer (Version 1 = 21 $\times$ 21, Version 2 = 25 $\times$ 25, ..., Version 40 = 177 $\times$ 177).

  'symbol_size': Ermittelte Symbolgröße in Modulen.

  'model_type': Typ des QR-Code-Modells. Unterstützt werden das ältere QR-Code-Modell 1 und das neue Modell 2.

  'error_correction_level': Für Kandidaten, die als QR-Code identifiziert werden konnten, wird als erstes versucht, die im Symbol kodierten Formatinformationen zu lesen. Diese enthalten unter anderem den Grad der Fehlerkorrektur, der bei der Kodierung verwendet wurde (*'error_correction_level'* $\in$ *'L'* (Low), *'M'* (Medium), *'Q'* (Quartile), *'H'* (High)]).

  'mask_pattern_ref': Außerdem enthalten die Formatinformationen den Code des Maskierungsmusters, das über die eigentlichen Symboldaten gelegt wurde ($0 \leq$ *'mask_pattern_ref'* ≤ 3).

- Micro-QR-Codes:

  'version': Die mit der Symbolgröße korrespondierende Versionsnummer (version M1 = 11 $\times$ 11, version M2 = 13 $\times$ 13, version M3 = 15 $\times$ 15, version M4 = 17 $\times$ 17).

  'symbol_size': Ermittelte Symbolgröße in Modulen.

  'error_correction_level': Für Kandidaten, die als Micro-QR-Code identifiziert werden konnten, wird als erstes versucht, die im Symbol kodierten Formatinformationen zu lesen. Diese enthalten unter anderem den Grad der Fehlerkorrektur, der bei der Kodierung verwendet wurde (*'error_correction_level'* $\in$ [*'N'* (None), *'L'* (Low), *'M'* (Medium), *'Q'* (Quartile)]).

  'mask_pattern_ref': Außerdem enthalten die Formatinformationen den Code des Maskierungsmusters, das über die eigentlichen Symboldaten gelegt wurde ($0 \leq$ *'mask_pattern_ref'* ≤ 3).

- PDF417:

  'module_aspect': Seitenverhältnis der Module; dies entspricht dem Verhältnis von *'module_height'* zu *'module_width'*.

  'error_correction_level': Für Kandidaten, die als PDF417 identifiziert werden konnten, wird als erstes versucht, die im Symbol kodierten Formatinformationen in den Row-Indicators zu lesen. Diese enthalten auch den Grad der Fehlerkorrektur, der bei der Kodierung verwendet wurde (*'error_correction_level'* $\in$ [*0*, *8*]).

'macro_exist': Symbole, die Bestandteil einer Gruppe von zusammengehörigen Symbolen sind, werden als *„Macro PDF417“* Symbole bezeichnet. Diese Symbole enthalten neben den herkömmlichen Daten einen Control Block, in dem weitere Informationen gespeichert sind. Für Macro-Symbole liefert *'macro_exist'* den Wert *1*, für herkömmliche Symbole den Wert *0* zurück.

'macro_segment_index': Liefert den Index des Symbols in der Gruppe. Diese Information ist für Macro-Symbole obligatorisch.

'macro_file_id': Liefert den Gruppenidentifikator in Form eines Strings. Diese Information ist für Macro-Symbole obligatorisch.

'macro_segment_count': Liefert die Anzahl zusammengehöriger Symbole. Diese Information ist für Macro-Symbole optional.

'macro_time_stamp': Liefert den Zeitstempel der Quelldatei als Anzahl der seit 1970:01:01:00:00:00 GMT vergangener Sekunden in Form eines Strings. Diese Information ist für Macro-Symbole optional.

'macro_checksum': Liefert die CRC-Checksumme für die gesamte Quelldatei unter Verwendung des CCITT-16 Polynoms. Diese Information ist für Macro-Symbole optional.

'macro_last_symbol': Liefert *1*, falls das Symbol das letzte innerhalb der Gruppe zusammengehöriger Symbole ist, ansonsten *0*. Diese Information ist für Macro-Symbole optional.

'quality_isoiec15415_float_grades': Ein Tupel mit der gleichen Bewertung der Druckqualität wie *'quality_isoiec15415'*. Beim PDF417 werden einzelne Grade nach dem internationalen Standard ISO/IEC 15416:2016 berechnet. Diese werden gemäß dem Standard mit einer Genauigkeit von einer Dezimalstelle zurück gegeben.

- Aztec Codes:

  'format': Format des dekodierten Symbols: *'compact'*, *'full_range'* oder *'rune'*.

  'symbol_size': Größe des Symbols in Modulen.

  'layer_num': In der Mode Message kodierte Anzahl der Layer.

  'codeword_num': In der Mode Message kodierte Anzahl Codewörter.

Ergebnisse, die sich auf einen konkreten Kandidaten beziehen - Tupelwert Ergebnisse:

Ergebnisse, die sich aus **mehreren Werten** zusammensetzen und daher immer nur einzeln und für genau **einen Kandidaten** abgefragt werden können. Bei diesen Ergebnissen ist zu beachten, dass für Kandidaten, deren Bearbeitung vorzeitig abgebrochen wurde (vgl. Bearbeitungsstatus *'status'*), nicht alle Ergebnisse sinnvolle Werte liefern:

- Alle 2D-Datacode-Modelle:

  'bin_module_data': Binäre Symboldaten, die den einzelnen Modulen zeilenweise entsprechen – ein Wert von 0 bedeutet, dass das entsprechende Modul als Hintergrund interpretiert wurde, bei einem Wert von 100 handelt es sich um ein Vordergrundmodul. Werte zwischen 0 und 100 können sowohl als Vorder- als auch als Hintergrund interpretiert werden. Die Modellpersistenz muss für dieses Ergebnis auf den Wert *1* gesetzt sein (siehe `set_data_code_2d_param`). Für DotCodes spielt die Modellpersistenz keine Rolle.

  'raw_coded_data': Unkorrigierte Rohdaten, die über ein 2D-Datacode-Typ spezifisches Mapping aus den binären Moduldaten bestimmt werden. Diese Rohdaten sind möglicherweise noch fehlerbehaftet und enthalten auch die Daten, die für die Fehlerkorrektur verwendet werden.

  'corr_coded_data': Nach Anwendung der Fehlerkorrektur enthalten die Daten nur noch die korrigierten Nutzdaten ohne die Fehlerkorrekturwörter. Die Daten sind aber immer noch entsprechend des verwendeten 2D-Datacode-Typs kodiert.

  'decoded_data': Tupel mit den dekodierten Daten im ASCII-Code (siehe auch Tupel / String-Operationen) oder – beim QR-Code und Micro QR-Code – als JIS8 oder Shift JIS Zeichen oder – beim QR-Code – als GB2312 Zeichen. Diese Daten entsprechen der im Symbol kodierten Zeichenkette.

- Alle 2D-Datacode-Modelle bis auf DotCode:

  'quality_isoiec15415': Tupel mit der Bewertung der Druckqualität gemäß dem internationalen Standard ISO/IEC 15415:2011. Das erste Element des Tupels enthält immer die Gesamtqualität des Symbols; Die Länge des Tupels und die Bedeutung der restlichen Elemente hängen vom speziellen Typ des Datacodes ab. Gemäß dem Standard werden die einzelnen Grade mit einem Wert von 0 bis 4 bewertet, wobei 0 für den niedrigsten und 4 für den höchsten Grad steht. Es ist zu beachten, dass, obwohl sich diese Implementierung streng an den Standard hält, die Bewertung der Grade von der Datacode-Dekodierungsprozedur

abhängt. Dadurch können die Bewertungsergebnisse leicht von den Ergebnissen anderer Datacode-Leser (von anderen Anbietern) abweichen.
Zur Verwendung dieser Werte ist ein gutes Verständnis der zugrundeliegenden Algorithmen notwendig. Wir empfehlen daher, zusätzlich zu dieser Dokumentation den Standard ISO/IEC 15415:2011 zu lesen.
Für eine exakte Qualitätsbewertung empfehlen wir Bilder zu verwenden, die frei von Störungen wie Defokussierung, Rauschen, Überbelichtung oder inhomogene Beleuchtung sind. Außerdem ist eine effektive Auflösung von mindestens zehn Pixeln in Höhe und Breite pro Modul empfohlen (siehe ISO/IEC 15415:2011, Kapitel 7.3.3). Die Qualitätswerte werden nur für Symbole berechnet, deren Symbolregion samt Ruhezone komplett im Bild liegen, ansonsten wird -1 als Wert für die Qualitätsnoten zurückgegeben.
Für die 2D-Datacodes *ECC 200*, *QR Code*, *Micro QR Codes* und *Aztec Code* gilt folgende Spezifikation zur Bewertung der Druckqualität, zusammengefasst in einem Tupel mit zwölf Elementen: (*overall quality, contrast, modulation, fixed pattern damage, decode, axial nonuniformity, grid nonuniformity, unused error correction, reflectance margin, print growth, contrast uniformity, aperture*). Für *QR Code* und *Micro QR Codes* wird zusätzlich noch die *format information* und *versions information* zurückgegeben.
Die Definitionen einzelner Grade sind wie folgt: Der Grad *overall quality* ist der Mindestwert aller anderen Grade. Der Grad *contrast* ist die Differenz zwischen maximalem und minimalem Reflexionswert des Grauwertprofils. Stärkerer Kontrast ergibt einen besseren Grad. Die *modulation* bewertet die Amplitude zwischen den Datacodemodulen. Höhere Amplituden bedingen, dass dunkle von hellen Modulen zuverlässiger voneinander unterschieden werden können, und dass dieser Grad höher bewertet wird. Es ist noch zu beachten, dass die Bewertung von *modulation* von der Fehlerkorrekturkapazität (*error correction capacity*) des Symbols abhängt. Das bedeutet, dass *modulation* bei Symbolen mit höherer Fehlerkorrekturkapazität langsamer abnimmt. Die *contrast uniformity* ist der geringste Zahlenwert für die Modulation im gesamten Code. Dieser Wert ist kein Grad, und kann daher reelle Werte annehmen. Er beeinflusst die Berechnung der *overall quality* nicht. Die *reflectance margin* bewertet, wie auch die *modulation*, die Amplitude zwischen den Datacodemodulen. Im Gegensatz zur *modulation* berücksichtigt die *reflectance margin*, ob ein Modul richtig klassifiziert werden konnte. Wie die *contrast uniformity* wird auch die *print growth* nach ISO/IEC 15415:2011 nicht zu den Graden dazugezählt und geht daher auch nicht in die Berechnung der *overall quality* ein. Stattdessen wird sie als zusätzliche Information ausgegeben, um einen Aufschluss darüber zu geben, ob die grafischen Strukturen aus denen der 2D-Datacode besteht nicht von ihrer nominellen Größe abweichen. Das bedeutet, dass durch die *print growth* getestet wird, in welchem Maße die dunklen und hellen Module ihre Modulgrenzen ausfüllen. Die Berechnung der *print growth* folgt dabei nicht den, im Standard ISO/IEC 15415:2011, spezifizierten Schema, sondern den ISO/IEC Standards des jeweiligen Datacodetyps. Das bedeutet, dass die Berechnung der *print growth* für *Aztec Codes* nach den Spezifikationen im ISO/IEC 24778:2008 durchgeführt wird. Für *ECC 200* gilt ANSI/AIM International Specification Data Matrix und für *QR Codes* sowie *Micro QR Codes* ISO/IEC 18004:2006. Für alle oben beschriebenen 2D-Datacodes wird die *print growth* in horizontale und vertikale Richtung berechnet. Der ausgegebene Grad entspricht dabei dem kleineren der beiden Grade.
Das Rahmenmuster bei ECC 200, QR Codes und Aztec Codes ist extrem wichtig für das Detektieren eines Symbols. Verschlechterung oder Störung des Rahmenmusters und der benachbarten Ruhezonen wird durch den Grad *fixed pattern damage* bewertet, welches auf deren Modulationswerten basierend. Der Grad *decode* besagt, ob der Code dekodiert werden konnte.
Der Wert für *decode* ist *4*, wenn der untersuchte Datacode entsprechend des Referenzdekodieralgorithmus des Standards gelesen werden konnte. Andernfalls ist er *0*. Es ist zu beachten, dass HALCONs Dekodieralgorithmus von dem des Referenzdekodieralgorithmus abweicht. So kann es oft sein, dass HALCON ein Symbol lesen kann, obwohl der Wert für Decode entsprechend dem Standard *0* ist.
Normalerweise haben Datacodesymbole quadratische Module, d.h. Höhe und Breite eines Moduls sind gleich. Ihr Verhältnis kann durch eine schräge Kameraansicht oder fehlerhaftes Erzeugen des Symbols auch ungleich werden. Diese Abweichung wird durch den Grad *axial nonuniformity* bewertet. Wenn das Symbol neben einer affinen auch einer perspektivischer Deformation unterliegt, wird das entsprechend im Grad *grid nonuniformity* bewertet. Da Datacodes über eine gewisse Redundanz verfügen, können Fehler in Modulen oder Codewörtern korrigiert werden. Die ungenutzte Fehlerkorrekturkapazität des untersuchten Symbols wird im Grad *unused error correction* berechnet. In gewisser Hinsicht bezeichnet dieser Grad die Robustheit des Dekodierungsprozesses. Es ist zu beachten, dass manche Codes mit Grad *unused error correction* von 0 trotzdem von `find_data_code_2d` dekodiert werden können. Das liegt daran, dass `find_data_code_2d` einen robusteren Dekodierungsalgorithmus implementiert als den Referenzdekodierungsalgorithmus, den der Standard vorschlägt.
Für *QR Codes* und *Micro QR Codes* werden zusätzlich noch die Module der *format information* und

version information ähnlich wie bei *fixed pattern damage* bewertet. Falls der Code keine *version information* enthält, wird *'N/A'* zurückgegeben.
Für den 2D-Stapelcode *PDF417* gilt folgende Spezifikation zur Bewertung der Druckqualität zusammengefasst in einem Tupel mit acht Elementen: (overall quality, start/stop pattern, codeword yield, unused error correction, modulation, decodability, defects, aperture). Die Definitionen einzelner Grade sind wie folgt: Der Grad *overall quality* ist der Mindestwert aller anderen Grade. Der PDF417 Datacode ist ein Stapelcode, der von laserbasierten Lesegeräten gelesen werden kann. Dementsprechend basiert das Verfahren für Druckqualitätsbewertung auf dem Verfahren für lineare Barcodes: Eine Reihe von Scanlinien wird über das Symbol gleichmäßig verteilt. Dann wird die Reflexionsqualität einzelner Scanlinien bewertet und zum Schluss in die obigen Grade aufsummiert. Mehr Informationen über die Reflexionsgrade sind im Standard ISO/IEC 15416:2016 zu finden. Der Grad *start/stop pattern* bezieht sich auf die Start- und Stoppmuster; der Grad bewertet ihre Reflexionseigenschaften und ob die Sequenzen von Strichen und Lücken korrekt sind. Der Grad *codeword yield* berechnet den relativen Anteil der richtig dekodierten Codewörter, die unter den Scans liegen. Die ungenutzte Fehlerkorrekturkapazität des behandelten Symbols wird im Grad *unused error correction* berechnet. Ähnlich wie bei den Matrixdatacodes bezeichnet der Grad *modulation* die Amplitude zwischen den Symbolmodulen. Der Grad *decodability* misst die Abweichung der Modulbreiten von ihren im Symbologiestandard vorgegebenen Werten. Zum Schluss bezeichnet der Grad *defects* Unregelmäßigkeiten im Scanprofil innerhalb einzelner Symbolmodule, deren Vorhandensein mit niedrigerem Grad berechnet wird.
Die *aperture* gibt die Größe der synthetischen Apertur relativ zur Modulgröße des Symbols an. Diese Apertur wird verwendet, um das Referenz-Grauwert-Bild für die Qualitätsbewertung zu erzeugen. Sie ist definiert in ISO/IEC 15415:2011, Kapitel 7.3.3.
Für die Berechnung der Qualitätsmerkmale ist Voraussetzung, dass der Parameter *'persistence'* (siehe `set_data_code_2d_param`) auf einen Wert größer gleich *0* gesetzt ist.

'quality_isoiec15415_values': Tupel aus Rohmesswerten für alle Grade, die *'direkt messbar'* sind. Das sind jene Grade, deren Definition im Standard ISO/IEC 15415:2011 einer *'direkten Ableitung'* der Geometrie- oder Reflexionseigenschaften des abgebildeten Symbols entsprechen, oder Grade, welche das Ergebnis eines *'direkten Zählens'* sind.
Die Elemente des Tupels werden in derselben Reihenfolge zurückgegeben wie die Elemente einer entsprechenden *'quality_isoiec15415'*-Abfrage. Für alle Grade, welche von dieser Liste ausgeschlossen sind, liefert die *'quality_isoiec15415_values'*-Abfrage den Wert *'N/A'* zurück. Für die 2D-Datacodes *ECC 200*, *QR Code* und *Aztec Code* sind die Grade *overall quality*, *modulation*, *fixed pattern damage*, *decode* und *reflectance margin* ausgeschlossen. Für den 2D-Stapelcode *PDF417* sind die Grade *overall quality*, *start/stop pattern*, *modulation*, *decodability*, *defects* ausgeschlossen.
Obwohl die Grade *modulation* (für *ECC 200* und *QR Code*) und *modulation*, *decodability*, *defects* (für *PDF417*) grundsätzlich die Reflexionseigenschaften des Symbols bewerten, werden sie in den nach dem Standard vorgeschriebenen Berechnungen durch ein komplexes Schema berechnet, welches die Dekodierungsroutine und den Fehlerkorrekturmechanismus der Symbologie einbezieht. Zur Folge haben diese Grade keine direkten darunterliegenden Rohmesswerte, die berichtet werden können.
Alle Werte bis auf die *print growth* sind zwischen *0.0* und *1.0* normalisiert. Somit entspricht z.B. ein Wert von *0.75* für den Grad *contrast* einem Grauwert von *191.25* (für BYTE Bilder). Für die *print growth* können Werte zwischen ca. *-0.9* und ca. *0.9* erreicht werden. Ein negativer Wert entspricht dabei einer *print shrinkage*, während bei positiven Werten eine *print growth* vorliegt. Der ausgegebene Wert beschreibt den prozentualen Anteil um welchen ein dunkles Module zu wenig (*print shrinkage*) bzw. zu viel (*print growth*) von seiner nominellen Modulgröße einnimmt. Ein Wert von *-0.50* bedeutet beispielsweise, dass ein dunkles Modul 50% kleiner als die nominelle Modulgröße ist. Falls die *print growth* nicht berechnet werden konnte, wird der Wert auf *'N/A'* gesetzt.
Für die Berechnung der Qualitätsmerkmale ist Voraussetzung, dass der Parameter *'persistence'* (siehe `set_data_code_2d_param`) auf einen Wert größer gleich *0* gesetzt ist.

- Data Matrix ECC 200:

'quality_isoiec15415_intermediate': Tupel mit Zwischenergebnissen der Qualitätsbewertung von ECC 200 Codes nach den internationalen Standards ISO/IEC 15415:2011 und ISO/IEC 16022:2006. Zur Verwendung dieser Werte ist ein gutes Verständnis der zugrundeliegenden Algorithmen notwendig. Wir empfehlen daher dringend, zusätzlich zu dieser Dokumentation die Standards ISO/IEC 15415:2011 und ISO/IEC 16022:2006 zu lesen. Die Namen der verfügbaren Zwischenergebnisse können mit *'quality_isoiec15415_intermediate_labels'* abgefragt werden.
Die Bedeutung der einzelnen Grade (*'quality_isoiec15415_intermediate'*) und Werte (*'quality_isoiec15415_intermediate_values'*) werden im folgenden beschrieben. Sie entsprechen jeweils dem

minimalen Grad und Wert des inspizierten Codes.

- *'Rmin'* und *'Rmax'*: Minimum und Maximum der Grauwerte in der Bewertungsregion des Symbols (siehe Kapitel 7.6, ISO/IEC 15415:2011). Zu diesem Wert gibt es keinen entsprechenden Grad.
- *'L1'* und *'L2'*: Grade der vertikalen und horizontalen Teile des äußeren L des Rahmenmusters (siehe Kapitel M.1.2, ISO/IEC 16022:2006). Die entsprechende Anzahl an fehlerhaften Modulen wird als Wert zurückgegeben.
- *'QZL1'* und *'QZL2'*: Grade der vertikalen und horizontalen Teile der L1 und L2 benachbarten Ruhezonen (siehe Kapitel M.1.2, ISO/IEC 16022:2006). Die entsprechende Anzahl an fehlerhaften Modulen wird als Wert zurückgegeben.
- *'Transition Ratio'*: Grad für den „transition ratio test" (Übergangsraten-Test) der in Kapitel M.1.3 b), ISO/IEC 16022:2006, beschrieben ist. Die Übergangsrate TR wird als Werte zurückgegeben. Sie gibt das Verhältnis zwischen der Anzahl von Übergängen auf dem „clock track" (alternierendes Muster) und der Anzahl von Übergängen in der „solid area" (Ruhezone oder Teil eines L) an.
- *'Clock track regularity'*: Grad für die Regelmäßigkeit des alternierenden Musters („clock track regularity test"). Dieser Test ist in Kapitel M.1.3 e), ISO/IEC 16022:2006 beschrieben. Wenn irgendeine Folge von fünf benachbarten Punkten mehr als zwei Modul-Fehler enthält, ist dieser Grad 0, sonst ist er 4. Für diesen Grad gibt es keinen entsprechenden Wert.
- *'Clock track damage'*: Grad für die Beschädigung des alternierenden Musters („clock track damage") (siehe Kapitel M.1.3 f), ISO/IEC 16022:2006). Die entsprechende Anzahl an fehlerhaften Modulen wird als Wert zurückgegeben.
- *'Solid fix pattern'*: Grad für die Beschädigung des zum alternierenden Muster gehörenden Teil der Ruhezone oder eines L („solid fixed pattern") (siehe Kapitel M.1.3 f), ISO/IEC 16022:2006). Die entsprechende Anzahl an fehlerhaften Modulen wird als Wert zurückgegeben.
- *'Clock track and adjacent solid pattern'*: Grad für das gesamte alternierende Muster inklusive der dazugehörenden Ruhezonen, sowie der inneren L Muster (siehe Kapitel M.1.3 k), ISO/IEC 16022: 2006). Für diesen Grad gibt es keinen entsprechenden Wert.
- *'Average grade'*: Durchschnittlicher Grad für das gesamte Rahmenmuster. Dieser Grad basiert auf den Ergebnissen der Bewertungen für L1, L2, QZL1, QZL2, und dem Ergebnis für *'Clock track and adjacent solid pattern'* (siehe Kapitel M.1.4, ISO/IEC 16022:2006). Für diesen Grad gibt es keinen entsprechenden Wert.

Der Grad für *fixed pattern damage* für das Symbol ist das Minimum der Grade für *'L1'*, *'L2'*, *'QZL1'*, *'QZL2'*, *'Clock track and adjacent solid pattern'* und *'Average grade'*.

'quality_semi_t10_values': Tupel mit der Bewertung der Druckqualität gemäß dem internationalen Standard SEMI T10-0701.

Für eine exakte Qualitätsbewertung empfehlen wir Bilder zu verwenden, die frei von Störungen wie Defokussierung, Rauschen, Überbelichtung oder inhomogene Beleuchtung sind. Außerdem ist eine effektive Auflösung von mindestens zehn Pixeln in Höhe und Breite pro Modul empfohlen. Die Qualitätswerte werden nur für Symbole berechnet, die komplett im Bild liegen, ansonsten wird -1 als Wert für die Qualitätsnoten zurückgegeben.

Das zurückgegebene Wertetupel umfasst die folgenden 21 Werte: [P1 Row, P1 Column, P2 Row, P2 Column, P3 Row, P3 Column, P4 Row, P4 Column, Rows, Columns, Symbol Contrast, Symbol Contrast SNR, Horizontal Mark Growth, Vertical Mark Growth, Data Matrix Cell Width, Data Matrix Cell Height, Horizontal Mark Misplacement, Vertical Mark Misplacement, Cell Defects, Finder Pattern Defects, Unused Error Correction]. Die *Unused Error Correction* wird für jeden Reed-Solomon Block einzeln zurück gegeben. Die tatsächliche Tupellänge hängt somit von der Anzahl der Reed-Solomon Blöcke ab.

Die Definitionen einzelner Werte sind wie folgt: Die ersten acht Einträge enthalten die Koordinaten der vier Eckpunkte des Codes. Diese sind dem Standard entsprechend gegen den Uhrzeigersinn (bei nicht gespiegeltem Code), beginnend beim vertikalen Finder-Pattern, angeordnet. *Rows* und *Columns* geben die Zeilen- und Spaltenanzahl des Codes an.

Der Wert *Symbol Contrast* gibt den prozentualen Kontrast zwischen den als hell bzw. dunkel klassifizierten Pixeln an. *Symbol Contrast SNR* ist das entsprechende Signal-Rausch-Verhältnis. Sollte der Signal-Rausch-Verhältnis einen unendlichen Wert haben, wird *'N/A'* zurückgegeben.

Die Werte für *Horizontal Mark Growth* und *Vertical Mark Growth* stellen die prozentuale Breite bzw. Höhe der markierten Module im Verhältnis zur Summe der Breite bzw. Höhe eines markierten Moduls und eines Zwischenraums dar. Ein Wert von 50% ist optimal. Falls Teile des alternierenden Patterns verdeckt sind, wird *'N/A'* zurückgegeben.

Die Werte für *Data Matrix Cell Width* und *Data Matrix Cell Height* werden aus den vier Eckpunkten und der Anzahl von Spalten bzw. Zeilen des Codes berechnet und geben die durchschnittlichen Modulbreiten bzw. -höhen an.
Die Werte für *Horizontal Mark Misplacement* und *Vertical Mark Misplacement* geben die Abweichungen der Markierungen des alternierenden Patterns von idealen Mittelpunkten in horizontaler und vertikaler Richtung an. Die Werte werden prozentual im Verhältnis zu *Data Matrix Cell Width* und *Data Matrix Cell Height* angegeben. Falls Teile des alternierenden Patterns verdeckt sind, wird *'N/A'* zurückgegeben.
Der Wert für *Cell Defects* gibt den Prozentsatz der Modulpixel an, die falsch klassifiziert wurden.
Der Wert für *Finder Pattern Defects* gibt den Prozentsatz der Findern pattern Pixel an, die nicht als Markierungen klassifiziert werden würden.
Der Wert für *Unused Error Correction* gibt die ungenutzte Fehlerkorrekturkapazität der einzelnen Reed-Solomon Blöcke an.
Für die Berechnung der Qualitätsmerkmale ist Voraussetzung, dass der Parameter *'persistence'* (siehe `set_data_code_2d_param`) auf einen Wert größer gleich *0* gesetzt ist.

'quality_semi_t10_labels': Gibt die Namen der Elemente des Qualitätstupels, das von `get_data_code_2d_results` mit Parameter *'quality_semi_t10_values'* ausgegeben wird, zurück. Besitzt der ECC 200 Code mehr als ein Reed-Solomon Block, wird der Namenstupel um diese Werte erweitert.

'quality_isoiec15415_reflectance_margin_module_grades': Tupel mit der Reflectance-Margin-Bewertung der einzelnen Module gemäß dem Standard ISO/IEC 15415:2011 Chapter 7.8.4.3. Dies beinhaltet die Module des Finder-Patterns und der 4 zum Symbol anliegenden Ruhezonen (2 horizontal und 2 vertikal). Ist die Symbolgröße $Rows \cdot Columns$, dann ist die Tupellänge $(Rows + 2) \cdot (Columns + 2)$, wegen der 4 Ruhezonen. Das Tupel ist zeilenweise sortiert, wobei die erste Zeile die obere Ruhezone und die letzte die untere Ruhezone ist (oder QZL2 wie in ISO/IEC 16022:2006 Annex M.1.2, Figure M.1 beschrieben).
Die berechneten Modulgrade sind im Intervall von 0 bis 4. Für unbenutzte Daten-Module wird kein Grad berechnet. In diesem Fall ist der entsprechende Tupelwert -1.

'quality_isoiec15415_rows': Tupel mit der Zeilenposition der Symbol-Module, für die die Modulgrade mit dem Parameter *'quality_isoiec15415_reflectance_margin_module_grades'* berechnet werden.
Das zurückgegebene Tupel hat die gleiche Anzahl und Reihenfolge von Elementen wie das zurückgegebene Tupel für den Parameter *'quality_isoiec15415_reflectance_margin_module_grades'*.

'quality_isoiec15415_cols': Tupel mit der Spaltenposition der Symbol-Module, für die die Modulgrade mit dem Parameter *'quality_isoiec15415_reflectance_margin_module_grades'* berechnet werden.
Das zurückgegebene Tupel hat die gleiche Anzahl und Reihenfolge von Elementen wie das zurückgegebene Tupel für den Parameter *'quality_isoiec15415_reflectance_margin_module_grades'*.

'quality_isoiec_tr_29158_reflectance_margin_module_grades': Ähnlich wie *'quality_isoiec15415_reflectance_margin_module_grades'*. Die Werte werden allerdings nach dem Standard ISO/IEC TR 29158 (AIM DPM-1-2006) berechnet.

'quality_aimdpm_1_2006_reflectance_margin_module_grades': Siehe den Eintrag zu *'quality_isoiec_tr_29158_reflectance_margin_module_grades'*.

'quality_isoiec_tr_29158_rows' **und** *'quality_aimdpm_1_2006_rows'*: Ähnlich wie *'quality_isoiec15415_rows'* allerdings bezogen auf *'quality_isoiec_tr_29158_reflectance_margin_module_grades'* und *'quality_aimdpm_1_2006_reflectance_margin_module_grades'*.

'quality_isoiec_tr_29158_cols' **und** *'quality_aimdpm_1_2006_cols'*: Ähnlich wie *'quality_isoiec15415_cols'* allerdings bezogen auf *'quality_isoiec_tr_29158_reflectance_margin_module_grades'* und *'quality_aimdpm_1_2006_reflectance_margin_module_grades'*.

- Data Matrix ECC 200, Aztec, QR Code und Micro QR Code:

  'quality_isoiec_tr_29158' **und** *'quality_aimdpm_1_2006'*: Tupel mit der Bewertung der Druckqualität gemäß dem internationalen Standard ISO/IEC TR 29158. Dieser Standard wurde zuvor von Automatic Identification Manufacturers als AIM DPM-1-2006 formuliert. ISO/IEC TR 29158 ist eine Erweiterung des Standards ISO/IEC 15415:2011, indem er bestimmte Vorgaben für die Grauwerteigenschaften des Datacodebildes setzt und dadurch die Vergleichbarkeit der Qualitätsbewertungen unterschiedlicher Datacodeverifikatoren (von unterschiedlichen Anbietern) erhöht.
  Zur Verwendung dieser Werte ist ein gutes Verständnis der zugrundeliegenden Algorithmen notwendig. Wir empfehlen daher, zusätzlich zu dieser Dokumentation die Standards ISO/IEC TR 29158 (AIM DPM-1-2006) und ISO/IEC 15415:2011 zu lesen.

Es ist sehr wichtig zu wissen, dass eine ISO/IEC TR 29158 (AIM DPM-1-2006) konforme Druckqualitätsbewertung nur mithilfe eines interaktiven Bildeinzugs durchgeführt wird! Bilder von Datacode Symbolen dürfen nicht berücksichtigt werden, wenn keine Informationen über die Konfiguration der Kamera-Beleuchtung zur Verfügung stehen (siehe den Standard ISO/IEC TR 29158 (AIM DPM-1-2006) für einige vorgeschriebene Kamerabeleuchtungskonfigurationen und allgemeine Konfigurationsprinzipien, die für den Standard von Bedeutung sind). Wir weisen darauf hin, dass, obwohl sich diese Implementierung streng an den Standard hält, die Bewertung der Grade von der Datacode Dekodierungsprozedur abhängt. Deswegen ist eine mithilfe dieses Operators erworbene Qualitätsbewertung nur für HALCON's Datacode-Leser maßgeblich.

Beim Ausführen von `get_data_code_2d_results` mit Parameter *'quality_isoiec_tr_29158'* oder *'quality_aimdpm_1_2006'* wird angenommen, dass das behandelte Bild die beiden unten beschriebenen Kriterien erfüllt. Die zurückgegebene Druckqualitätsbewertung umfasst die folgenden 13 Grade: [overall quality, cell contrast, cell modulation, fixed pattern damage, decode, axial nonuniformity, grid nonuniformity, unused error correction, mean light, reflectance margin, print growth, contrast uniformity, aperture]. Die ersten acht sowie die letzten vier Grade haben dieselbe Bedeutung wie im ISO/IEC 15415:2011 Standard. Es gibt nur zwei Ausnahmen: *contrast* und *modulation* wurden zu *cell contrast* bzw. *cell modulation* umbenannt, um auf die Unterschiede zwischen den entsprechenden Formeln beider Standards hinzuweisen. Der Wert *mean light* ist im ISO/IEC TR 29158 (AIM DPM-1-2006) nicht als Grad definiert. Er ist eine Bewertung der Qualität des behandelten Bildes und wird als Mittelgrauwert der Zentren der hellen Datacodesymbolmodulen definiert. *mean light* kann einen Wert im Bereich 0.0 bis 1.0 haben, was 0% bis bzw. 100% des maximalen Grauwerts (255 bei Byte Bildern) entspricht. Für *QR Code* und *Micro QR Codes* wird zusätzlich noch die *format information* und *versions information* zurückgegeben.

Der Standard ist im wesentlichen die Definition einer Prozedur für die Einstellung der Kameraparameter (der so genannte *camera system response* - SR): *Belichtungszeit*, *Gain*, und/oder andere Einstellungen, die spezifisch für den Kamerabeleuchtungsaufbau des Benutzers sind. Das Ziel ist es, ein Bild des 2D-Datacodesymbols einzuziehen, das optimale Grauwerteigenschaften hat. In Bildverarbeitungskontext, welcher im Mittelpunkt des Standards steht, ist das die Voraussetzung für die erhöhte Vergleichbarkeit der Bewertungsergebnisse unterschiedlicher Verifikatoren. Laut Standard hat ein Bild optimale Grauwerteigenschaften, wenn sein *mean light* zwischen 70% und 86% liegt (bzw. 0.70 und 0.86). Um diese Bedingung für Bilder eines Datacodesymbols mit niedrigem materiellen Kontrast zu erfüllen, kann es häufig zu sehr hohen SR-Werten führen. Deswegen definiert der Standard eine zusätzliche (Kalibrierungs-)Prozedur, deren Aufgabe es ist, den normalen (so-genannten *kalibrierten*) system response (SRcal) für Vergleichszwecke zu berechnen. Das wird gemacht, indem die Bilder eines Objektes mit bekannten Reflexionseigenschaften (z.B. *NIST traceable calibrated conformance test card*) bewertet werden. Letztendlich sind die **zwei Bildeinzugskriterien** wie folgt formuliert: **Zum einen**, muss das Bewertungsbild einen *mean light* Wert im Bereich von 70% bis 86% (0.70 und bzw. 0.86) haben; **zum anderen**, muss das Bild mit Kameraeinstellungen (SR) eingezogen worden sein, welche die Bedingung SR/SRcal ≤ 16 erfüllen. Falls kein Bild eines Datacodes, das diese Kriterien erfüllt, eingezogen werden kann, wird der Datacode mit Grad 0 bewertet.

Da weder die Einstellung der Kameraparameter (SR) noch die Überprüfung des SR-Kriteriums innerhalb eines HALCON Operators durchgeführt werden kann, ist eine ISO/IEC TR 29158 (AIM DPM-1-2006) konforme Druckqualitätsbewertung nur in einem geeigneten Programmaufbau möglich. Der Benutzer sei auf zwei mitgelieferte Beispielprogramme verwiesen (calibration_aimdpm_1_2006.hdev und print_quality_aimdpm_1_2006.hdev im Verzeichnis HALCONEXAMPLES/hdevelop/Identification/Data-Code). Dort sind beide Bildeinzugskriterien näher erklärt und demonstriert. Jeder, der den ISO/IEC TR 29158 (AIM DPM-1-2006) Standard für den eigenen Kamerabeleuchtungaufbau anpassen will, kann diese Programme als Startpunkt verwenden.

Die *aperture* gibt die Größe der synthetischen Apertur relativ zur Modulgröße des Symbols an. Diese Apertur wird verwendet um das Referenz-Grauwert-Bild für die Qualitätsbewertung zu erzeugen. Sie wird nach dem in ISO/IEC TR 29158 (AIM DPM-1-2006), Kapitel 7.3 beschriebenen Verfahren festgelegt.

Für die Berechnung der Qualitätsmerkmale ist Voraussetzung, dass der Parameter *'persistence'* (siehe `set_data_code_2d_param`) auf einen Wert größer gleich *0* gesetzt ist.

'quality_isoiec_tr_29158_values' **und** *'quality_aimdpm_1_2006_values'***:** Ähnlich wie *'quality_isoiec15415_values'* gibt diese Abfrage einen Tupel aus Rohmesswerten für alle Grade, die *'direkt messbar'* sind zurück. Die Werte werden allerdings nach dem Standard ISO/IEC TR 29158 (AIM DPM-1-2006) berechnet. Die Liste der zurückgegebenen sowie der ausgeschlossenen Werte ist dieselbe wie für die *'quality_isoiec15415_values'*-Abfrage. Der ISO/IEC TR 29158 (AIM DPM-1-

2006)-spezifische Hilfsgrad *mean light* wird von beiden Abfragen (*'quality_isoiec_tr_29158'* und *'quality_isoiec_tr_29158_values'* sowie *'quality_aimdpm_1_2006'* und *'quality_aimdpm_1_2006_values'*) in demselben Format zurückgegeben.
Für die Berechnung der Qualitätsmerkmale ist Voraussetzung, dass der Parameter *'persistence'* (siehe `set_data_code_2d_param`) auf einen Wert größer gleich *0* gesetzt ist.

- Data Matrix ECC 200:

  'quality_isoiec_tr_29158_intermediate' **und** *'quality_aimdpm_1_2006_intermediate'***:** Tupel mit Zwischenergebnissen der Qualitätsbewertung von ECC 200 Codes nach den internationalen Standards ISO/IEC TR 29158 (AIM DPM-1-2006), ISO/IEC 15415:2011 und ISO/IEC 16022:2006. Zur Verwendung dieser Werte ist ein gutes Verständnis der zugrundeliegenden Algorithmen notwendig. Wir empfehlen daher dringend zusätzlich zu dieser Dokumentation die Standards ISO/IEC TR 29158 (AIM DPM-1-2006), ISO/IEC 15415:2011 und ISO/IEC 16022:2006 zu lesen.
  Die Bedeutung der einzelnen Grade und Werte werden im folgenden beschrieben. Sie entsprechen jeweils dem minimalen Grad (*'quality_isoiec_tr_29158_intermediate'* oder *'quality_aimdpm_1_2006_intermediate'*) und Wert (*'quality_isoiec_tr_29158_intermediate_values'* oder *'quality_aimdpm_1_2006_intermediate_values'*) des inspizierten Codes. Die Namen der verfügbaren Zwischenergebnisse können mit *'quality_isoiec_tr_29158_intermediate_labels'* oder *'quality_aimdpm_1_2006_intermediate_labels'* abgefragt werden.
  - *'T2'*: Schwellenwert aus dem Histogram der Grauwerte des Referenz-Grauwert-Bildes (siehe Anhang A, ISO/IEC TR 29158 (AIM DPM-1-2006)).
  - *'MD'*: Der zu T2 gehörende Wert von MeanDark (siehe Anhang A, ISO/IEC TR 29158 (AIM DPM-1-2006)).
  - *'MLtarget'*: Der zu T2 gehörende Wert von MeanLight (siehe Anhang A, ISO/IEC TR 29158 (AIM DPM-1-2006)).
  - *'L1'* und *'L2'*: Grade der vertikalen und horizontalen Teile des äußeren L des Rahmenmusters (siehe Kapitel M.1.2, ISO/IEC 16022:2006). Die entsprechende Anzahl an fehlerhaften Modulen wird als Wert zurückgegeben.
  - *'QZL1'* und *'QZL2'*: Grade der vertikalen und horizontalen Teile der L1 und L2 benachbarten Ruhezonen (siehe Kapitel M.1.2, ISO/IEC 16022:2006). Die entsprechende Anzahl an fehlerhaften Modulen wird als Wert zurückgegeben.
  - *'Transition Ratio'*: Grad für den „transition ratio test“ (Übergangsraten-Test) der in Kapitel M.1.3 b), ISO/IEC 16022:2006, beschrieben ist. Die Übergangsrate TR wird als Werte zurückgegeben. Sie gibt das Verhältnis zwischen der Anzahl von Übergängen auf dem „clock track“ (alternierendes Muster) und der Anzahl von Übergängen in der „solid area“ (Ruhezone oder Teil eines L) an.
  - *'Clock track regularity'*: Grad für die Regelmäßigkeit des alternierenden Musters („clock track regularity test“). Dieser Test ist in Kapitel M.1.3 e), ISO/IEC 16022:2006 beschrieben. Wenn irgendeine Folge von fünf benachbarten Punkten mehr als zwei Modul-Fehler enthält, ist dieser Grad 0, sonst ist er 4. Für diesen Grad gibt es keinen entsprechenden Wert.
  - *'Clock track damage'*: Grad für die Beschädigung des alternierenden Musters („clock track damage“) (siehe Kapitel M.1.3 f), ISO/IEC 16022:2006). Die entsprechende Anzahl an fehlerhaften Modulen wird als Wert zurückgegeben.
  - *'Solid fix pattern'*: Grad für die Beschädigung des zum alternierenden Muster gehörenden Teil der Ruhezone oder eines L („solid fixed pattern“) (siehe Kapitel M.1.3 f), ISO/IEC 16022:2006). Die entsprechende Anzahl an fehlerhaften Modulen wird als Wert zurückgegeben.
  - *'Clock track and adjacent solid pattern'*: Grad für das gesamte alternierende Muster inklusive der dazugehörenden Ruhezonen, sowie der inneren L Muster (siehe Kapitel M.1.3 k), ISO/IEC 16022:2006). Für diesen Grad gibt es keinen entsprechenden Wert.
  - *'Average grade'*: Durchschnittlicher Grad für das gesamte Rahmenmuster. Dieser Grad basiert auf den Ergebnissen der Bewertungen für L1, L2, QZL1, QZL2, und dem Ergebnis für *'Clock track and adjacent solid pattern'* (siehe Kapitel M.1.4, ISO/IEC 16022:2006). Für diesen Grad gibt es keinen entsprechenden Wert.
  - *'Distributed damage grade'*: Durchschnittlicher Grad für das gesamte Rahmenmuster. Dieser Grad basiert auf den Ergebnissen der Bewertungen für L1, L2, QZL1, QZL2, und dem Ergebnis für *'Clock track and adjacent solid pattern'* (siehe Kapitel M.1.4, ISO/IEC 16022:2006, sowie Kapitel 9.4, ISO/IEC TR 29158 (AIM DPM-1-2006)). Für diesen Grad gibt es keinen entsprechenden Wert.

  Der Grad für *fixed pattern damage* für das Symbol ist das Minimum der Grade für *'L1'*, *'L2'*, *'QZL1'*, *'QZL2'*, *'Clock track and adjacent solid pattern'* und *'Distributed damage grade'*.

- Data Matrix ECC 200, QR Code, Aztec Code und DotCode:

  'structured_append': Ist das Symbol Bestandteil einer Gruppe von zusammengehörigen Symbolen (*„Structured Append“*), enthält das Ergebnistupel (1.) den Index des Symbols in der Gruppe, (2.) die Anzahl zusammengehöriger Symbole sowie (3.) eine Zahl als Gruppenidentifikator. Für Aztec Codes wird der Gruppenidentifikator als String zurückgegeben. Ist das Symbol kein Bestandteil einer Gruppe von zusammengehörigen Symbolen (*„Structured Append“*) wird für Aztec Codes ein leeres Tupel zurückgegeben.

- DotCode:

  Mit DotCodes können mehrere getrennte Nachrichten in einem einzigen DotCode Symbol enkodiert werden. Nach AIM International Symbology Specification für DotCode sollen solche getrennten Nachrichten behandelt werden als ob sie von verschiedenen Symbolen kommen. Die folgende Auflistung enthält diejenigen Ergebnisse, welche für jede einzelne der getrennten Nachrichten eines DotCodes dediziert abgefragt werden können. Dabei ist an die entsprechenden Parameternamen das Suffix *'_separated'* angehängt. Sie geben ein Tupel mit den getrennten Ergebnissen für jede einzelne getrennte Nachricht zurück. Wird der Parameter ohne angehängtes *'_separated'* abgefragt, wird nur das Ergebnis der ersten Nachricht zurückgegeben. Achtung: Kann eine der Nachrichten nicht dekodiert werden, werden keine Ergebnisse zurückgeliefert.

  'symbology_ident_separated': Siehe *'symbology_ident'* für Details.

  'decoded_string_separated': Siehe *'decoded_string'* für Details.

  'structured_append_separated': Siehe *'structured_append'* für Details

  'reader_programming_separated': Siehe *'reader_programming'* für Details.

  'decoded_data_separated': Siehe *'decoded_data'* für Details.

  'segment_num': Anzahl an getrennten Nachrichten.

Statusinformation

Anhand der Statusinformation, die für jeden Kandidaten abgefragt werden kann, lässt sich leicht nachvollziehen, an welcher Stelle ein Kandidat verworfen wurde. Die folgende Übersicht enthält die wichtigsten Statusmeldungen in der Reihenfolge, in der sie bei der Bearbeitung auftreten können:

- Datamatrix ECC 200:

  'aborted: too close to image border' Symbol ist zu nahe am Bildrand. Damit das Symbol gelesen werden kann, muss es vollständig im Bild liegen.

  'aborted adjusting: ...' Es ist nicht möglich, die genaue Position des Finder-Patterns in der Verarbeitungsebene zu bestimmen, da kein signifikanter Hell-Dunkel-Übergang gefunden werden kann.

  'aborted finder pattern: ...' Für eine der beiden Seiten des Finder-Patterns kann dessen Breite nicht ermittelt werden, da keine signifikanten Hell-Dunkel-Übergänge gefunden werden können.

  'aborted leg widths: widths of the finder pattern legs differ too much' Die für die beiden Seiten des Finder-Patterns ermittelten Breiten unterscheiden sich zu sehr.

  'aborted alternating side: ...' Bei der Symbolsuche wurden (in einer Richtung) zwar die beiden gegenüberliegenden Seiten des Symbols gefunden, es kann aber nicht bestimmt werden, welches die alternierende Seite des Finder-Patterns ist.

  'aborted border search: ...' Bei der Symbolsuche wurde (in einer Richtung) nur die durchgehende Seite des Finder-Patterns gefunden. Bei der weiteren Verarbeitung war es jedoch nicht möglich, die gegenüberliegende (alternierende) Seite zu finden.

  'aborted symbol: invalid size' Die aus den beiden alternierenden Seiten bestimmte Anzahl von Symbolzeilen und -spalten ergibt keinen gültigen ECC 200-Code.

  'aborted symbol: size does not fit strict model definition' Die ermittelte Symbolgröße ist zwar gültig, liegt jedoch nicht in dem vom Modell vorgegebenen Bereich.

  'aborted symbol: rectangular symbol does not fit strict mirror definition of model' Anhand der Symbolgröße wurde ein rechteckiger ECC 200-Code ermittelt. Auf Grund der im Modell angegebenen Restriktion hinsichtlich der Spiegelung des Symbols sind jedoch Zeilen und Spalten vertauscht, so dass das Symbol keinen gültigen ECC 200-Code enthält. Bei quadratischen Symbolen kann dieser Test natürlich nicht erfolgen, so dass sich hier eine falsche Angabe zur Spiegelung erst beim Lesen auswirkt und i.d.R. zum folgenden Fehler führt:

  'error correction failed' Die Fehlerkorrektur scheitert an zu vielen falsch interpretierten Modulen. Das deutet i.d.R. auf eine zu schlechte Bild- und/oder Druckqualität hin, kann aber auch an einer falschen Vorgabe hinsichtlich der Spiegelung des Symbols liegen.

'decoding failed: special decoding reader requested' Diese Fehlermeldung zeigt an, dass die kodierten Daten eine Nachricht zum Programmieren der Leseeinheit enthalten. Dies wird nicht unterstützt.

'decoding failed: inconsistent data' Die im Symbol kodierten Daten können nicht dekodiert werden, da sie nicht konsistent sind.

'decoding failed: invalid GS1 format' Die Daten im Symbol entsprechen nicht den allgemeinen GS1 Spezifikationen. Diese Fehlermeldung wird zurückgegeben, wenn versucht wird, ein gültiges Nicht-GS1-Symbol mit einem als GS1-Datacode erstellten Modell zu lesen.

'quiet zone is missing' Die Ruhezone dieses Kandidaten fehlt oder weist starke Defekte auf.

'invalid finder pattern or quiet zone check failed': Der Kandidat konnte dekodiert werden, aber wurde als möglicher Falschpositiv verworfen, da das Finder-Pattern und die Ruhezone nicht gefunden wurden. Diese zusätzliche Überprüfung wird für Symbole der Größe 10x10, 12x12 und 8x18 durchgeführt, da diese Codes über wenig Fehlerkorrektur verfügen und Falschpositive häufig in anderen Bildstrukturen gefunden werden.

- Datamatrix ECC 200 (falls eine hohe Finder-Pattern-Toleranz gewählt wurde, können zusätzlich folgende Meldungen auftreten):

  'aborted module grid search: no regular module grid found' Es konnte keine regelmäßige Gitterstruktur innerhalb der Kandidatenregion gefunden werden. Daher wird davon ausgegangen, dass es sich bei dem Kandidaten nicht um ein ECC 200 Symbol handelt.

  'aborted border search: overlap with previous result' Der aktuelle Kandidat überlappt mit einem zuvor erfolgreich dekodierten Symbol. Daher wird die Suche nach der Symbolgrenze im Bild abgebrochen.

  'aborted border search: no consistent border found' Es wurde keine konsistente Symbolgrenze innerhalb der vorgegebenen Grenzen gefunden, die zu einem erfolgreich dekodierten Symbol geführt hätte.

  'aborted reading: result is likely to be random' Der Kandidat konnte zwar dekodiert werden, da die Symbolgröße allerdings sehr klein und die Anzahl der Dekodierungsfehler relativ hoch war, handelt es sich höchstwahrscheinlich nicht um einen ECC 200. Daher wurde der Kandidat verworfen. Diese Heuristik gilt für Symbole der Größe 10×10 mit mindestens 1 Dekodierungsfehler und für Symbole der Größe 12×12 oder 8×18 mit mindestens 2 Dekodierungsfehlern.

- QR-Code:

  'aborted: too close to image border' Symbol ist zu nahe am Bildrand. Damit das Symbol gelesen werden kann, muss es vollständig im Bild liegen.

  'aborted adjusting: finder patterns' Es ist nicht möglich, anhand der Position-Detection-Patterns die genaue Position des Finder-Patterns in der Verarbeitungsebene zu bestimmen, da kein signifikanter Hell-Dunkel-Übergang gefunden werden kann.

  'aborted symbol: different number of rows and columns' Es konnte mit Hilfe des Finder-Patterns keine einheitliche Symbolgröße ermittelt werden. Beim Modelltyp 2 führt das nur bei kleinen Symbolen ($<$ Version 7 bzw. 45×45 Modulen) zu einem Fehler, da bei größeren Symbolen die Versionsinformation im Symbol kodiert und von dort gelesen wird.

  'aborted symbol: invalid size' Die anhand des Finder-Patterns ermittelte Größe ist zu klein bzw. (nur Modelltyp 1) zu groß.

  'decoding of version information failed' Beim Versuch, ein Modell-2-Symbol zu lesen, wurde mit Hilfe der Finder-Patterns ermittelt, dass die Symbolversion mindestens 7 ist ($\geq 45 \times 45$ Module). Es ist jedoch nicht möglich, die Versionsdaten aus den entsprechenden Feldern im Symbol zu dekodieren.

  'aborted symbol: size does not fit strict model definition' Die ermittelte Symbolgröße ist zwar gültig, liegt jedoch nicht in dem vom Modell vorgegebenen Bereich.

  'decoding of format information failed' Die im Symbol kodierten Formatinformationen (Maskierungsmuster und Grad der Fehlerkorrektur) konnten nicht gelesen werden.

  'error correction failed' Die Fehlerkorrektur scheitert an zu vielen falsch interpretierten Modulen. Das deutet i.d.R. auf eine zu schlechte Bild- und/oder Druckqualität hin, kann aber auch an einer falschen Vorgabe hinsichtlich der Spiegelung des Symbols liegen.

  'decoding failed: inconsistent data' Die im Symbol kodierten Daten können nicht dekodiert werden, da sie nicht konsistent sind.

  'decoding failed: invalid GS1 format' Die Daten im Symbol entsprechen nicht den allgemeinen GS1 Spezifikationen. Diese Fehlermeldung wird zurückgegeben, wenn versucht wird, ein gültiges Nicht-GS1-Symbol mit einem als GS1-Datacode erstellten Modell zu lesen.

- Aztec Code:

  'aborted: too close to image border' Symbol ist zu nahe am Bildrand oder teilweise außerhalb des Bildes.

  'aborted: mode message could not be decoded' Die Mode Message dieses Kandidaten konnte nicht dekodiert werden, da die Fehlerkorrektur an zu vielen falsch interpretierten Modulen gescheitert ist.

  'aborted: symbol expansion failed' Die Expansion des Kandidaten ist fehlgeschlagen. Die Ursache sind meist zu starke perspektivische Verzeichnungen.

  'aborted symbol: size does not fit strict model definition' Die ermittelte Symbolgröße ist zwar gültig, liegt jedoch nicht in dem vom Modell vorgegebenen Bereich. Das Modell muss angepasst werden (`set_data_code_2d_param`), damit dieser Kandidat auch dekodiert werden kann.

  'aborted symbol: rectangular symbol does not fit strict mirror definition of model' Dieser Kandidat ist gültig, erfüllt aber nicht die im Modell bezüglich Spiegelung (*'format'*) vorgegebenen Einschränkungen. Das Modell muss angepasst werden (`set_data_code_2d_param`), damit dieser Kandidat auch dekodiert werden kann.

  'aborted symbol: invalid format' Dieser Kandidat ist gültig, erfüllt aber nicht die im Modell bezüglich des Formats (*'mirrored'*) vorgegebenen Einschränkungen. Das Modell muss angepasst werden (`set_data_code_2d_param`), damit dieser Kandidat auch dekodiert werden kann.

  'error correction failed' Die Fehlerkorrektur scheitert an zu vielen falsch interpretierten Modulen. Das deutet i.d.R. auf eine zu schlechte Bild- und/oder Druckqualität hin, kann aber auch an einer falschen Vorgabe hinsichtlich der Spiegelung des Symbols liegen.

  'decoding failed: invalid GS1 format' Die Daten im Symbol entsprechen nicht den allgemeinen GS1 Spezifikationen. Diese Fehlermeldung wird zurückgegeben, wenn versucht wird, ein gültiges Nicht-GS1-Symbol mit einem als GS1-Datacode erstellten Modell zu lesen.

- PDF417:

  'aborted: too close to image border' Symbol ist zu nahe am Bildrand. Damit das Symbol gelesen werden kann, muss es vollständig im Bild liegen.

  'aborted symbol: size does not fit strict model definition' Die ermittelte Symbolgröße ist zwar gültig, liegt jedoch nicht in dem vom Modell vorgegebenen Bereich.

  'error correction failed' Die Fehlerkorrektur scheitert an zu vielen falsch interpretierten Modulen. Das deutet i.d.R. auf eine zu schlechte Bild- und/oder Druckqualität hin, kann aber auch an einer falschen Vorgabe hinsichtlich der Spiegelung des Symbols liegen.

  'decoding failed: special decoding reader requested' Diese Fehlermeldung zeigt an, dass die kodierten Daten eine Nachricht zum Programmieren der Leseeinheit enthalten. Dies wird nicht unterstützt.

  'decoding failed: inconsistent data' Die im Symbol kodierten Daten können nicht dekodiert werden, da sie nicht konsistent sind.

- DotCode:

  'aborted module grid search: no regular module grid found' Es konnte keine regelmäßige Gitterstruktur innerhalb der Kandidatenregion gefunden werden. Daher wird davon ausgegangen, dass es sich bei dem Kandidaten nicht um einen DotCode handelt.

  'aborted symbol: size does not fit strict model definition' Die ermittelte Symbolgröße ist zwar gültig, liegt jedoch nicht in dem vom Modell vorgegebenen Bereich.

  'error correction failed' Die Fehlerkorrektur scheitert an zu vielen falsch interpretierten Modulen. Das deutet i.d.R. auf eine zu schlechte Bild- und/oder Druckqualität hin, kann aber auch an einer falschen Vorgabe hinsichtlich der Spiegelung des Symbols liegen.

  'decoding failed: inconsistent data' Die im Symbol kodierten Daten können nicht dekodiert werden, da sie nicht konsistent sind.

  'unmasking: invalid mask type' Die ersten beiden Bits eines jeden DotCode enkodieren die sogenannte Maske des Symbols. Die Maske wird im Enkodierungsprozess benutzt um die graphische Robustheit des Symbols zu erhöhen. Mögliche Werte sind 0, 1, 2 und 3. Ist ein anderer Wert für die Maske enkodiert, wird der Kandidat mit diesem Status markiert.

  'quiet zone is missing' Die Ruhezone dieses Kandidaten fehlt oder weist starke Defekte auf.

  'decoding failed: invalid GS1 format' Die Daten im Symbol entsprechen nicht den allgemeinen GS1 Spezifikationen. Diese Fehlermeldung wird zurückgegeben, wenn versucht wird, ein gültiges Nicht-GS1-Symbol mit einem als GS1-Datacode erstellten Modell zu lesen.

Bei der Bearbeitung eines Kandidaten kann es vorkommen, dass mehrere Versuche unternommen werden, um ein Symbol zu lesen, wobei intern über die verschiedenen Annahmen, die das Modell zulässt, iteriert wird. Dabei bestimmt i.d.R. der letzte Versuch den Abbruchstatus. Erlaubt ein QR-Code-Modell beispielsweise sowohl Modelltyp 1 als auch Typ 2, wird zuerst versucht, das Symbol als Modell 2 zu interpretieren. Scheitert dies, wird von einem Modell-1-Symbol ausgegangen. Ist auch dieser Versuch nicht erfolgreich, enthält die Statusvariable den Fehlercode, mit dem die Interpretation als Modell-1-Symbol abgebrochen wurde. Um den ersten Fehlercode zu erhalten, muss vor dem Aufruf von `find_data_code_2d` das Modell entsprechend eingeschränkt werden.

Parameter

- ▷ **`DataCodeHandle`** (input_control) . . . . . datacode_2d ⇝ *handle*
  Handle des zu verwendenden 2D-Datacode-Modells.
- ▷ **`CandidateHandle`** (input_control) . . . . . integer ⇝ *string* / integer
  Handle des 2D-Datacode-Kandidaten. Entweder ein Integer (üblicherweise das ResultHandle von `find_data_code_2d`) oder ein String, der eine Gruppe von Kandidaten beschreibt.
  Defaultwert: 'all_candidates'
  Wertevorschläge: `CandidateHandle` ∈ {0, 1, 2, 'general', 'all_candidates', 'all_results', 'all_undecoded', 'all_aborted'}
- ▷ **`ResultNames`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Namen der Ergebnisse, die für einen oder eine Gruppe von 2D-Datacode-Kandidaten abgefragt werden sollen.
  Defaultwert: 'status'
  Werteliste: `ResultNames` ∈ {'min_search_level', 'max_search_level', 'pass_num', 'result_num', 'candidate_num', 'undecoded_num', 'aborted_num', 'handle', 'pass', 'status', 'search_level', 'process_level', 'polarity', 'module_gap', 'mirrored', 'model_type', 'symbol_rows', 'symbol_cols', 'symbol_size', 'version', 'module_height', 'module_width', 'small_modules_robustness', 'module_aspect', 'slant', 'contrast', 'module_grid', 'decoded_string', 'decoding_error', 'symbology_ident', 'mask_pattern_ref', 'finder_pattern_tolerance', 'contrast_tolerance', 'error_correction_level', 'bin_module_data', 'raw_coded_data', 'corr_coded_data', 'decoded_data', 'quality_isoiec15415', 'quality_isoiec15415_labels', 'quality_isoiec15415_values', 'quality_isoiec15415_float_grades', 'quality_isoiec15415_reflectance_margin_module_grades', 'quality_isoiec15415_rows', 'quality_isoiec15415_cols', 'quality_aimdpm_1_2006', 'quality_aimdpm_1_2006_labels', 'quality_aimdpm_1_2006_values', 'quality_aimdpm_1_2006_reflectance_margin_module_grades', 'quality_aimdpm_1_2006_rows', 'quality_aimdpm_1_2006_cols', 'quality_isoiec_tr_29158', 'quality_isoiec_tr_29158_labels', 'quality_isoiec_tr_29158_values', 'quality_isoiec_tr_29158_reflectance_margin_module_grades', 'quality_isoiec_tr_29158_rows', 'quality_isoiec_tr_29158_cols', 'quality_isoiec15415_intermediate', 'quality_isoiec15415_intermediate_labels', 'quality_isoiec15415_intermediate_values', 'quality_aimdpm_1_2006_intermediate', 'quality_aimdpm_1_2006_intermediate_labels', 'quality_aimdpm_1_2006_intermediate_values', 'quality_isoiec_tr_29158_intermediate', 'quality_isoiec_tr_29158_intermediate_labels', 'quality_isoiec_tr_29158_intermediate_values', 'quality_semi_t10_values', 'quality_semi_t10_labels', 'structured_append', 'macro_exist', 'macro_segment_index', 'macro_file_id', 'macro_segment_count', 'macro_time_stamp', 'macro_checksum', 'macro_last_symbol', 'reader_programming', 'aborted', 'orientation', 'layer_num', 'codeword_num', 'format', 'symbology_ident_separated', 'decoded_string_separated', 'structured_append_separated', 'reader_programming_separated', 'decoded_data_separated', 'segment_num'}
- ▷ **`ResultValues`** (output_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
  Liste mit den Ergebnissen.

Beispiel

```
* Example demonstrating how to access the results of the data code search.

* Create a model for reading Data matrix ECC 200 codes
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
* Read an image
read_image (Image, 'datacode/ecc200/ecc200_cpu_010')
* Read the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                   ResultHandles, DecodedDataStrings)
```

```
* Get the number of passes
get_data_code_2d_results (DataCodeHandle, 'general', 'pass_num', Passes)

* Get a tuple with the status of all candidates
get_data_code_2d_results (DataCodeHandle, 'all_candidates', 'status', \
                          AllStatus)
* Get the handles of all candidates that were detected as a symbol but
* could not be read
get_data_code_2d_results (DataCodeHandle, 'all_undecoded', 'handle', \
                          HandlesUndecoded)

* For every undecoded symbol, get the contour, the symbol size, and
* the binary module data
dev_set_color ('red')
for i := 0 to |HandlesUndecoded| - 1 by 1
    * Get the contour of the symbol
    get_data_code_2d_objects (SymbolXLD, DataCodeHandle, \
                              HandlesUndecoded[i], 'candidate_xld')
    * Get the symbol size
    get_data_code_2d_results (DataCodeHandle, HandlesUndecoded[i], \
                              ['symbol_rows','symbol_cols'], SymbolSize)
    * Get the binary module data (has to be queried exclusively)
    get_data_code_2d_results (DataCodeHandle, HandlesUndecoded[i], \
                              'bin_module_data', BinModuleData)
    * Stop for inspecting the data
    stop ()
endfor

* Clear the model
clear_data_code_2d_model (DataCodeHandle)
```

Ergebnis

`get_data_code_2d_results` liefert den Wert TRUE, falls die übergebenen Parameter korrekt und die angeforderten Ergebnisse für die letzte Symbolsuche verfügbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_data_code_2d`, `query_data_code_2d_params`

Nachfolger

`get_data_code_2d_objects`

Siehe auch

`query_data_code_2d_params`, `get_data_code_2d_objects`, `get_data_code_2d_param`, `set_data_code_2d_param`

Literatur

AIM Global Document: AIM DPM-1-2006: „Direct Part Mark (DPM) Quality Guideline“; Document Type: AIM Bar Code Guideline; Document Version: 1.0, 2006-12-12.

International Standard ISO/IEC 15415: „Information technology - Automatic identification and data capture techniques - Bar code symbol print quality test specification - Two-dimensional symbols“; Reference number ISO/IEC 15415:2011 (E); ISO/IEC 2011.

International Standard ISO/IEC 16022: „Information technology - Automatic identification and data capture techniques - Data Matrix bar code symbology specification“; Reference number ISO/IEC 16022:2006 (E); ISO/IEC 2006.

International Standard ISO/IEC 15438: „Information technology - Automatic identification and data capture techniques - PDF417 bar code symbology specification“; Reference number ISO/IEC 15438:2006 (E); ISO/IEC 2006.

International Standard ISO/IEC 18004: „Information technology - Automatic identification and data capture techniques - QR Code 2005 bar code symbology specification“; Reference number ISO/IEC 18004:2006 (E); ISO/IEC 2006.

International Standard ISO/IEC 24778: „Information technology - Automatic identification and data capture techniques - Aztec Code bar code symbology specification“; Reference number ISO/IEC 24778:2008 (E); ISO/IEC 2008.

Technical Report ISO/IEC TR 29158: „Information technology - Automatic identification and data capture techniques - Direct Part Mark (DPM) Quality Guideline“; Reference number ISO/IEC TR 29158:2011 (E); ISO/IEC 2011.

GS1 General Specifications; Version 12; Issue 1, Jan-2012; GS1.

Modul

Data Code

query_data_code_2d_params (: : DataCodeHandle,
QueryName : GenParamName)

Liefert für das übergebene 2D-Datacode-Modell die Namen der generischen Parameter bzw. Objekte, die in den anderen Datacode-Operatoren gesetzt oder abgefragt werden können.

Der Operator `query_data_code_2d_params` ermöglicht es, die Namen der generischen Parameter abzurufen, die in den Operatoren `set_data_code_2d_param`, `get_data_code_2d_param`, `find_data_code_2d`, `get_data_code_2d_results` und `get_data_code_2d_objects` verwendet werden können.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Die Auswahl der **Parametergruppe** erfolgt über den Parameter `QueryName`:

'get_model_params': `get_data_code_2d_param` - Parameter zum Abfragen des 2D-Datacode-Modells.

'set_model_params': `set_data_code_2d_param` - Parameter für das Setzen des 2D-Datacode-Modells.

'find_params': `find_data_code_2d` - Parameter beim Suchen und Lesen von 2D-Datacode-Symbolen.

'get_result_params': `get_data_code_2d_results` - Parameter zum Abfragen der alphanumerischen Ergebnisse bei der Suche nach 2D-Datacode-Symbolen.

'get_result_objects': `get_data_code_2d_objects` - Parameter zum Zugriff auf ikonische (Zwischen-) Ergebnisse bei der Suche nach 2D-Datacode-Symbolen.

'trained': `set_data_code_2d_param` - Parameter deren Werte trainiert wurden. Das nächste Training setzt nicht die Parameter neu, sondern erweitert den Suchraum.

Die erstellte Parameterliste hängt dabei ausschließlich von dem zugrundeliegenden Datacode-Typ ab und nicht vom gegenwärtigen Bearbeitungszustand des Modells oder einzelner Ergebnisse.

Parameter

▷ **DataCodeHandle** (input_control) ..... datacode_2d ⇝ *handle*
Handle des 2D-Datacode-Modells.

▷ **QueryName** (input_control) ..... attribute.name ⇝ *string*
Name der Parametergruppe.
Defaultwert: 'get_result_params'
Werteliste: `QueryName` ∈ {'get_model_params', 'set_model_params', 'find_params', 'get_result_params', 'get_result_objects', 'trained'}

▷ **GenParamName** (output_control) ..... attribute.value-array ⇝ *string*
Liste mit den Namen der jeweils gültigen generischen Parameter.

Beispiel

```
* This example demonstrates how the names of all available model parameters
* can be queried.  This is used to request first the settings of the
* untrained and then the settings of the trained model.

* Create a model for reading Data matrix ECC 200 codes
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
* Query all the names of the generic parameters that can be passed to the
* operator get_data_code_2d_param to request the model
query_data_code_2d_params (DataCodeHandle, 'get_model_params', GenParamName)
* Request the current settings of the (untrained) model
get_data_code_2d_param(DataCodeHandle, GenParamName, ModelParams)

* Read a training image
read_image (Image, 'datacode/ecc200/ecc200_cpu_007')
* train the model with the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all', \
                   ResultHandles, DecodedDataStrings)
* Request the current settings of the (now trained) model
get_data_code_2d_param(DataCodeHandle, GenParamName, TrainedModelParams)
* Create a tuple that demonstrates the changings
ModelAdaption := GenParamName + ':   ' + ModelParams + '  ->  ' + \
                 TrainedModelParams

* Clear the model
clear_data_code_2d_model (DataCodeHandle)
```

Ergebnis

`query_data_code_2d_params` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_data_code_2d_model`

Nachfolger

`get_data_code_2d_param`, `get_data_code_2d_results`, `get_data_code_2d_objects`

Modul

Data Code

read_data_code_2d_model (: : FileName : DataCodeHandle)

Liest ein 2D-Datacode-Modell aus einer Datei und legt ein neues Modell an.

Der Operator `read_data_code_2d_model` liest aus der Datei `FileName` ein 2D-Datacode-Modell ein und erzeugt aus den eingelesenen Daten ein neues Modell. In `DataCodeHandle` wird ein Handle auf das neue Modell zurückgeliefert. Die Datei `FileName` wird mit dem Operator `write_data_code_2d_model` erzeugt. Die Dateiendung für das 2D-Datacode-Modell ist in HALCON 'dcm'.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der einzulesenden 2D-Datacode-Datei.
Defaultwert: 'data_code_model.dcm'
Dateiendung: .dcm

▷ **DataCodeHandle** (output_control) . . . . . datacode_2d ⇝ *handle*
Handle für das neu eingelesene 2D-Datacode-Modell.

Beispiel

```
* This example demonstrates how a model that was saved in an earlier
* session can be used again by reading the model file

* Create a model by reading by reading a data code model file
read_data_code_2d_model ('ecc200_trained_model.dcm', DataCodeHandle)
* Read a symbol image
read_image (Image, 'datacode/ecc200/ecc200_cpu_010')
* Read the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                   ResultHandles, DecodedDataStrings)
* Clear the model
clear_data_code_2d_model (DataCodeHandle)
```

Ergebnis

read_data_code_2d_model liefert den Wert TRUE, falls eine 2D-Datacode-Datei mit dem angegebenen Namen gefunden wurde und korrekt eingelesen werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

find_data_code_2d

Alternativen

create_data_code_2d_model

Siehe auch

write_data_code_2d_model, clear_data_code_2d_model

Modul

Data Code

serialize_data_code_2d_model (: : DataCodeHandle : SerializedItemHandle)

Serialisieren eines 2D-Datacode-Modells.

serialize_data_code_2d_model serialisiert die Daten eines 2D-Datacode-Modells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_data_code_2d_model in eine Datei geschrieben werden. Das 2D-Datacode-Modell wird in dem Handle DataCodeHandle übergeben. Das serialisierte 2D-Datacode-Modell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_data_code_2d_model wieder deserialisiert werden.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Parameter

▷ **DataCodeHandle** (input_control) ..... datacode_2d ⇝ *handle*
Handle des 2D-Datacode-Modells.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_data_code_2d_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`set_data_code_2d_param`, `find_data_code_2d`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_data_code_2d_model`

Alternativen

`get_data_code_2d_param`

Siehe auch

`create_data_code_2d_model`, `set_data_code_2d_param`, `find_data_code_2d`

Modul

Data Code

set_data_code_2d_param (: : DataCodeHandle, GenParamName, GenParamValue :)

Setzen ausgewählter Parameter des 2D-Datacode-Modells.

Der Operator `set_data_code_2d_param` erlaubt es, zahlreiche Parameter, durch die ein 2D-Datacode beschrieben oder das Verhalten des Modells bei der Symbolsuche gesteuert wird, zu verändern, um so das 2D-Datacode-Modell an die eigenen Erfordernisse anzupassen. Alle Parameter können auch direkt beim Anlegen des Modells mit `create_data_code_2d_model` übergeben werden. Die aktuelle Konfiguration kann mit Hilfe von `get_data_code_2d_param` ausgelesen werden. Eine Liste mit den Namen aller Parameter, die für den gegebenen 2D-Datacode-Typ gesetzt werden können, liefert der Operator `query_data_code_2d_params`.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Es ist zu beachten, dass die Symbolstruktur von GS1 DataMatrix, GS1 QR Code, GS1 Aztec Code und GS1 DotCode der der Struktur von Data Matrix ECC 200, QR Code, Aztec Code, bzw. DotCode identisch ist. Deswegen gelten alle typspezifischen Parameter, Einstellungen und Vorgaben, die Data Matrix ECC 200, QR Code, Aztec Code oder DotCode betreffen, auch für ihre entsprechende GS1-Variante. Im Folgenden wird das ausführliche Auflisten typspezifischer Parameter einzelner GS1-Typen ausgelassen, um die Lesbarkeit der Dokumentation zu erleichtern. Stattdessen können die relevante Parameter eines GS1-Datacodetyps entweder von den Parametern von dessen nicht-GS1-Variante hergeleitet werden oder explizit durch `query_data_code_2d_params` mit dem Parameter *'set_model_params'* abgefragt werden.

Es folgt eine Übersicht über die verschiedenen generischen Parameter mit den jeweiligen Wertebereichen und Defaultwerten im Standardmodus (*'standard_recognition'*). Falls die Defaultwerte im erweiterten Modus (*'enhanced_recognition'*) von denen im Standardmodus abweichen, sind diese zusätzlich angegeben. Das gleiche gilt, falls die Defaultwerte im maximalen Modus *'maximum_recognition'* von denen im erweiterten Modus abweichen:

Grundeinstellung:

- Alle 2D-Datacode-Modelle:

'default_parameters': Setzt alle Modell-Parameter in einen der drei Grundzustände *standard*, *erweitert* oder *maximal* zurück (vgl. die folgende Übersicht sowie `create_data_code_2d_model`). Dabei wird neben den Parameterwerten auch der Trainingsstatus aller Parameter zurückgesetzt (vgl. `find_data_code_2d`).
Werteliste: 'standard_recognition', 'enhanced_recognition', 'maximum_recognition'
Default: 'standard_recognition'
Achtung: Soll dieser Parameter zusammen mit anderen Parametern gesetzt werden, muss *'default_parameters'* immer das erste Element in der übergebenen Liste sein.

'trained': Setzt für den übergebenen Parameter den Trainingsstatus auf trainiert. Alle gesetzten Parameter werden beim Training nicht mehr überschrieben. Falls nötig wird nur der Suchraum durch das Training erweitert. Für die möglichen Parameter, die für die einzelnen Codetypen trainiert werden können siehe `find_data_code_2d`.

Größe und Form des Symbols:

- Data Matrix ECC 200 (inklusive Finder-Pattern):

  Achtung: Beim Ändern der Größe oder der Form über den Operator `set_data_code_2d_param`, werden die Parameter der Reihe nach ausgewertet, sowie jeweils die Gültigkeit der aktuellen Einstellungen überprüft und ggf. angepasst. Daher kann sich die Reihenfolge der durchgeführten Änderungen auf die endgültigen Einstellungen auswirken.

  'symbol_cols_min': Mindestanzahl der Spalten des Symbols in Modulen.
  Wertebereich: [*10*, *12*, *14*, *...144*]
  Achtung: Das Setzen von *'symbol_cols_min'* führt dazu, dass *'symbol_shape'* auf *'any'* gesetzt wird. Wenn für die Größe des Symbols nur eine Symbol Form möglich ist, wird *'symbol_shape'* entsprechend auf *'rectangle'* oder *'square'* gesetzt.
  Default: 10

  'symbol_cols_max': Höchstanzahl der Spalten des Symbols in Modulen.
  Wertebereich: [*10*, *12*, *14*, *...144*]
  Achtung: Das Setzen von *'symbol_cols_max'* führt dazu, dass *'symbol_shape'* auf *'any'* gesetzt wird. Wenn für die Größe des Symbols nur eine Symbol Form möglich ist, wird *'symbol_shape'* entsprechend auf *'rectangle'* oder *'square'* gesetzt.
  Default: 144

  'symbol_rows_min': Mindestanzahl der Zeilen des Symbols in Modulen.
  Wertebereich: [*8*, *10*, *12*, *...144*]
  Achtung: Das Setzen von *'symbol_rows_min'* führt dazu, dass *'symbol_shape'* auf *'any'* gesetzt wird. Wenn für die Größe des Symbols nur eine Symbol Form möglich ist, wird *'symbol_shape'* entsprechend auf *'rectangle'* oder *'square'* gesetzt.
  Default: 8

  'symbol_rows_max': Höchstanzahl der Zeilen des Symbols in Modulen.
  Wertebereich: [*8*, *10*, *12*, *...144*]
  Achtung: Das Setzen von *'symbol_rows_max'* führt dazu, dass *'symbol_shape'* auf *'any'* gesetzt wird. Wenn für die Größe des Symbols nur eine Symbol Form möglich ist, wird *'symbol_shape'* entsprechend auf *'rectangle'* oder *'square'* gesetzt.
  Default: 144

  'symbol_cols': Setzt *'symbol_cols_min'* und *'symbol_cols_max'* auf angegebenen Wert.
  Wertebereich: [*10*, *12*, *14*, *...144*]
  Achtung: Das Setzen von *'symbol_cols'* führt dazu, dass *'symbol_shape'* auf *'any'* gesetzt wird. Wenn bei der Größe des Symbols nur eine Symbol Form möglich ist, wird *'symbol_shape'* entsprechend auf *'rectangle'* oder *'square'* gesetzt.

  'symbol_rows': Setzt *'symbol_rows_min'* und *'symbol_rows_max'* auf angegebenen Wert.
  Wertebereich: [*8*, *10*, *12*, *...144*]
  Achtung: Das Setzen von *'symbol_rows'* führt dazu, dass *'symbol_shape'* auf *'any'* gesetzt wird. Wenn bei der Größe des Symbols nur eine Symbol Form möglich ist, wird *'symbol_shape'* entsprechend auf *'rectangle'* oder *'square'* gesetzt.

  'symbol_shape': Mögliche Einschränkungen hinsichtlich der Form des Symbols (Rechteck und/oder Quadrat).
  Achtung: Mit dem Setzen der Symbolform ändern sich eventuell die vorher gemachten Einschränkungen hinsichtlich der Symbolgröße. Für *'square'* werden die minimalen Werte von *'symbol_cols_min'*

und *'symbol_rows_min'* und die maximalen Werte von *'symbol_cols_max'* und *'symbol_rows_max'* verwendet. Zusätzlich gelten die Beschränkungen gemäß folgender Tabelle:

symbol_shape	'any'	'rectangle'	'square'
'symbol_cols_min'	>= 10	>= 18	>= 10
'symbol_cols_max'	<= 144	<= 144	<= 144
'symbol_rows_min'	>= 8	>= 8	>= 10
'symbol_rows_max'	<= 144	<= 26	<= 144

Weiter wird für den Fall, dass *'symbol_cols_min'* größer ist als *'symbol_rows_max'*, *'symbol_shape'* auf *'rectangle'* gesetzt.
Seit HALCON 7.1.1 wird der gleiche Suchalgorithmus für beide Symbolformen verwendet wenn *'finder_pattern_tolerance'* auf *'low'* gesetzt ist. Für die Symbolsuche ist der Parameter in diesem Fall nicht mehr relevant. Ist *'finder_pattern_tolerance'* dagegen auf *'high'* oder *'any'* gesetzt, kann der Wert von *'symbol_shape'* die Symbolsuche deutlich beschleunigen wenn *'rectangle'* oder *'square'* gewählt wird.
Werteliste: 'rectangle', *'square'*, *'any'*

'symbol_size_min': Setzt *'symbol_cols_min'* und *'symbol_rows_min'* auf angegebenen Wert.

'symbol_size_max': Setzt *'symbol_cols_max'* und *'symbol_rows_max'* auf angegebenen Wert.

'symbol_size': Setzt *'symbol_cols_min'*, *'symbol_cols_max'*, *'symbol_rows_min'* und *'symbol_rows_max'* auf angegebenen Wert sowie *'symbol_shape'* auf *'square'*.

- QR-Code (inklusive Finder-Pattern):

  'model_type': Typ des QR-Code-Modells. Unterstützt werden das ältere QR-Code-Modell *'1'* und das neue Modell *'2'*. Mit Angabe von *'any'* oder *'0'* werden beide Modelle durchgegangen.
  Werteliste: 1, *2*, *'any'*, *0*
  Default: 'any'

  'version_min': Kleinste zu lesende Symbol-Version. Die Symbolversion korrespondiert direkt mit der Symbolgröße. Version 1 entspricht dabei einem Symbol mit 21×21 Modulen, Version 2: 25×25 Module, usw. bis Version 40: 177×177 Module. Die maximale Symbolgröße beim Modell 1 ist 73×73 bzw. Version 14.
  Wertebereich: [*1 ... 40*] (Modelltyp 1: [*1 ... 14*])
  Default: 1

  'version_max': Größte zu lesende Symbol-Version:
  Wertebereich: [*1 ... 40*] (Modelltyp 1: [*1 ... 14*])
  Default: 40

  'version': Setzt *'version_min'* und *'version_max'* auf denselben Wert.

  'symbol_size_min': Kleinste zu lesende Symbolgröße in Modulen. Dieser Parameter kann alternativ zu *'version_min'* verwendet werden:
  Wertebereich: [*21 ... 177*] (Modelltyp 1: [*21 ... 73*])
  Default: 21

  'symbol_size_max': Größte zu lesende Symbolgröße in Modulen. Dieser Parameter kann alternativ zu *'version_max'* verwendet werden:
  Wertebereich: [*21 ... 177*] (Modelltyp 1: [*21 ... 73*])
  Default: 177

  'symbol_size': Setzt *'symbol_size_min'* und *'symbol_size_max'* auf denselben Wert.

- PDF417:

  'symbol_cols_min': Mindestanzahl der Datenspalten des Symbols in Codewords, d.h. exklusive der beiden Codewords des Start/Stop-Pattern sowie der beiden Codewords der Row-Indicators.
  Wertebereich: [*1 ... 30*]
  Default: 1

  'symbol_cols_max': Höchstanzahl der Datenspalten des Symbols in Codewords, d.h. exklusive der beiden Codewords des Start/Stop-Pattern sowie der beiden Codewords der Row-Indicators.
  Wertebereich: [*1 ... 30*]
  Default: 20 (enhanced: *30*)

  'symbol_rows_min': Mindestanzahl der Zeilen des Symbols in Modulen.
  Wertebereich: [*3 ... 90*]
  Default: 5 (enhanced: *3*)

'symbol_rows_max': Höchstanzahl der Zeilen des Symbols in Modulen.
Wertebereich: [*3 ... 90*]
Default: 45 (enhanced: *90*)

'symbol_cols': Setzt *'symbol_cols_min'* und *'symbol_cols_max'* auf denselben Wert.

'symbol_rows': Setzt *'symbol_rows_min'* und *'symbol_rows_max'* auf denselben Wert.

- Aztec Code (inklusive Finder-Pattern):

  'format': Format des Aztec Codes: Durch Leerzeichen getrennte Liste mit den Werten *'compact'*, *'full_range'* oder *'rune'*.
  Default: 'compact full_range'

  'symbol_size_min': Kleinste zu lesende Symbolgröße in Modulen:
  [*11...151*]
  Default: 11

  'symbol_size_max': Größte zu lesende Symbolgröße in Modulen:
  [*11...151*]
  Default: 151

- DotCode: Ein nach AIM DotCode Revision 4.0-1 standardkonformer DotCode besteht aus einer geraden und einer ungeraden Dimension. In HALCON ist die Spaltenrichtung durch die Leserichtung definiert und bezieht sich immer auf die ungerade Dimension.

  'symbol_cols_min': Mindestanzahl der Spalten des Symbols in Modulen. Muss ungerade sein.
  Wertebereich: [*5 ... 999*]
  Default: 5

  'symbol_cols_max': Höchstanzahl der Spalten des Symbols in Modulen. Muss ungerade sein.
  Wertebereich: [*5 ... 999*]
  Default: 999

  'symbol_rows_min': Mindestanzahl der Zeilen des Symbols in Modulen. Muss gerade sein.
  Wertebereich: [*4 ... 998*]
  Default: 4

  'symbol_rows_max': Höchstanzahl der Zeilen des Symbols in Modulen. Muss gerade sein.
  Wertebereich: [*4 ... 998*]
  Default: 998

  'symbol_cols': Setzt *'symbol_cols_min'* und *'symbol_cols_max'* auf angegebenen Wert. Muss ungerade sein.
  Wertebereich: [*5 ... 999*]

  'symbol_rows': Setzt *'symbol_rows_min'* und *'symbol_rows_max'* auf angegebenen Wert. Muss gerade sein.
  Wertebereich: [*4 ... 998*]

Erscheinungsbild der Module im Bild:

- Alle 2D-Datacode-Modelle:

  'polarity': Beschreibt die Polarität des Symbols im Bild und bestimmt damit, ob das Symbol im Bild dunkel vor hellem Hintergrund oder hell vor dunklem Hintergrund erscheint.
  Werteliste: 'dark_on_light', *'light_on_dark'*, *'any'*.
  Default: 'dark_on_light' (enhanced: *'any'*)

  'mirrored': Angabe über eine mögliche Spiegelung des Symbols (entspricht einem Vertauschen von Spalten und Zeilen).
  Werteliste: 'no', *'yes'*, *'any'*
  Default: 'any'

- Alle 2D-Datacode-Modelle bis auf Data Matrix ECC 200 und DotCode:

  'contrast_min': Minimaler Kontrast zwischen dem Symbolvordergrund und dem Bildhintergrund (angegeben als Grauwertdifferenz); dieser Wert kann nicht allein aus der Differenz der Grauwerte von Vorder- und Hintergrund bestimmt werden, er korrespondiert vielmehr auch mit dem Anstieg der Modulkanten und damit mit der Schärfe des Bildes.
  Wertebereich: [*1 ... 255*]
  Default: 30 (enhanced: *10*)

- Alle 2D-Datacode-Modelle bis auf PDF417:

'module_size_min': Minimale Größe der Module im Bild in Pixel. Für eine optimale Leserate sollten Module mindestens eine Größe von 3-4 Pixeln haben.
Wertebereich: [*1* ... *100*]
Default: 6 (enhanced: *2*, maximum: *1*)
Für DotCode:
Default: 4 (enhanced, maximum: *2*)

'module_size_max': Maximale Größe der Module im Bild in Pixel.
Wertebereich: [*2* ... *100*]
Default: 20 (enhanced: *100*)
Für DotCode:
Default: 100

'module_size': Setzt *'module_size_min'* und *'module_size_max'* auf denselben Wert

- Alle 2D-Datacode-Modelle bis auf DotCode:

'small_modules_robustness': Robustheit der Dekodierung gegenüber Datacodes mit sehr kleiner Modulgröße. Wird der Parameter *'small_modules_robustness'* auf *'high'* gesetzt, so erhöht sich die Wahrscheinlichkeit, dass Datacodes mit sehr kleinen Modulen dekodiert werden können. Zusätzlich sollte in diesem Fall auch die minimale Modulgröße entsprechend angepasst werden, d.h. *'module_size_min'* bzw. *'module_width_min'* (PDF417) sollte auf die angenommene minimale Modulgröße bzw. Modulbreite gesetzt werden. Ist *'small_modules_robustness'* auf *'high'* gesetzt, so kann sich der interne Speicherbedarf von `find_data_code_2d` deutlich erhöhen. Folglich sollte *'small_modules_robustness'* im Normalfall auf *'low'* gesetzt werden. Ist *'small_modules_robustness'* auf *'high'* gesetzt, halbiert sich die maximal erlaubte Bildgröße (siehe `get_system` *'halcon_xl'*).
Werteliste: 'low', *'high'*
Default: 'low' (enhanced: *'low'*, maximum: *'high'*)

Die Lücken zwischen Modulen können mit Hilfe der *'module_gap*'* Parameter angegeben werden, wie im folgenden Paragraph erklärt:
Es kann in drei Stufen angegeben werden, ob benachbarte Vordergrundmodule direkt aneinanderstoßen oder ob es zwischen ihnen eine Lücke geben kann. Füllen die Vordergrundmodule den ihnen zustehenden Platz voll aus und stoßen sie direkt aneinander, kann für die folgenden Parameter *'no'* angegeben werden. *'small'* deutet auf einen sehr kleinen Zwischenraum, $< 10\%$ der Modulfläche, und *'big'* auf einen größeren Zwischenraum, $< 50\%$ der Modulfläche, zwischen benachbarten Modulen hin (*'big'* ist für DotCode nicht verfügbar). Diese letzten beiden Einstellungen sind u.U. auch dann sinnvoll, wenn die Vordergrundmodule zwar zusammenhängen aber (z.B. durch ein Überstrahlen der Aufnahme) den ihnen zustehenden Platz nicht voll ausfüllen und dünner erscheinen. Sind die Vordergrundmodule im Verhältnis zur Modulgröße nur als sehr kleine Punkte im Bild sichtbar, ist i.d.R. eine geeignete Vorverarbeitung zum Detektieren bzw. Vergrößern der Punkte notwendig (z.B. mit Hilfe von `gray_erosion_shape` oder `gray_dilation_shape`).

'module_gap_min': Minimale Lücke.
Werteliste: 'no', *'small'*, *'big'* (Nicht für DotCode)
Default: 'no'

'module_gap_max': Maximale Lücke.
Werteliste: 'no', *'small'*, *'big'*
Default (Data Matrix ECC 200): 'no' (enhanced: *'small'*, maximum: *'big'*)
Default (DotCode): 'no' (enhanced: *'small'*, maximum: *'small'*)
Default (Aztec Code, Micro QR Code, QR Code): 'small' (enhanced: *'big'*)

'module_gap': Setzt *'module_gap_min'* und *'module_gap_max'* auf denselben Wert.

- PDF417:

'module_width_min': Minimale Breite der Module im Bild in Pixel.
Wertebereich: [*1* ... *100*]
Default: 3 (enhanced: *2*, maximum: *1*)

'module_width_max': Maximale Breite der Module im Bild in Pixel.
Wertebereich: [*2* ... *100*]
Default: 15 (enhanced: *100*)

'module_width': Setzt *'module_width_min'* und *'module_width_max'* auf denselben Wert.

'module_aspect_min': Minimales Seitenverhältnis der Module im Bild (Höhe zu Breite).
Wertebereich: [*0.5* ... *20.0*]
Default: 1.0

'module_aspect_max': Maximales Seitenverhältnis der Module im Bild (Höhe zu Breite).
Wertebereich: [*0.5* ... *20.0*]
Default: 4.0 (enhanced: *10.0*)

'module_aspect': Setzt *'module_aspect_min'* und *'module_aspect_max'* auf denselben Wert.

- Data Matrix ECC 200:

'slant_max': Maximale Abweichung des Winkels im L-förmigen Finder-Pattern vom (idealerweise) rechten Winkel; die Angabe erfolgt im Bogenmaß und korrespondiert mit perspektivischen Verzerrungen, die beim Druck des Symbols oder bei der Bildaufnahme auftreten können.
Wertebereich: [*0.0* ... *0.5235*]
Default: 0.1745 = 10° (enhanced: *0.5235* = 30°)

'finder_pattern_tolerance': Toleranz der Suche gegenüber einem gestörten oder teilweise fehlenden Finder-Pattern. Das Finder-Pattern umschließt sowohl die L-förmige als auch die gegenüberliegende alternierende Seite. In Abhängigkeit von diesem Parameter werden verschiedene Algorithmen für die Suche in `find_data_code_2d` verwendet. In einem Fall (*'low'*) wird davon ausgegangen, dass das Finder-Pattern zum großen Teil vorhanden ist und kaum Störungen aufweist. Im anderen Fall (*'high'*) kann das Finder-Pattern gestört oder teilweise fehlend sein, ohne dass die Erkennung beeinträchtigt wird. Zu beachten ist allerdings, dass bei dieser Variante die Parameter für die Symbolsuche möglichst stark mit `set_data_code_2d_param` eingeschränkt werden sollten, da ansonsten eine erhöhte Rechenzeit für `find_data_code_2d` zu erwarten ist. Zu beachten ist außerdem, dass sich beide Algorithmen leicht in ihrer Robustheit voneinander unterscheiden. Dies kann dazu führen, dass es selbst bei Symbolen mit intaktem Finder-Pattern in Abhängigkeit von *'finder_pattern_tolerance'* zu unterschiedlichen Ergebnissen kommt. Wenn *'high'* ausgewählt wird, können z.B. nur Symbole mit festem Gitter gefunden werden (siehe unten), wodurch die Robustheit gegenüber perspektivischen Verzerrungen abnimmt. Im Fall *'any'* werden beide Algorithmen ausgeführt.
Werteliste: 'low', *'high'*, *'any'*
Default: 'low' (enhanced: *'low'*, maximum: *'any'*)

'contrast_tolerance': Toleranz der Suche beim Auftreten starker lokaler Kontrastvariationen (z.B. bei Reflexionen oder Überbelichtung). Abhängig von diesem Parameter wird zwischen zwei Algorithmen gewählt. Im Falle von *'high'* ist die Robustheit gegenüber starken lokalen Kontrastvariationen erhöht. Für *'low'* wird ein Algorithmus gewählt, der zwar etwas weniger robust bei lokalen Kontrastvariationen ist, dafür aber schneller. Unter normalen Umständen ist daher *'low'* zu bevorzugen. Setzt man den Parameter auf *'any'*, so werden beide Varianten ausgeführt.
Werteliste: 'low', *'high'*, *'any'*
Default: 'low' (enhanced: *'low'*, maximum: *'any'*)

'module_grid': Angabe darüber, ob die Größe der Module in einem gewissen Rahmen variieren darf oder nicht. In Abhängigkeit von diesem Parameter werden verschiedene Algorithmen für die Berechnung der Modulpositionen verwendet. In einem Fall (*'fixed'*) wird ein festes Gitter, bei dem die Abstände zwischen den Modulmittelpunkten alle gleich sind, verwendet. Im anderen Fall (*'variable'*) wird das Gitter an der alternierenden Seite des Finder-Patterns ausgerichtet. Bei *'any'* werden beide Varianten für das Gitter nacheinander ausprobiert. Zu beachten ist, dass der Wert von *'module_grid'* ignoriert wird, wenn *'finder_pattern_tolerance'* auf *'high'* gesetzt ist. In diesem Fall wird immer von einem festen Gitter ausgegangen.
Werteliste: 'fixed', *'variable'*, *'any'*
Default: 'fixed' (enhanced: *'any'*)

- QR Code:

'position_pattern_min': Anzahl von Position-Detection-Patterns, die im Bild gut sichtbar sein müssen, damit ein Symbolkandidat generiert wird.
Wertebereich: [*2*, *3*]
Default: 3 (enhanced: *2*)

- Aztec Code:

'finder_pattern_tolerance': Toleranz der Suche gegenüber einem gestörten oder teilweise fehlenden Finder-Pattern. Abhängig von diesem Parameter werden verschiedene Algorithmen für die Suche in `find_data_code_2d` verwendet. Für *'low'* wird angenommen, dass alle Ringe des Finder-Patterns extrahiert werden können. Für *'high'* wird angenommen, dass mindestens einer der Ringe des Finder-Patterns extrahiert werden kann. Da dadurch mehr Kandidaten gefunden werden, erhöht sich die Laufzeit der Suche für *'high'*.

Wertebereich: [*'low'*, *'high'*]
Default: 'low' (enhanced: *'high'*)

'additional_levels': Um die Robustheit des Aztec Code Lesers zu erhöhen, kann die Anzahl der Pyramidenebenen, die zusätzlich zu den durch die minimale und maximale Modulgröße bedingten Ebenen untersucht werden sollen, angegeben werden. Dies erhöht die Laufzeit der Suche, insbesondere wenn kein Code gefunden wird.
Wertebereich: [*0*... *2*]
Default: 0

Allgemeines Modellverhalten:

- Alle 2D-Datacode-Modelle:

  'persistence': Steuert, ob bestimmte Zwischenergebnisse bei der Symbolsuche mit `find_data_code_2d` nur temporär oder persistent im Modell gespeichert werden sollen. Bei der temporären Speicherung (default) ist der Speicherbedarf von `find_data_code_2d` deutlich geringer. Die persistente Speicherung erlaubt hingegen, nach der Symbolsuche auf einzelne Zwischenergebnisse zuzugreifen, etwa wenn untersucht werden soll, warum einzelne Symbole nicht gelesen werden konnten. Der Speicherbedarf von `find_data_code_2d` kann weiter reduziert werden, indem *'persistence'* auf *-1* gesetzt wird. In diesem Fall werden nur noch die Daten gespeichert, die nötig sind um die dekodierten Daten abzufragen. Das hat zur Folge, dass die Druckbildkontrolle mit dieser Einstellung für *'persistence'* nicht verwendet werden kann.
  Werteliste: 0 (temporär) oder *1* (persistent)
  Default: 0

  'discard_undecoded_candidates' Steuert, ob Kandidaten, die nicht erfolgreich dekodiert werden konnten, beim Modell gespeichert werden. Um den Speicherbedarf des Modells zu reduzieren, sollte dieser Parameter auf *'yes'* gesetzt werden. In diesem Fall ist zu beachten, dass mit `get_data_code_2d_objects` oder `get_data_code_2d_results` keine Information über Kandidaten abgefragt werden kann, die nicht erfolgreich dekodiert werden konnten.
  Werteliste: 'yes', *'no'*
  Default: 'no'

  'strict_model': Steuert das Verhalten von `find_data_code_2d` bei der Detektion von Symbolen, die hinsichtlich der Symbolgröße nicht den Modellvorgaben entsprechen. Diese können entweder verworfen werden (*'yes'*) oder trotz der abweichenden Größe als Ergebnis zurückgeliefert werden (*'no'*). Es ist zu beachten, dass für DotCode-Symbole die Modellvorgaben bzgl. der Modulgröße (*'module_size_min'* und *'module_size_max'*) auch dann nicht überprüft werden, wenn *'strict_model'* auf *'yes'* gesetzt ist.
  Werteliste: 'yes' (strikt), *'no'* (nicht strikt)
  Default: 'yes'

  'string_encoding': Setzt die erwartete Zeichenkodierung des im Symbol kodierten Strings. Es ist möglich UTF-8, Latin-1 oder die Zeichenkodierung der aktuellen Systemeinstellung zu wählen. Falls notwendig wird der String entsprechend umkodiert. Im Raw Modus wird der String nicht verändert.
  Werteliste: 'utf8', *'locale'*, *'latin1'*, *'raw'*
  Default: 'latin1'

  'timeout': Mit Hilfe dieses Parameters ist es möglich, dass `find_data_code_2d` nach einer definierten Zeit in ms abbricht. Dies ist vor allem dann nützlich, wenn maximale Zykluszeiten eingehalten werden müssen. Alle bis zu diesem Abbruch gewonnenen Ergebnisse sind anschließend verfügbar und können mit `get_data_code_2d_results` abgerufen werden. Werden Werte kleiner oder gleich Null übergeben, wird der Timeout deaktiviert (Default).
  Die zeitliche Genauigkeit des Timeouts liegt bei ca. 10 ms. Sie hängt von vielen Faktoren ab, u.a. von der Geschwindigkeit des Rechners, der Bildgröße und dem *'timer_mode'*, der über `set_system` gesetzt wurde.
  `find_data_code_2d` wirft keinen Fehler, falls ein Timeout auftritt. Um zu prüfen, ob `find_data_code_2d` unterbrochen wurde, kann der Parameter *'aborted'* mit `get_data_code_2d_results` abgefragt werden.
  Im Trainingsmodus von `find_data_code_2d` wird der Timeout ignoriert.
  Typische Werte: 'false', *-1*, *20* ... *100*
  Default: 'false'

  'abort': Mit dieser Option ist es möglich `find_data_code_2d` von einem anderen Thread aus abzubrechen. Wenn `set_data_code_2d_param` mit *'abort'* aufgerufen wird, wird einer Instanz von

find_data_code_2d mit dem Model DataCodeHandle in einem anderen Thread signalisiert abzubrechen. Falls es keine solche find_data_code_2d Instanz mit diesem Model gibt, passiert nichts.
Es ist möglich, dass der Operator find_data_code_2d nicht sofort abbricht. Er muss zunächst einen internen Abbruchpunkt erreichen, an dem ein sauberes Verlassen des Operators möglich ist. Dies hängt von Rahmenbedingungen wie Rechnerleistung ab und kann ungefähr bis zu 10ms benötigen. Im Trainingsmodus von find_data_code_2d wird der Parameter ignoriert.
Hinweis: Dies ist die einzige Aktion mit Datacode Handles, die von unterschiedlichen Threads genutzt werden kann, ohne dass zusätzliche Synchronisierung nötig ist.
Standardwert: 'true' (Der Wert wird nicht ausgewertet.)

- Alle 2D-Datacode-Modelle bis auf Aztec Code:

  'strict_quiet_zone': Steuert das Verhalten von find_data_code_2d bei der Detektion von Symbolen deren Ruhezone fehlt oder starke Defekte aufweist: *'yes'* (Symbole mit fehlender oder defekter Ruhezone werden verworfen) oder *'no'* (Symbole werden, wenn sie gelesen werden können, als Ergebnis zurückgeliefert). Der *'status'* von Ergebnissen, die verworfen wurden ist *'quiet zone is missing'*.
  Werteliste: 'yes' , *'no'*
  Default: 'no'

- Alle 2D-Datacode-Modelle bis auf DotCode:

  'quality_isoiec15415_aperture_size': Das Referenzbild, auf dem die Qualitätswerte Symbol Contrast, Modulation, Reflectance Margin und Fixed Pattern Damage bestimmt werden, wird durch eine künstliche Blendeneinstellung (kreisförmiger Mean-Filter) aus dem Originalbild gewonnen. Dieser Parameter bestimmt die Filtergröße als Anteil der Modulgröße. Laut Standard sollte dieser Wert je nach Anwendung zwischen 0.5 und 0.8 liegen (siehe ISO/IEC 15415:2011 Anhang D.2).
  Wertebereich: [*0 . . . 1*]
  Standardwert: 0.8

- Data Matrix ECC 200:

  'candidate_selection': Bestimmt die Auswahl der Kandidatenregionen, die verwendet werden, um Symbole im Bild zu finden. Wird dieser Parameter auf *'extensive'* gesetzt, so erhöht sich die Anzahl der generierten Kandidatenregionen und damit die Wahrscheinlichkeit einen Code zu finden. Eine größere Anzahl an Kandidaten kann jedoch auch zu höheren Laufzeiten führen. Wenn *'candidate_selection'* auf *'default'* gesetzt ist, werden weniger Kandidatenregionen verwendet. In den meisten Fällen ist die kleinere Anzahl von Kandidaten ausreichend.
  Werte: 'default', *'extensive'*
  Standardwert: 'default' (enhanced: *'extensive'*)

  'decoding_scheme': Steuert den Dekodierschritt bei Data Matrix ECC 200. Wenn dieser Parameter auf *'raw'* gesetzt ist, können auch Symbole gelesen werden, deren Aussehen und Fehlerkorrekturverfahren dem Standard ISO/IEC 16022:2006 entsprechen, die jedoch eine andere Enkodierung verwenden. Die dekodierten Daten entsprechen den fehlerkorrigierten Rohdaten, können mit dem Parameter *'decoded_data'* von get_data_code_2d_results abgefragt werden und müssen vom Benutzer weiterverarbeitet werden.
  Werte: 'default', *'raw'*
  Standardwert: 'default'

- DotCode:

  'candidate_selection': Bestimmt die Auswahl der Kandidatenregionen, die verwendet werden, um Symbole im Bild zu finden. Wird dieser Parameter auf *'extensive'* gesetzt, so erhöht sich die Anzahl der generierten Kandidatenregionen und damit die Wahrscheinlichkeit einen Code zu finden. Wird dieser Parameter auf *'all'* gesetzt, werden alle möglichen Kandidaten genutzt und nicht anhand ihrer Wahrscheinlichkeit ein DotCode zu sein gefiltert. Wenn *'candidate_selection'* auf *'default'* gesetzt ist, werden weniger Kandidatenregionen verwendet.
  Werte: 'default', *'extensive'*,*'all'*
  Standardwert: 'default' (enhanced: *'extensive'*, maximum: *'all'*)

  'max_allowed_error_correction': Bestimmt die maximal zulässige Fehlerkorrektur. Aufgrund der hohen Fehlerkorrekturkapazität, ist es möglich 'false positive' DotCode Kandidaten erfolgreich zu dekodieren. Besonders Kandidaten, die nur einen kleinen Teil eines echten DotCode-Symbols bedecken, werden aufgrund der Fehlerkorrektur eventuell erfolgreich dekodiert. Der Grund dafür ist, dass DotCode-Symbole so gut wie jede Größe annehmen können. Das macht es schwieriger falsche Kandidaten auszusortieren.

Um diesem Problem entgegenzuwirken, kann der Parameter *'max_allowed_error_correction'* verwendet werden, um anzugeben wie viel Prozent der Fehlerkorrekturkapazität genutzt werden dürfen. Standardmäßig ist der Wert auf *'1.0'*, also 100% gesetzt. Wird der Wert z.B. auf *'0.5'* herabgesetzt, werden nur noch die Kandidaten als erfolgreich dekodierte Ergebnisse zurückgegeben, die mit maximal 50% genutzter Fehlerkorrektur dekodiert werden konnten.
Wertebereich: [*0.0* . . . *1.0*]
Standardwert: '1.0'

Beim Setzen der Modellparameter sollte den folgenden Punkten **besondere Beachtung** geschenkt werden:

- Ist der Parameter *'strict_model'* auf *'yes'* eingestellt, was standardmäßig der Fall ist, werden Symbole, die nicht den mit *'symbol_rows*'*, *'symbol_cols*'*, *'symbol_size*'* oder *'version*'* gemachten Größenvorgaben entsprechen, nicht gelesen. Dieses Verhalten kann dazu ausgenutzt werden, um im Bild nach Symbolen einer bestimmten Größe zu suchen, während von dieser Größe abweichende Symbole bewusst ignoriert werden sollen. Das kann jedoch auch zu Problemen führen, insbesondere dann wenn aus einer Serie von Datacodes ein Symbol zum Einstellen des Modells als Beispiel herausgegriffen wurde, in der späteren Anwendung jedoch unterschiedlich große Datacodes vorkommen können, was in der Praxis relativ häufig der Fall ist.
- Auf die Laufzeit von `find_data_code_2d` wird sich vor allem das Setzen der folgenden Parameter auswirken, und zwar in erster Linie dann, wenn im Bild *nicht* die geforderte Anzahl an Datacodes gefunden werden kann: *'polarity'*, *'module_size_min'* (Data Matrix ECC 200 und QR Code) bzw. *'module_width_min'* und *'module_aspect_min'* (PDF417) und bei sehr kleinen minimalen Modulgrößen auch die Parameter *'module_gap_*'* (Data Matrix ECC 200, QR Code und Aztec Code), für QR Code auch *'position_pattern_min'*. Ist bei Data Matrix ECC 200 Symbolen *'finder_pattern_tolerance'* auf *'high'* oder *'any'* gesetzt, sollte die Symbolgröße so stark wie möglich mit *'symbol_size_min'*, *'symbol_size_max'*, *'symbol_size'* und *'symbol_shape'* eingeschränkt werden.

Parameter

▷ **DataCodeHandle** (input_control) . . . . . datacode_2d ⇝ *handle*
Handle des zu verwendenden 2D-Datacode-Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für den 2D-Datacode eingestellt werden sollen.
Defaultwert: 'polarity'
Werteliste: `GenParamName` ∈ {'default_parameters', 'strict_model', 'persistence', 'polarity', 'mirrored', 'contrast_min', 'candidate_selection', 'discard_undecoded_candidates', 'model_type', 'version', 'version_min', 'version_max', 'string_encoding', 'symbol_size', 'symbol_size_min', 'symbol_size_max', 'symbol_cols', 'symbol_cols_min', 'symbol_cols_max', 'symbol_rows', 'symbol_rows_min', 'symbol_rows_max', 'symbol_shape', 'module_size', 'module_size_min', 'module_size_max', 'small_modules_robustness', 'module_width_min', 'module_width_max', 'module_width', 'module_aspect_min', 'module_aspect_max', 'module_aspect', 'module_gap', 'module_gap_min', 'module_gap_max', 'slant_max', 'module_grid', 'position_pattern_min', 'strict_quiet_zone', 'abort', 'trained', 'timeout', 'finder_pattern_tolerance', 'format', 'additional_levels', 'contrast_tolerance', 'quality_isoiec15415_aperture_size', 'decoding_scheme', 'max_allowed_error_correction'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Werte der generischen Parameter, die für den 2D-Datacode eingestellt werden sollen.
Defaultwert: 'light_on_dark'
Wertevorschläge: `GenParamValue` ∈ {'standard_recognition', 'enhanced_recognition', 'maximum_recognition', 'yes', 'no', 'any', 'dark_on_light', 'light_on_dark', 'square', 'rectangle', 'small', 'big', 'fixed', 'variable', 'low', 'high', 'default', 'extensive', 'utf8', 'locale', 'raw', 0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 30, 50, 70, 90, 12, 14, 16, 18, 20, 22, 24, 26, 32, 36, 40, 44, 48, 52, 64, 72, 80, 88, 96, 104, 120, 132, 144}

Beispiel

```
* This examples shows how a model can be adapted to a specific symbol if
* the symbol parameters are known

* Create a model for reading Data matrix ECC 200 codes
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
* Restrict the model by setting the module size
set_data_code_2d_param (DataCodeHandle, \
```

```
                        ['module_size_min','module_size_max'], [4,7])
* Change the polarity setting of the model from 'dark_on_light' to
* 'light_on_dark'
set_data_code_2d_param (DataCodeHandle, 'polarity', \
                        'light_on_dark')

* Read an image
read_image (Image, 'datacode/ecc200/ecc200_cpu_010')
* Read the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                   ResultHandles, DecodedDataStrings)
* Clear the model
clear_data_code_2d_model (DataCodeHandle)
```

Ergebnis

`set_data_code_2d_param` liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DataCodeHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_data_code_2d_model`, `read_data_code_2d_model`

Nachfolger

`get_data_code_2d_param`, `find_data_code_2d`, `write_data_code_2d_model`

Alternativen

`read_data_code_2d_model`

Siehe auch

`query_data_code_2d_params`, `get_data_code_2d_param`, `get_data_code_2d_results`, `get_data_code_2d_objects`

Modul

Data Code

write_data_code_2d_model (: : DataCodeHandle, FileName :)

Schreibt ein 2D-Datacode-Modell in eine Datei.

Der Operator `write_data_code_2d_model` schreibt ein mit `create_data_code_2d_model` angelegtes 2D-Datacode-Modell in eine Datei mit dem Namen `FileName`. Das so gesicherte Modell kann mit `read_data_code_2d_model` später wieder eingelesen und wiederverwendet werden. Die Dateiendung für das 2D-Datacode-Modell ist in HALCON 'dcm'. Der Handle des zu schreibenden 2D-Datacode-Modells ist in `DataCodeHandle` zu übergeben.

Das Konzept des 2D-Datacode-Lesers ist bei der Einleitung zu Kapitel Identifizierung / Datacode beschrieben.

Parameter

▷ **DataCodeHandle** (input_control) ..... datacode_2d ⇝ *handle*
Handle des zu schreibenden 2D-Datacode-Modells.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der zu schreibenden 2D-Datacode-Datei.
Defaultwert: 'data_code_model.dcm'
Dateiendung: .dcm

Beispiel

```
* This example demonstrates how a trained model can be saved for
* a future session

* Create a model for reading Data matrix ECC 200 codes
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
* Read a training image
read_image (Image, 'datacode/ecc200/ecc200_cpu_007')
* Train the model with the symbol in the image
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all', \
                   ResultHandles, DecodedDataStrings)
* Write the model into a file
write_data_code_2d_model (DataCodeHandle, 'ecc200_trained_model.dcm')
* Clear the model
clear_data_code_2d_model (DataCodeHandle)
```

Ergebnis

write_data_code_2d_model liefert den Wert TRUE, falls der übergebene Handle gültig ist und das Modell in die Datei mit dem angegebenen Namen geschrieben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_data_code_2d_param, find_data_code_2d

Alternativen

get_data_code_2d_param

Siehe auch

create_data_code_2d_model, set_data_code_2d_param, find_data_code_2d

Modul

Data Code

12.3 Samplebasiert

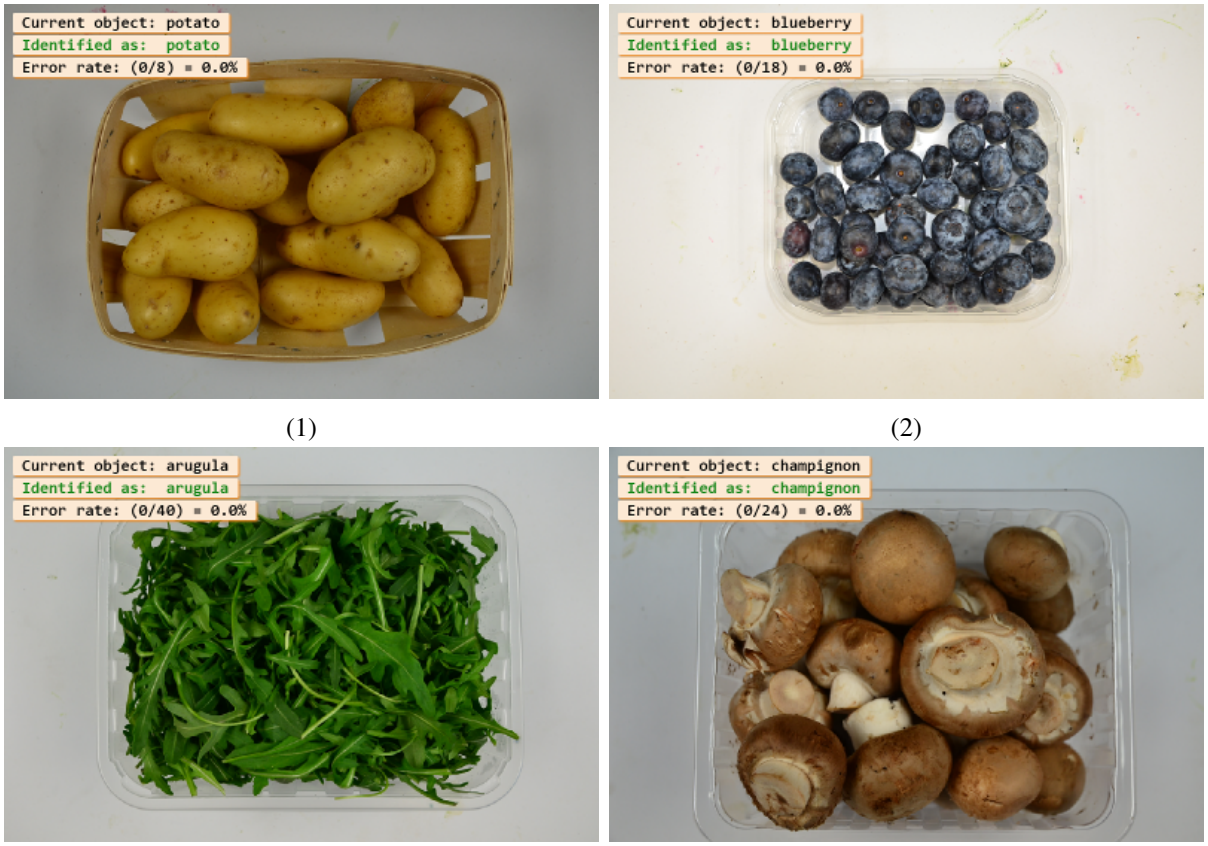
Dieses Kapitel beschreibt Operatoren für die samplebasierte Identifikation.

Konzept der samplebasierten Identifikation

Mit der samplebasierten Identifikation können eintrainierte Objekte anhand von charakteristischen Merkmalen wie Textur oder Farbe identifiziert werden. Dies ermöglicht es, Objekte zu identifizieren, die nicht mit einem Barcode oder einem Datacode markiert sind. Verglichen mit den Klassifikationsverfahren aus Kapitel Klassifikation, ist die Vorbereitung und das Training für die samplebasierte Identifikation sehr komfortabel, da hier kein komplexes Parametertuning nötig ist. Mit der samplebasierten Identifikation können sehr viele Objekte voneinander unterschieden werden. Die Identifikation ist robust gegenüber Rotation, Skalierung und Beleuchtungsänderungen sowie

gegenüber Verdeckungen, Störungen und moderaten perspektivischen Verzerrungen. Darüber hinaus ist die Identifikation robust gegenüber moderaten Deformationen des Objekts und sie erlaubt in gewissen Grenzen auch die Identifikation von Schüttgut. Andererseits bedeutet dies, dass Objekte nicht unterschieden werden können, die sich sehr ähnlich sind oder sich nur in einem der Punkte unterscheiden, gegenüber denen die samplebasierte Identifikation robust ist. Zu beachten ist, dass die samplebasierte Identifikation nur mit texturierten Objekten funktioniert.

Mit der samplebasierten Identifikation kann immer nur ein Objekt pro Bild identifiziert werden. Das bedeutet, dass jedes Suchbild, oder genauer gesagt die Domain jedes Suchbildes nur ein Objekt enthalten darf.



Samplebasierte Identifikation der Objekte 'potato', 'blueberry', 'arugula' und 'champignon'. Diese Bilder stammen aus dem Beispielprogramm `identify_vegetables.hdev`.

Im Folgenden werden die einzelnen Schritte, die zur Benutzung der samplebasierten Identifikation benötigt werden, kurz beschrieben.

Anlegen und Vorbereiten eines Sample-Identifikators: Zuerst muss ein Sample-Identifikator bereitgestellt werden. Dies erfolgt durch das Anlegen und Vorbereiten mit den Operatoren

- `create_sample_identifier`,
- `add_sample_identifier_preparation_data`, und
- `prepare_sample_identifier`.

Der Schritt der Vorbereitung ist notwendig, um die internen Datenstrukturen des Sample-Identifikators an die Art der Objekte anzupassen, die unterschieden werden sollen. Alternativ dazu kann ein bereits verfügbarer Sample-Identifikator, der mit `write_sample_identifier` in eine Datei geschrieben wurde, mit dem Operator `read_sample_identifier` von Datei eingelesen werden. Einen vorbereiteten Sample-Identifikator kann man sich in gewisser Weise wie eine Lagerhalle vorstellen, die für die Lagerung einer bestimmten Art von Produkten ausgelegt ist.

Trainieren des Sample-Identifikators: Im Anschluss an die Vorbereitung muss der Sample-Identifikator mit Samples der zu unterscheidenden Objekte trainiert werden. Dies erfolgt mit den Operatoren

- add_sample_identifier_training_data und
- train_sample_identifier

Es ist jederzeit möglich, den Sample-Identifikator nachzutrainieren. Hierfür können Samples aus dem Sample-Identifikator mit dem Operator remove_sample_identifier_training_data gelöscht werden und es können neue Samples mit dem Operator add_sample_identifier_training_data hinzugefügt werden. Sofern sich die Art der zu unterscheidenden Objekte nicht zu stark ändert, ist es nicht notwendig, die Vorbereitung des Sample-Identifikator erneut durchzuführen. Das Training entspricht - um im obigen Bild zu bleiben - dem Auffüllen der Lagerhalle.

Anwenden des Sample-Identifikators zum Identifizieren von Objekten: Nun kann der trainierte Sample-Identifikator verwendet werden um Objekte zu identifizieren. Das Identifizieren erfolgt mit dem Operator

- apply_sample_identifier.

Weitere Operatoren zur Administration und Kontrolle des Sample-Identifikators

Zusätzlich zu den oben genannten Operatoren können folgende weitere Operatoren zur Administration des Sample-Identifikator verwendet werden. Mit remove_sample_identifier_preparation_data und remove_sample_identifier_training_data können die Daten, die zur Vorbereitung bzw. zum Training des Sample-Identifikator hinzugefügt wurden, wieder gelöscht werden. Sie werden dann bei einer weiteren Vorbereitung bzw. einem weiteren Training nicht mehr verwendet. Die Operatoren set_sample_identifier_object_info und get_sample_identifier_object_info können dazu verwendet werden, die einzelnen Objekte mit einem Namen oder einer anderen Beschreibung zu kennzeichnen. Darüber hinaus können mit dem Operator get_sample_identifier_object_info weitere Informationen über die Anzahl der Objekte, die für die Vorbereitung bzw. das Training zur Verfügung stehen abgefragt werden. Mit den Operatoren set_sample_identifier_param und get_sample_identifier_param können Steuerparameter gesetzt und abgefragt werden.

Glossar

Im Folgenden werden die wichtigsten Begriffe, die im Umfeld der samplebasierten Identifikation verwendet werden, beschrieben:

Objekt Ein Objekt, das mit der samplebasierten Identifikation identifiziert werden soll.

Objektindex Der Index eines Objekts. Der Index ist eine eindeutige Kennzeichnung des Objekts, der im SampleIdentifier im Zuge des Hinzufügens von Daten für die Vorbereitung bzw. das Training gesetzt wird. Der Objektindex ist auch das wesentliche Ergebnis des Operators apply_sample_identifier. Mit dem Operator set_sample_identifier_object_info kann für jedes Objekt auch ein beschreibender Name gesetzt werden, der die Interpretation der Ergebnisse erleichtert.

Objektsample Ein Sample oder eine Ansicht eines Objekts. Manchmal wird hierfür auch der Begriff „Samplebild´´ verwendet.

Samplebild Dieser Begriff wird als Synonym für Objektsample verwendet, insbesondere dann, wenn der Schwerpunkt auf dem Bild liegt.

Objektsampleindex Der Index eines Objektsamples. Dieser Index wird für jedes Objekt getrennt vergeben. Er startet für jedes Objekt bei *0*. Daher ist der Objektsampleindex nur zusammen mit dem entsprechenden Objektindex eindeutig.

Vorbereitung Die Anpassung der internen Datenstruktur des Sample-Identifikators an die Merkmale eines Satzes von Objektsamples. Diese Satz sollte die bei der Anwendung der Identifikation zu erwartenden Objektsamples beispielhaft abdecken.

Vorbereitungsobjekt Ein Objekt, das dem Sample-Identifikator mit dem Operator add_sample_identifier_preparation_data hinzugefügt wurde.

Sample für die Vorbereitung Ein Objektsample eines Vorbereitungsobjekts. Die Vorbereitung des Sample-Identifikators wird typischerweise mit mehreren Samples für die Vorbereitung pro Vorbereitungsobjekt durchgeführt.

Vorbereitungsdaten Die Menge aller Samples für die Vorbereitung.

Training Das Training des Sample-Identifikators. Bei diesem Schritt lernt der Sample-Identifikator die vorgegebenen Objekte zu unterscheiden.

Trainingsobjekt Ein Objekt, das dem Sample-Identifikator mit dem Operator `add_sample_identifier_training_data` hinzugefügt wurde oder das aus der Wiederverwendung der Vorbereitungsdaten entstanden ist. Im Gegensatz zu Vorbereitungsobjekten müssen alle Trainingsobjekte mit einem eindeutigen Objektindex gekennzeichnet sein.

Sample für das Training Ein Objektsample eines Trainingsobjekts. Das Training des Sample-Identifikators wird typischerweise mit mehreren Samples für das Training pro Trainingsobjekt durchgeführt.

Trainingsdaten Die Menge aller Samples für das Training.

Suchbild Ein Bild, das ein Objekt enthält, welches mit dem Sample-Identifikator identifiziert werden soll.

```
add_sample_identifier_preparation_data (
    SampleImage : : SampleIdentifier, ObjectIdx, GenParamName,
    GenParamValue : ObjectSampleIdx )
```

Hinzufügen von Samples für die Vorbereitung eines Sample-Identifikators.

`add_sample_identifier_preparation_data` fügt einem bestehenden Sample-Identifikator Samples für die Vorbereitung hinzu. Dies ist eine Voraussetzung für die Vorbereitung des Sample-Identifikators mit dem Operator `prepare_sample_identifier`.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Um die besten Identifikationsergebnisse zu erzielen, muss die interne Datenstruktur des Sample-Identifikators an die Art der Objekte angepasst werden, die unterschieden werden sollen. Hierfür sollten Samplebilder für alle zu unterscheidenden Arten von Objekten verwendet werden.

Mit jedem Aufruf von `add_sample_identifier_preparation_data` werden dem Sample-Identifikator `SampleIdentifier` die Merkmale eines Samplebildes `SampleImage` hinzugefügt. Die gesamten Vorbereitungsdaten bestehen aus den Merkmalen aller Samplebilder, die dem Sample-Identifikator mit mehreren Aufrufen von `add_sample_identifier_preparation_data` hinzugefügt wurden. Die Vorbereitungsdaten sollten

- alle Arten von Objekten, die unterschieden werden sollen und
- alle unterschiedlichen Ansichten der Objekte, die bei der Anwendung des Sample-Identifikators auftreten können (in maximal 45°-Schritten abgetastet)

abdecken.

Die Domain des Samplebildes `SampleImage` sollte auf das im Bild sichtbare Objekt reduziert sein. Dies ist notwendig, um zu vermeiden, dass sich der `SampleIdentifier` an den Hintergrund der Samplebilder anpasst.

Wenn die Vorbereitungsdaten die oben genannten Anforderungen nicht vollständig erfüllen, kann der Sample-Identifikator trotzdem damit vorbereitet werden. Allerdings kann die Qualität des Identifikationsergebnisses darunter leiden.

Es ist zu beachten, dass RGB-Farbbilder zur Vorbereitung verwendet werden müssen, wenn beim Anlegen des Sample-Identifikators mit dem Operator `create_sample_identifier` der generische Parameter *'add_color_info'* auf *'true'* gesetzt wurde.

Idealerweise sind die Bilder, die für die Vorbereitung und für das Training (siehe `add_sample_identifier_training_data` und `train_sample_identifier`) verwendet werden, identisch. Dies kann sehr einfach durch eine Wiederverwendung der Vorbereitungsdaten für das Training erreicht werden. Hierzu ist es allerdings notwendig, dass für jedes Samplebild der Objektindex in dem Parameter `ObjectIdx` angegeben wird. Wird die Wiederverwendung der Vorbereitungsdaten nicht gewünscht, kann `ObjectIdx` auf *'unknown'* gesetzt werden. Siehe `prepare_sample_identifier` für eine Beschreibung, wie die Vorbereitungsdaten für das Training des Sample-Identifikators wiederverwendet werden können.

Mit den folgenden generischen Parametern kann das Verhalten des Operators `add_sample_identifier_preparation_data` beeinflusst werden. Diese Parameter und ihre Werte können in `GenParamName` und `GenParamValue` übergeben werden. Die folgenden generischen Parameter können gesetzt werden:

'image_resize_method': Siehe create_sample_identifier für eine Beschreibung dieses Parameters.

Werteliste: 'none', 'scale_factor', 'subsampling_step', 'image_area'

Defaultwert: Wenn *'image_resize_method'* nicht explizit durch diesen Operator gesetzt wird, wird der Wert verwendet, der mit create_sample_identifier oder set_sample_identifier_param gesetzt wurde.

'image_resize_value': Siehe create_sample_identifier für eine Beschreibung dieses Parameters.

Wertevorschläge: 0.25, 0.5, 1.0, 2, 3, 4

Defaultwert: Wenn *'image_resize_value'* nicht explizit durch diesen Operator gesetzt wird, wird der Wert verwendet, der mit create_sample_identifier oder set_sample_identifier_param gesetzt wurde.

add_sample_identifier_preparation_data gibt den Objektsampleindex für das in SampleImage übergebene Sample für die Vorbereitung zurück. Dieser Index kann z.B. verwendet werden, um das entsprechende Sample aus den Vorbereitungsdaten zu entfernen, falls der Sample-Identifikator mit einem anderen Satz von Vorbereitungsdaten vorbereitet werden soll.

Parameter

- ▷ **SampleImage** (input_object) ..... (multichannel-)image ⇝ *object* : byte
  Bild, das ein Objekt zeigt.
- ▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
  Handle des Sample-Identifikators.
- ▷ **ObjectIdx** (input_control) ..... integer ⇝ *integer* / string
  Index des im Bild SampleImage sichtbaren Objekts.
  Defaultwert: 'unknown'
  Wertevorschläge: ObjectIdx ∈ {'unknown', 0, 1, 2, 3, 4, 5}
- ▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
  Generischer Parametername.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'image_resize_method', 'image_resize_value'}
- ▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *real* / string / integer
  Generischer Parameterwert.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {'none', 'scale_factor', 'subsampling_step', 'image_area', 0.25, 0.5, 0.75, 1.0, 2, 3, 4}
- ▷ **ObjectSampleIdx** (output_control) ..... integer ⇝ *integer*
  Index des hinzugefügten Objektsamples.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_sample_identifier_preparation_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

create_sample_identifier, read_sample_identifier

Nachfolger

prepare_sample_identifier

Alternativen

read_sample_identifier

Siehe auch

add_sample_identifier_training_data, train_sample_identifier, apply_sample_identifier, set_sample_identifier_param, get_sample_identifier_param, get_sample_identifier_object_info,

remove_sample_identifier_preparation_data,
remove_sample_identifier_training_data, write_sample_identifier,
serialize_sample_identifier, deserialize_sample_identifier,
clear_sample_identifier, set_sample_identifier_object_info

Modul

Matching

add_sample_identifier_training_data (
SampleImage : : SampleIdentifier, ObjectIdx, GenParamName,
GenParamValue : ObjectSampleIdx)

Hinzufügen von Samples für das Training eines Sample-Identifikators.

add_sample_identifier_training_data fügt einem bestehenden Sample-Identifikator Samples für das Training hinzu. Dies ist eine Voraussetzung für das Training des Sample-Identifikators mit dem Operator train_sample_identifier.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Um Objekte identifizieren zu können, muss der Sample-Identifikator SampleIdentifier mit einem repräsentativen Satz von Samplebildern der zu unterscheidenden Objekte trainiert werden. Diese Samplebilder müssen dem Sample-Identifikator SampleIdentifier mit dem Operator add_sample_identifier_training_data hinzugefügt werden.

Mit jedem Aufruf von add_sample_identifier_training_data werden dem Sample-Identifikator SampleIdentifier die Merkmale eines Samplebildes SampleImage hinzugefügt. Die gesamten Trainingsdaten bestehen aus den Merkmalen aller Samplebilder, die dem Sample-Identifikator mit mehreren Aufrufen von add_sample_identifier_training_data hinzugefügt wurden. Die Trainingsdaten müssen

- alle Arten von Objekten, die unterschieden werden sollen und
- alle unterschiedlichen Ansichten der Objekte, die bei der Anwendung des Sample-Identifikators auftreten können (in maximal 45°-Schritten abgetastet)

abdecken.

Die Domain des Samplebildes SampleImage sollte auf das im Bild sichtbare Objekt reduziert sein. Dies ist notwendig, um zu vermeiden, dass der SampleIdentifier auf den Hintergrund der Samplebilder trainiert wird.

Es ist zu beachten, dass es, im Gegensatz zu den Vorbereitungsdaten, unbedingt notwendig ist, dass die Trainingsdaten den oben genannten Anforderungen entsprechen. Andernfalls kann der Sample-Identifikator die Objekte, für die die relevanten Samplebilder fehlen, nicht identifizieren.

Es ist auch zu beachten, dass RGB-Farbbilder für das Training verwendet werden müssen, wenn beim Anlegen des Sample-Identifikators mit dem Operator create_sample_identifier der generische Parameter *'add_color_info'* auf *'true'* gesetzt wurde.

Idealerweise sind die Bilder, die für die Vorbereitung (siehe add_sample_identifier_preparation_data und prepare_sample_identifier) und für das Training verwendet werden, identisch. Siehe prepare_sample_identifier für eine Beschreibung, wie die Vorbereitungsdaten für das Training des Sample-Identifikators wiederverwendet werden können.

Mit den folgenden generischen Parametern kann das Verhalten des Operators add_sample_identifier_training_data beeinflusst werden. Diese Parameter und ihre Werte können in GenParamName und GenParamValue übergeben werden. Die folgenden generischen Parameter können gesetzt werden:

'image_resize_method': Siehe create_sample_identifier für eine Beschreibung dieses Parameters.

Werteliste: 'none', 'scale_factor', 'subsampling_step', 'image_area'

Defaultwert: Wenn *'image_resize_method'* nicht explizit durch diesen Operator gesetzt wird, wird der Wert verwendet, der mit create_sample_identifier oder set_sample_identifier_param gesetzt wurde.

'image_resize_value': Siehe create_sample_identifier für eine Beschreibung dieses Parameters.

Wertevorschläge: 0.25, 0.5, 1.0, 2, 3, 4

Defaultwert: Wenn *'image_resize_value'* nicht explizit durch diesen Operator gesetzt wird, wird der Wert verwendet, der mit create_sample_identifier oder set_sample_identifier_param gesetzt wurde.

add_sample_identifier_training_data gibt den Objektsampleindex des Samples für das Training, das in SampleImage übergeben wurde, zurück. Dieser Index kann z.B. verwendet werden, um das entsprechende Sample aus den Trainingsdaten zu entfernen, falls der Sample-Identifikator mit einem anderen Satz von Trainingsdaten trainiert werden soll.

Parameter

- ▷ **SampleImage** (input_object) ..... (multichannel-)image ⇝ *object : byte*
  Bild, das ein Objekt zeigt.
- ▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
  Handle des Sample-Identifikators.
- ▷ **ObjectIdx** (input_control) ..... integer ⇝ *integer* / string
  Index des im Bild SampleImage sichtbaren Objekts.
  Wertevorschläge: ObjectIdx ∈ {0, 1, 2, 3, 4, 5}
- ▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
  Generischer Parametername.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'image_resize_method', 'image_resize_value'}
- ▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *real* / string / integer
  Generischer Parameterwert.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {'none', 'scale_factor', 'subsampling_step', 'image_area', 0.25, 0.5, 0.75, 1.0, 2, 3, 4}
- ▷ **ObjectSampleIdx** (output_control) ..... integer ⇝ *integer*
  Index des hinzugefügten Objektsamples.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_sample_identifier_training_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

prepare_sample_identifier, read_sample_identifier

Nachfolger

train_sample_identifier

Alternativen

read_sample_identifier

Siehe auch

create_sample_identifier, add_sample_identifier_preparation_data, apply_sample_identifier, set_sample_identifier_param, get_sample_identifier_param, get_sample_identifier_object_info, remove_sample_identifier_preparation_data, remove_sample_identifier_training_data, write_sample_identifier, serialize_sample_identifier, deserialize_sample_identifier, clear_sample_identifier, set_sample_identifier_object_info

Modul

Matching

apply_sample_identifier (Image : : SampleIdentifier, NumResults, RatingThreshold, GenParamName, GenParamValue : ObjectIdx, Rating)

Identifizieren von Objekten mit einem Sample-Identifikator.

apply_sample_identifier identifiziert das Objekt im Bild Image mit dem gegebenen Sample-Identifikator SampleIdentifier und liefert den entsprechenden Objektindex in ObjectIdx zurück.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Der Operator apply_sample_identifier kann nur ein Objekt pro Suchbild identifizieren. Das bedeutet, dass das in Image übergebene Suchbild, oder genauer gesagt dessen Domain, nur ein Objekt enthalten darf. Der Parameter NumResults gibt die Anzahl von Hypothesen an, die zurückgegeben werden. Die Hypothesen sind dabei nach ihrer Bewertung sortiert. Wenn der generische Parameter *'apply_rating_threshold'* auf *'true'* gesetzt ist (siehe unten), werden nur solche Hypothesen zurück gegeben, deren Bewertung besser als der in RatingThreshold angegebenen Schwellenwert ist.

Der Objektindex des identifizierten Objekts wird in ObjectIdx zurückgegeben und die entsprechende Bewertung in Rating. Wenn NumResults auf einen Wert größer als *1* gesetzt ist, werden die Objektindizes und die Bewertungen der besten NumResults Hypothesen in ObjectIdx und Rating zurückgegeben.

Mit den folgenden generischen Parametern kann das Verhalten des Sample-Identifikators SampleIdentifier beeinflusst werden. Die Parameter und ihre Werte können in GenParamName und GenParamValue übergeben werden. Die folgenden generischen Parameter können gesetzt werden:

'rating_method': Dieser Parameter legt fest, welche Bewertungsmethode für die Identifikation der Objekte verwendet wird. Es stehen drei unterschiedliche Bewertungsmethoden zur Auswahl:

'distance' Die Bewertung wird aufgrund der Distanz zwischen den Merkmalen im Suchbild und den trainierten Samplebildern durchgeführt. Die Bewertungen liegen zwischen *0.0* und *2.0*. Sind Suchbild und Samplebild identisch, so wird eine Bewertung von *0.0* im Parameter Rating zurückgegeben. Je stärker sich die Bilder unterscheiden, desto höher wird die Bewertung. Bei dieser Bewertungsmethode werden die Rohdaten der Bewertung zurückgegeben. Diese Bewertungsmethode hat den Vorteil, dass die Bewertung nur von den Samples für das Training abhängt, die für das identifizierte Objekt trainiert wurden. Die Bewertungen der beiden anderen Bewertungsmethoden hängen auch von allen übrigen Samples für das Training ab.

'score' Die Bewertung basiert auf einer komplizierten Kombination der einzelnen Auswertungen für Textur und Farbe. Die Bewertungen liegen zwischen *0.0* und *1.0*. Höhere Werte bedeuten eine bessere Übereinstimmung. Wird sowohl Textur als auch Farbe verwendet, liefert diese Bewertungsmethode die besten Ergebnisse und sollte daher in diesem Fall auch verwendet werden. Ein Nachteil dieser Bewertungsmethode ist, dass die Bewertungen von Trainingsbildern typischerweise deutlich unter *1.0* liegt. Außerdem muss ein passender Schwellenwert für jede Anwendung gesondert bestimmt werden.

'score_single' Die Bewertung basiert auf einer einfachen Kombination der einzelnen Auswertungen für Textur und Farbe. Die Bewertungen liegen zwischen *0.0* und *1.0*. Höhere Werte bedeuten eine bessere Übereinstimmung. Wird sowohl Textur wie auch Farbe verwendet, liefert die Bewertungsmethode *'score'* bessere Ergebnisse. Der Vorteil der Bewertungsmethode *'score_single'* ist, dass sie für Trainingsbilder die Bewertung *1.0* liefert. Daher sollte diese Methode angewendet werden, wenn nur Textur oder nur Farbe für die Identifikation verwendet werden. Bei dieser Bewertungsmethode ist es auch leichter einen passenden Schwellenwert festzulegen, als bei der Bewertungsmethode *'score'*.

Es ist zu beachten, dass die Wahl der Bewertungsmethode nicht nur die (Zahlenwerte der) Bewertung beeinflusst, die im Parameter Rating zurückgegeben wird, sondern auch die eigentliche Identifikation der Objekte.

Werteliste: 'distance', 'score', 'score_single'

Defaultwert: 'score'

'apply_rating_threshold': Dieser Parameter legt fest, ob der in RatingThreshold übergebene Schwellenwert angewendet wird, oder nicht. Wenn *'apply_rating_threshold'* auf *'true'* gesetzt ist, wird der Schwellenwert angewendet.

Werteliste: 'true', 'false'

Defaultwert: 'true'

'use_color_info': Siehe `set_sample_identifier_param` für eine Beschreibung dieses Parameters.

Werteliste: 'true', 'false'

Defaultwert: So lange *'use_color_info'* nicht gesetzt wird, wird der Wert verwendet, der mit *'add_color_info'* in `create_sample_identifier` gesetzt wurde.

'use_texture_info': Siehe `set_sample_identifier_param` für eine Beschreibung dieses Parameters.

Werteliste: 'true', 'false'

Defaultwert: So lange *'use_texture_info'* nicht gesetzt wird, wird der Wert verwendet, der mit *'add_texture_info'* in `create_sample_identifier` gesetzt wurde.

'image_resize_method': Siehe `create_sample_identifier` für eine Beschreibung dieses Parameters.

Werteliste: 'none', 'scale_factor', 'subsampling_step', 'image_area'

Defaultwert: Wenn *'image_resize_method'* nicht explizit mit diesem Operator gesetzt wird, wird der Wert verwendet, der mit `create_sample_identifier` oder mit `set_sample_identifier_param` gesetzt wurde.

'image_resize_value': Siehe `create_sample_identifier` für eine Beschreibung dieses Parameters. Es ist hierbei zu beachten, dass die zu identifizierenden Objekte in den Suchbildern in etwa in derselben Größe erscheinen, wie in den Samples für das Training.

Wertevorschläge: 0.25, 0.5, 1.0, 2, 3, 4

Defaultwert: Wenn *'image_resize_value'* nicht explizit mit diesem Operator gesetzt wird, wird der Wert verwendet, der mit `create_sample_identifier` oder mit `set_sample_identifier_param` gesetzt wurde.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object* : byte
Bild, das das zu identifizierende Objekt zeigt.

▷ **`SampleIdentifier`** (input_control) ..... sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

▷ **`NumResults`** (input_control) ..... integer ⇝ *integer*
Anzahl der Hypothesen, die zurückgegeben werden.
Defaultwert: 1
Wertevorschläge: `NumResults` ∈ {1, 2, 3, 4, 5, 10}

▷ **`RatingThreshold`** (input_control) ..... real ⇝ *real*
Schwellenwert für die Bewertung.
Defaultwert: 0.0
Wertevorschläge: `RatingThreshold` ∈ {0.05, 0.1, 0.15, 0.2}

▷ **`GenParamName`** (input_control) ..... attribute.name-array ⇝ *string*
Generischer Parametername.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'use_texture_info', 'use_color_info', 'image_resize_method', 'image_resize_value', 'rating_method', 'apply_rating_threshold'}

▷ **`GenParamValue`** (input_control) ..... attribute.value-array ⇝ *real* / string / integer
Generischer Parameterwert.
Defaultwert: []
Werteliste: `GenParamValue` ∈ {'true', 'false', 'distance', 'score', 'score_single'}

▷ **`ObjectIdx`** (output_control) ..... integer(-array) ⇝ *integer*
Index des identifizierten Objekts.

▷ **`Rating`** (output_control) ..... real(-array) ⇝ *real*
Bewertung des identifizierten Objekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `apply_sample_identifier` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

train_sample_identifier, read_sample_identifier

Nachfolger

add_sample_identifier_training_data

Siehe auch

create_sample_identifier, add_sample_identifier_preparation_data, prepare_sample_identifier, set_sample_identifier_param, get_sample_identifier_param, get_sample_identifier_object_info, remove_sample_identifier_preparation_data, remove_sample_identifier_training_data, write_sample_identifier, serialize_sample_identifier, deserialize_sample_identifier, clear_sample_identifier, set_sample_identifier_object_info

Modul

Matching

clear_sample_identifier (: : SampleIdentifier :)

Freigeben des Speichers eines Sample-Identifikators.

clear_sample_identifier gibt den Speicher eines Sample-Identifikators, der mit create_sample_identifier, read_sample_identifier, oder deserialize_sample_identifier angelegt wurde, wieder frei. Der Sample-Identifikator kann nach dem Aufruf nicht mehr verwendet werden. Der Handle SampleIdentifier ist nach dem Aufruf ungültig.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Parameter

▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

Ergebnis

Ist der Handle des Sample-Identifikators gültig, dann liefert clear_sample_identifier den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SampleIdentifier

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

write_sample_identifier, serialize_sample_identifier

Siehe auch

create_sample_identifier, add_sample_identifier_preparation_data, prepare_sample_identifier, add_sample_identifier_training_data, train_sample_identifier, apply_sample_identifier, set_sample_identifier_param, get_sample_identifier_param, get_sample_identifier_object_info, remove_sample_identifier_preparation_data, remove_sample_identifier_training_data, read_sample_identifier, deserialize_sample_identifier, set_sample_identifier_object_info

Modul

Matching

create_sample_identifier (: : GenParamName, GenParamValue : SampleIdentifier)

Anlegen eines neuen Sample-Identifikators.

Der Operator create_sample_identifier legt einen neuen Sample-Identifikator an. Alternativ dazu kann ein bereits verfügbarer Sample-Identifikator mit dem Operator read_sample_identifier von Datei eingelesen werden.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Wenn Farbinformation für die Identifikation verwendet werden soll, ist es notwendig, dies explizit anzugeben. Dies erfolgt durch das Setzen des generischen Parameters *'add_color_info'* auf *'true'*.

Mit den generischen Parametern kann das Verhalten des Sample-Identifikators beeinflusst werden. Typischerweise muss hierbei aber nur der Parameter *'add_color_info'* berücksichtigt werden. Die generischen Parameter und ihre Werte können in GenParamName und GenParamValue übergeben werden. Die folgenden generischen Parameter können gesetzt werden:

'add_color_info': Dieser Parameter legt fest, ob Farbinformation für die Identifikation verwendet wird. Ist *'add_color_info'* auf *'true'* gesetzt, so wird Farbinformation verwendet. In diesem Fall müssen alle Bilder, die für die Vorbereitung und das Training des Sample-Identifikators verwendet werden, RGB-Farbbilder sein. Ist *'add_color_info'* auf *'false'* gesetzt, wird keine Farbinformation verwendet. Farbinformation sollte verwendet werden, wenn die zu unterscheidenden Objekte unterschiedliche Farben haben und die Beleuchtung einigermaßen gut kontrolliert werden kann. In diesem Fall wird die Identifikation durch die Verwendung von Farbinformation robuster.

Werteliste: 'true', 'false'

Defaultwert: 'false'

'add_texture_info': Dieser Parameter legt fest, ob Texturinformation für die Identifikation verwendet wird. Ist *'add_texture_info'* auf *'true'* gesetzt, so wird Texturinformation verwendet. Ist *'add_texture_info'* auf *'false'* gesetzt, wird keine Texturinformation verwendet. Normalerweise sollte *'add_texture_info'* auf *'true'* gesetzt sein, da die samplebasierte Identifikation nur mit texturierten Objekten funktioniert. Es ist zu beachten, dass mindestens einer der beiden generischen Parameter *'add_color_info'* und *'add_texture_info'* auf *'true'* gesetzt sein muss.

Werteliste: 'true', 'false'

Defaultwert: 'true'

'image_resize_method': Um die Identifikation zu beschleunigen werden die Bilder intern auf eine vorgegebene Größe oder mit einem bestimmten Faktor skaliert. Ist *'image_resize_method'* auf *'none'* gesetzt, so wird diese Skalierung nicht durchgeführt. Für den Parameter *'scale_factor'* kann ein konstanter Skalierungsfaktor angegeben werden und für *'subsampling_step'* eine konstante Schrittweite für die Unterabtastung (der inverse Skalierungsfaktor). Für *'image_area'* kann eine feste Größe für das interne, skalierte Bild angegeben werden. Die entsprechenden Werte können mit *'image_resize_value'* gesetzt werden (siehe unten).

Um den Identifikationsprozess zu beschleunigen, sollte ein Wert gewählt werden, der zu einem kleineren internen Bild führt, also ein kleinerer Skalierungsfaktor oder eine kleinere Bildgröße bzw. eine größere Schrittweite für die Unterabtastung. Wenn die zu unterscheidenden Objekte hochfrequente Textur aufweisen, wird die Identifikation möglicherweise bessere Ergebnisse liefern, wenn eine größere interne Bildgröße verwendet wird, da ansonsten die Textur bei der Verkleinerung des Bildes verloren gehen kann.

Werteliste: 'none', 'scale_factor', 'subsampling_step', 'image_area'

Defaultwert: 'image_area'

'image_resize_value': Mit diesem Parameter wird die Skalierungsmethode (siehe *'image_resize_method'*) parametrisiert. Wenn für *'image_resize_method'* der Wert *'scale_factor'* gesetzt ist, wird mit *'image_resize_value'* der Skalierungsfaktor festgelegt. Wenn für *'image_resize_method'* der Wert *'subsampling_step'* gesetzt ist, wird mit *'image_resize_value'* die Schrittweite für die Unterabtastung, also der inverse Skalierungsfaktor, festgelegt. Wenn für *'image_resize_method'* der Wert *'image_area'* gesetzt ist, wird mit

'image_resize_value' die Größe des internen, skalierten Bildes festgelegt. Diese Größe wird in Megapixeln, d.h., in Millionen Pixel angegeben.

Wertevorschläge: 0.25, *0.5*, *1.0*, *2*, *3*, *4*

Defaultwert: Der Defaultwert hängt von der mit *'image_resize_method'* gewählten Methode ab. Er beträgt

- *0.5* für *'scale_factor'*,
- *2.0* für *'subsampling_step'* und
- *0.5* für *'image_area'*.

Parameter

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Parametername.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'image_resize_method', 'image_resize_value', 'add_texture_info', 'add_color_info'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *string* / real / integer
Parameterwert.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'none', 'scale_factor', 'subsampling_step', 'image_area', 0.25, 0.5, 0.75, 1.0, 2, 3, 4, 'true', 'false'}

▷ **SampleIdentifier** (output_control) . . . . . sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_sample_identifier` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`add_sample_identifier_preparation_data`, `set_sample_identifier_param`

Alternativen

`read_sample_identifier`

Siehe auch

`prepare_sample_identifier`, `add_sample_identifier_training_data`, `train_sample_identifier`, `apply_sample_identifier`, `get_sample_identifier_param`, `get_sample_identifier_object_info`, `remove_sample_identifier_preparation_data`, `remove_sample_identifier_training_data`, `write_sample_identifier`, `serialize_sample_identifier`, `deserialize_sample_identifier`, `clear_sample_identifier`, `set_sample_identifier_object_info`

Modul

Matching

deserialize_sample_identifier

```
deserialize_sample_identifier (
    : : SerializedItemHandle : SampleIdentifier )
```

Deserialisieren eines Sample-Identifikators.

`deserialize_sample_identifier` deserialisiert einen Sample-Identifikator, welcher mit `serialize_sample_identifier` serialisiert wurde (siehe `fwrite_serialized_item` für eine

Einführung in die Grundlagen der Serialisierung). Der serialisierte Sample-Identifikator wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Sample-Identifikator mit dem Handle `SampleIdentifier` gespeichert.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Parameter

▷ **`SerializedItemHandle`** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **`SampleIdentifier`** (output_control) ..... sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_sample_identifier` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`apply_sample_identifier`, `add_sample_identifier_training_data`

Alternativen

`read_sample_identifier`

Siehe auch

`create_sample_identifier`, `add_sample_identifier_preparation_data`, `prepare_sample_identifier`, `train_sample_identifier`, `set_sample_identifier_param`, `get_sample_identifier_param`, `get_sample_identifier_object_info`, `remove_sample_identifier_preparation_data`, `remove_sample_identifier_training_data`, `write_sample_identifier`, `serialize_sample_identifier`, `clear_sample_identifier`, `set_sample_identifier_object_info`

Modul

Matching

`get_sample_identifier_object_info` `( : : SampleIdentifier, ObjectIdx, InfoName : InfoValue )`

Abfragen von Informationen über einen Sample-Identifikator.

`get_sample_identifier_object_info` fragt Informationen über den Sample-Identifikator `SampleIdentifier` ab. Diese Informationen umfassen die Anzahl von Objekten und Objektsamples, sowie deren Indizes. Darüber hinaus können auch die Informationen abgefragt werden, die mit `set_sample_identifier_object_info` gesetzt wurden.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Mit dem Parameter `ObjectIdx` wird der Index des Objekts angegeben, für das Informationen abgefragt werden. Dieser Parameter wird nicht ausgewertet, falls der Parameter `InfoName` auf *'num_preparation_objects'*, *'preparation_object_idx'*, *'num_training_objects'* oder *'training_object_idx'* gesetzt ist.

Mit dem Parameter `InfoName` wird die Art der abzufragende Information, die im Parameter `InfoValue` zurückgegeben wird, festgelegt. Folgende Werte sind für den Parameter `InfoName` möglich:

'num_preparation_objects': Die Anzahl von Vorbereitungsobjekten. Es ist dabei zu beachten, dass sämtliche Vorbereitungsobjekte, für die der Objektindex auf *'unknown'* gesetzt wurde, als ein einziges Vorbereitungsobjekt gezählt werden.

'preparation_object_idx': Die Liste der Indizes aller verfügbaren Vorbereitungsobjekte. Es ist dabei zu beachten, dass diese Liste die Zeichenkette *'unknown'* enthält, wenn der Objektindex von mindestens einem Vorbereitungsobjekt auf *'unknown'* gesetzt wurde.

'num_preparation_samples': Die Anzahl von Samples für die Vorbereitung, die dem Sample-Identifikator für das Vorbereitungsobjekt mit dem Objektindex `ObjectIdx` hinzugefügt wurden.

'preparation_sample_idx': Die Liste der Indizes aller verfügbaren Samples für die Vorbereitung, die dem Sample-Identifikator für das Vorbereitungsobjekt mit dem Objektindex `ObjectIdx` hinzugefügt wurden.

'num_training_objects': Die Anzahl der Trainingsobjekte.

'training_object_idx': Die Liste der Indizes aller verfügbaren Trainingsobjekte.

'num_training_samples': Die Anzahl von Samples für das Training, die dem Sample-Identifikator für das Trainingsobjekt mit dem Objektindex `ObjectIdx` hinzugefügt wurden.

'training_sample_idx': Die Liste der Indizes aller verfügbaren Samples für das Training, die dem Sample-Identifikator für das Trainingsobjekt mit dem Objektindex `ObjectIdx` hinzugefügt wurden.

'preparation_object_name': Die Information, die mit dem Operator `set_sample_identifier_object_info` für das Vorbereitungsobjekt mit dem Objektindex `ObjectIdx` gesetzt wurde.

'training_object_name': Die Information, die mit dem Operator `set_sample_identifier_object_info` für das Trainingsobjekt mit dem Objektindex `ObjectIdx` gesetzt wurde.

Parameter

▷ **`SampleIdentifier`** (input_control) . . . . . sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

▷ **`ObjectIdx`** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Index des Objekts, für das Information abgefragt wird.
Wertevorschläge: `ObjectIdx` ∈ {0, 1, 2, 3, 4, 5, 'unknown'}

▷ **`InfoName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Gibt an, welche Art von Information abgefragt wird.
Defaultwert: 'num_training_objects'
Werteliste: `InfoName` ∈ {'num_preparation_objects', 'preparation_object_idx', 'num_preparation_samples', 'preparation_sample_idx', 'num_training_objects', 'training_object_idx', 'num_training_samples', 'training_sample_idx', 'preparation_object_name', 'training_object_name'}

▷ **`InfoValue`** (output_control) . . . . . attribute.value(-array) ⇝ *integer* / string
Information zu dem Sample-Identifikator bzw. einem Objekt.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_sample_identifier_object_info` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_identifier_preparation_data`, `add_sample_identifier_training_data`

Siehe auch

`create_sample_identifier`, `prepare_sample_identifier`, `train_sample_identifier`, `apply_sample_identifier`, `set_sample_identifier_param`, `get_sample_identifier_param`, `remove_sample_identifier_preparation_data`, `remove_sample_identifier_training_data`, `write_sample_identifier`, `read_sample_identifier`, `serialize_sample_identifier`, `deserialize_sample_identifier`, `clear_sample_identifier`

Modul

Matching

get_sample_identifier_param (: : SampleIdentifier, GenParamName : GenParamValue)

Abfragen bestimmter Parameter eines Sample-Identifikators.

Mit dem Operator get_sample_identifier_param können die Werte bestimmte Parameter des Sample-Identifikators SampleIdentifier abgefragt werden.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Die Werte folgender Parameter können abgefragt werden:

'add_color_info': Fügen die Operatoren add_sample_identifier_preparation_data und add_sample_identifier_training_data dem Sample-Identifikator SampleIdentifier Farbinformation hinzu?

'add_texture_info': Fügen die Operatoren add_sample_identifier_preparation_data und add_sample_identifier_training_data dem Sample-Identifikator SampleIdentifier Texturinformation hinzu?

'use_color_info': Ist der Sample-Identifikator SampleIdentifier so konfiguriert, dass für die Identifikation von Objekten mit apply_sample_identifier Farbinformation verwendet wird?

'use_texture_info': Ist der Sample-Identifikator SampleIdentifier so konfiguriert, dass für die Identifikation von Objekten mit apply_sample_identifier Texturinformation verwendet wird?

'image_resize_method': Die eingestellte Methode für die Skalierung der internen Bilder.

'image_resize_value': Der eingestellte Skalierungswert, der für die Skalierung der internen Bilder verwendet wird.

'rating_method': Die gewählte Bewertungsmethode, die nur von apply_sample_identifier verwendet wird.

Parameter

▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

▷ **GenParamName** (input_control) ..... attribute.name ⇝ *string*
Parametername.
Defaultwert: 'rating_method'
Werteliste: GenParamName ∈ {'add_texture_info', 'add_color_info', 'use_texture_info', 'use_color_info', 'image_resize_method', 'image_resize_value', 'rating_method'}

▷ **GenParamValue** (output_control) ..... attribute.value ⇝ *real* / string / integer
Parameterwert.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sample_identifier_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_sample_identifier, set_sample_identifier_param

Siehe auch

add_sample_identifier_preparation_data, prepare_sample_identifier, add_sample_identifier_training_data, train_sample_identifier, apply_sample_identifier, get_sample_identifier_object_info, remove_sample_identifier_preparation_data, remove_sample_identifier_training_data, write_sample_identifier, read_sample_identifier, serialize_sample_identifier,

deserialize_sample_identifier, clear_sample_identifier, set_sample_identifier_object_info

Modul

Matching

prepare_sample_identifier (: : SampleIdentifier, RemovePreparationData, GenParamName, GenParamValue :)

Anpassen der internen Datenstruktur eines Sample-Identifikators an die Art der zu unterscheidenden Objekte.

prepare_sample_identifier bereitet den Sample-Identifikator SampleIdentifier vor. Hierzu werden die Vorbereitungsdaten verwendet, die mit add_sample_identifier_preparation_data gesetzt wurden.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Um die besten Identifikationsergebnisse zu erzielen, muss die interne Datenstruktur des Sample-Identifikators SampleIdentifier an die Art der Objekte angepasst werden, die unterschieden werden sollen.

Zur Vorbereitung des Sample-Identifikators SampleIdentifier müssen diesem zuerst alle relevanten Samplebilder der zu unterscheidenden Objekte mit add_sample_identifier_preparation_data hinzugefügt werden. Dann wird die interne Datenstruktur des Sample-Identifikators durch den Aufruf des Operators prepare_sample_identifier an die Merkmale der Samplebilder angepasst.

Dieser Vorgang dauert relativ lange und benötigt unter Umständen sehr viel Speicher. Wenn z.B. 100 Samplebilder für die Vorbereitung verwendet werden, dauert dies einige Minuten und es werden ca. 300 MB Speicher benötigt. Wenn 1000 Samplebilder verwendet werden, dauert die Vorbereitung je nach Rechner etwa eine Stunde und es werden ca. 2 GB Speicher benötigt.

Insbesondere die Vorbereitungsdaten benötigen viel Speicher. Normalerweise werden die Daten aber nach der Vorbereitung des Sample-Identifikators nicht mehr benötigt. Daher können sie durch das Setzen des Parameters RemovePreparationData auf *'true'* gelöscht werden.

Alternativ zur Vorbereitung des Sample-Identifikator kann ein bereits verfügbarer Sample-Identifikator mit dem Operator read_sample_identifier von Datei eingelesen werden. Dieser sollte mit ähnlichen Objekten vorbereitet worden sein. Allerdings ist zu beachten, dass die Qualität der Ergebnisse der Identifikation nicht so hoch sein wird, wie wenn der Sample-Identifikator mit Samples der zu unterscheidenden Objekte vorbereitet wurde.

Um die Vorbereitungsdaten für das Training wiederzuverwenden, ist es möglich, Trainingsdaten aus den Vorbereitungsdaten abzuleiten. Dies erspart den Aufwand, dem Sample-Identifikator die Trainingsdaten erneut mit add_sample_identifier_training_data hinzuzufügen (siehe train_sample_identifier für eine Beschreibung des Trainings eines Sample-Identifikators). Die Trainingsdaten erfordern deutlich weniger Speicher als die Vorbereitungsdaten. Wenn der generische Parameter *'copy_preparation_data_to_training_data'* auf *'true'* gesetzt ist, was der Default ist, so werden die Trainingsdaten automatisch aus den Vorbereitungsdaten abgeleitet. Dies ist allerdings nur für die Samples möglich, für die der Objektindex beim Hinzufügen zum Sample-Identifikator mit dem Operator add_sample_identifier_preparation_data gesetzt wurde. Der Objektindex wird dabei unverändert in die Trainingsdaten übernommen. Die Objektsampleindizes können sich allerdings ändern, wenn Samples für die Vorbereitung aus den Vorbereitungsdaten mit remove_sample_identifier_preparation_data gelöscht wurden, da Samples für das Training neu fortlaufend und beginnend bei null nummeriert werden.

Parameter

- ▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
  Handle des Sample-Identifikators.
- ▷ **RemovePreparationData** (input_control) ..... string ⇝ *string*
  Gibt an, ob die Vorbereitungsdaten gelöscht werden sollen.
  Defaultwert: 'true'
  Werteliste: RemovePreparationData ∈ {'true', 'false'}
- ▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
  Generischer Parametername.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'copy_preparation_data_to_training_data'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *string* / integer / real
Generischer Parameterwert.
Defaultwert: []
Werteliste: `GenParamValue` ∈ {'true', 'false'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `prepare_sample_identifier` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_identifier_preparation_data`

Nachfolger

`write_sample_identifier`, `add_sample_identifier_training_data`, `train_sample_identifier`

Alternativen

`read_sample_identifier`

Siehe auch

`create_sample_identifier`, `apply_sample_identifier`, `set_sample_identifier_param`, `get_sample_identifier_param`, `get_sample_identifier_object_info`, `remove_sample_identifier_preparation_data`, `remove_sample_identifier_training_data`, `write_sample_identifier`, `serialize_sample_identifier`, `deserialize_sample_identifier`, `clear_sample_identifier`, `set_sample_identifier_object_info`

Modul

Matching

read_sample_identifier (: : FileName : SampleIdentifier)

Einlesen eines Sample-Identifikators von Datei.

`read_sample_identifier` liest einen Sample-Identifikator, der mit `write_sample_identifier` geschrieben wurde, aus der Datei `FileName` ein. Die Dateiendung für den Sample-Identifikator ist in HALCON 'sid'.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.sid`

▷ **SampleIdentifier** (output_control) . . . . . sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

Ergebnis

Ist der Dateiname korrekt, dann liefert `read_sample_identifier` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`apply_sample_identifier`, `add_sample_identifier_training_data`

Alternativen

`deserialize_sample_identifier`

Siehe auch

`create_sample_identifier`, `add_sample_identifier_preparation_data`, `prepare_sample_identifier`, `train_sample_identifier`, `apply_sample_identifier`, `set_sample_identifier_param`, `get_sample_identifier_param`, `get_sample_identifier_object_info`, `remove_sample_identifier_preparation_data`, `remove_sample_identifier_training_data`, `write_sample_identifier`, `serialize_sample_identifier`, `clear_sample_identifier`, `set_sample_identifier_object_info`

Modul

Matching

remove_sample_identifier_preparation_data (: : SampleIdentifier, ObjectIdx, ObjectSampleIdx :)

Löschen von Vorbereitungsdaten eines Sample-Identifikators.

`remove_sample_identifier_preparation_data` löscht die Vorbereitungsdaten eines Sample-Identifikators.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

`ObjectIdx` gibt den Index des Vorbereitungsobjekts an, von dem Samples gelöscht werden sollen. Um Samples von Vorbereitungsobjekten zu löschen, für die der Objektindex mit `add_sample_identifier_preparation_data` auf *'unknown'* gesetzt wurde, muss `ObjectIdx` ebenfalls auf *'unknown'* gesetzt werden. Die Indexe aller momentan verfügbaren Vorbereitungsobjekte können mit `get_sample_identifier_object_info` abgefragt werden.

`ObjectSampleIdx` gibt den Index des Samples für die Vorbereitung an, das aus den Vorbereitungsdaten gelöscht werden soll. Beim Löschen des letzten Samples eines Vorbereitungsobjekts wird auch gleichzeitig das entsprechende Vorbereitungsobjekt selbst aus dem Sample-Identifikator `SampleIdentifier` gelöscht. Um alle Samples zu löschen, die zu dem Objekt mit dem gegebenen Objektindex `ObjectIdx` gehören, kann `ObjectSampleIdx` auf *'all'* gesetzt werden.

Um alle Samples **aller** Vorbereitungsobjekte zu löschen, müssen sowohl `ObjectIdx` als auch `ObjectSampleIdx` auf *'all'* gesetzt werden.

Parameter

- ▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
  Handle des Sample-Identifikators.
- ▷ **ObjectIdx** (input_control) ..... integer ⇝ *integer* / string
  Index des Vorbereitungsobjekts, dessen Samples gelöscht werden sollen.
  Wertevorschläge: `ObjectIdx` ∈ {0, 1, 2, 3, 4, 5, 'all', 'unknown'}
- ▷ **ObjectSampleIdx** (input_control) ..... integer ⇝ *integer* / string
  Index des Samples das gelöscht werden soll.
  Wertevorschläge: `ObjectSampleIdx` ∈ {0, 1, 2, 3, 4, 5, 'all'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `remove_sample_identifier_preparation_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_identifier_preparation_data`

Nachfolger

`prepare_sample_identifier`

Alternativen

`create_sample_identifier`

Siehe auch

`add_sample_identifier_training_data`, `train_sample_identifier`, `apply_sample_identifier`, `set_sample_identifier_param`, `get_sample_identifier_param`, `get_sample_identifier_object_info`, `remove_sample_identifier_training_data`, `write_sample_identifier`, `read_sample_identifier`, `serialize_sample_identifier`, `deserialize_sample_identifier`, `clear_sample_identifier`, `set_sample_identifier_object_info`

Modul

Matching

remove_sample_identifier_training_data (: : SampleIdentifier, ObjectIdx, ObjectSampleIdx :)

Löschen von Trainingsdaten eines Sample-Identifikators.

`remove_sample_identifier_training_data` löscht die Trainingsdaten eines Sample-Identifikators.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

`ObjectIdx` gibt den Index des Trainingsobjekts an, von dem Samples gelöscht werden sollen.

`ObjectSampleIdx` gibt den Index des Samples für das Training an, das aus den Trainingsdaten gelöscht werden soll. Um alle Samples zu löschen, die zu dem Objekt mit dem gegebenen Objektindex `ObjectIdx` gehören, kann `ObjectSampleIdx` auf *'all'* gesetzt werden. Hiermit wird auch gleichzeitig das entsprechende Trainingsobjekt selbst aus dem Sample-Identifikator `SampleIdentifier` gelöscht.

Um alle Samples aller Trainingsobjekte zu löschen, müssen sowohl `ObjectIdx` als auch `ObjectSampleIdx` auf *'all'* gesetzt werden.

Parameter

- ▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
  Handle des Sample-Identifikators.
- ▷ **ObjectIdx** (input_control) ..... integer ⇝ *integer* / string
  Index des Trainingsobjekts, dessen Samples gelöscht werden sollen.
  Wertevorschläge: `ObjectIdx` ∈ {0, 1, 2, 3, 4, 5, 'all'}
- ▷ **ObjectSampleIdx** (input_control) ..... integer ⇝ *integer* / string
  Index des Samples das gelöscht werden soll.
  Wertevorschläge: `ObjectSampleIdx` ∈ {0, 1, 2, 3, 4, 5, 'all'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `remove_sample_identifier_training_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_identifier_training_data

Nachfolger

train_sample_identifier

Alternativen

read_sample_identifier

Siehe auch

create_sample_identifier, add_sample_identifier_preparation_data, prepare_sample_identifier, apply_sample_identifier, set_sample_identifier_param, get_sample_identifier_param, get_sample_identifier_object_info, remove_sample_identifier_preparation_data, write_sample_identifier, serialize_sample_identifier, deserialize_sample_identifier, clear_sample_identifier, set_sample_identifier_object_info

Modul

Matching

serialize_sample_identifier (
: : SampleIdentifier : SerializedItemHandle)

Serialisieren eines Sample-Identifikators.

serialize_sample_identifier serialisiert die Daten eines Sample-Identifikators (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_sample_identifier in eine Datei geschrieben werden. Das Formmodell wird in dem Handle SampleIdentifier übergeben. Das serialisierte Formmodell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_sample_identifier wieder deserialisiert werden.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Parameter

▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_sample_identifier den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

prepare_sample_identifier, train_sample_identifier, add_sample_identifier_preparation_data, add_sample_identifier_training_data, set_sample_identifier_param

Nachfolger

clear_sample_identifier

Alternativen

write_sample_identifier

Modul

Matching

set_sample_identifier_object_info (: : SampleIdentifier, ObjectIdx, InfoName, InfoValue :)

Setzen eines Namens oder einer Beschreibung für ein Objekt eines Sample-Identifikators.

set_sample_identifier_object_info setzt Informationen, wie z.B. einen Namen oder eine Beschreibung, zu einen Objekt des Sample-Identifikators SampleIdentifier. Diese Informationen können mit dem Operator get_sample_identifier_object_info wieder abgefragt werden.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Um die Informationen für Vorbereitungsobjekte zu setzen, muss InfoName auf *'preparation_object_name'* gesetzt werden. Um die Informationen für Trainingsobjekte zu setzen, muss InfoName auf *'training_object_name'* gesetzt werden. Die Informationen selber werden mit dem Parameter InfoValue gesetzt.

Parameter

- ▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
  Handle des Sample-Identifikators.
- ▷ **ObjectIdx** (input_control) ..... integer(-array) ⇝ *integer* / string
  Index des Objekts für das Information gesetzt wird.
  Wertevorschläge: ObjectIdx ∈ {0, 1, 2, 3, 4, 5}
- ▷ **InfoName** (input_control) ..... attribute.name ⇝ *string*
  Gibt an, für welche Objektart die Information gesetzt wird.
  Defaultwert: 'training_object_name'
  Werteliste: InfoName ∈ {'preparation_object_name', 'training_object_name'}
- ▷ **InfoValue** (input_control) ..... attribute.value(-array) ⇝ *string*
  Information zu dem Objekt.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_sample_identifier_object_info den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_identifier_preparation_data, add_sample_identifier_training_data

Siehe auch

create_sample_identifier, prepare_sample_identifier, train_sample_identifier, apply_sample_identifier, get_sample_identifier_object_info, set_sample_identifier_param, get_sample_identifier_param, get_sample_identifier_object_info, remove_sample_identifier_preparation_data, remove_sample_identifier_training_data, write_sample_identifier, read_sample_identifier, serialize_sample_identifier, deserialize_sample_identifier, clear_sample_identifier

Modul

Matching

set_sample_identifier_param (: : SampleIdentifier, GenParamName, GenParamValue :)

Setzen bestimmter Parameter eines Sample-Identifikators.

Mit dem Operator set_sample_identifier_param können bestimmte Parameter des Sample-Identifikators SampleIdentifier gesetzt oder geändert werden.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Mit den folgenden Parametern kann das Verhalten des Sample-Identifikators SampleIdentifier beeinflusst werden. Die Parameter und ihre Werte können in GenParamName und GenParamValue übergeben werden. Die folgenden Parameter können gesetzt werden:

'use_color_info': Dieser Parameter legt fest, ob Farbinformation für die Identifikation verwendet wird. Ist *'use_color_info'* auf *'true'* gesetzt, so wird Farbinformation verwendet. In diesem Fall müssen die Bilder, die an apply_sample_identifier übergeben werden, RGB-Farbbilder sein. Ist *'use_color_info'* auf *'false'* gesetzt, wird keine Farbinformation verwendet. Farbinformation sollte verwendet werden, wenn die zu unterscheidenden Objekte unterschiedliche Farben haben und die Beleuchtung einigermaßen gut kontrolliert werden kann. In diesem Fall wird die Identifikation durch die Verwendung von Farbinformation robuster.

Es ist zu beachten, dass dieser Parameter, im Gegensatz zu dem generischen Parameter *'add_color_info'*, der nur mit create_sample_identifier gesetzt werden kann, nur das Verhalten des Operators apply_sample_identifier beeinflusst. Damit ist es z.B. möglich, den Sample-Identifikator unter Verwendung von Textur und Farbe zu trainieren, für die Identifikation bestimmter Objekte aber nur die Texturinformation zu verwenden.

Werteliste: 'true', 'false'

Defaultwert: So lange *'use_color_info'* nicht gesetzt wird, wird der Wert verwendet, der mit *'add_color_info'* in create_sample_identifier gesetzt wurde.

'use_texture_info': Dieser Parameter legt fest, ob Texturinformation für die Identifikation verwendet wird. Ist *'use_texture_info'* auf *'true'* gesetzt, so wird Texturinformation verwendet. Ist *'use_texture_info'* auf *'false'* gesetzt, wird keine Texturinformation verwendet. Normalerweise sollte *'use_texture_info'* auf *'true'* gesetzt sein, da die samplebasierte Identifikation nur mit texturierten Objekten funktioniert. Es ist zu beachten, dass mindestens einer der beiden generischen Parameter *'use_color_info'* und *'use_texture_info'* auf *'true'* gesetzt sein muss.

Es ist zu beachten, dass dieser Parameter, im Gegensatz zu dem generischen Parameter *'add_texture_info'*, der nur mit create_sample_identifier gesetzt werden kann, nur das Verhalten des Operators apply_sample_identifier beeinflusst.

Werteliste: 'true', 'false'

Defaultwert: So lange *'use_texture_info'* nicht gesetzt wird, wird der Wert verwendet, der mit *'add_texture_info'* in create_sample_identifier gesetzt wurde.

'image_resize_method': Siehe create_sample_identifier für eine Beschreibung dieses Parameters.

Werteliste: 'none', 'scale_factor', 'subsampling_step', 'image_area'

Defaultwert: Wenn *'image_resize_method'* nicht explizit mit diesem Operator gesetzt wird, wird der Wert verwendet, der mit create_sample_identifier gesetzt wurde.

'image_resize_value': Siehe create_sample_identifier für eine Beschreibung dieses Parameters.

Wertevorschläge: 0.25, 0.5, 1.0, 2, 3, 4

Defaultwert: Wenn *'image_resize_value'* nicht explizit mit diesem Operator gesetzt wird, wird der Wert verwendet, der mit create_sample_identifier gesetzt wurde.

'rating_method': Siehe apply_sample_identifier für eine Beschreibung dieses Parameters, der ausschließlich von apply_sample_identifier verwendet wird.

Werteliste: 'distance', 'score', 'score_single'

Defaultwert: 'score'

Parameter

▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

▷ **GenParamName** (input_control) ..... attribute.name ⇝ *string*
Parametername.
Defaultwert: 'rating_method'
Werteliste: GenParamName ∈ {'use_texture_info', 'use_color_info', 'image_resize_method', 'image_resize_value', 'rating_method'}

▷ **GenParamValue** (input_control) . . . . . attribute.value ⇝ *real* / integer / string
Parameterwert.
Defaultwert: 'score_single'
Werteliste: GenParamValue ∈ {'true', 'false', 'none', 'scale_factor', 'subsampling_step', 'distance', 'score', 'score_single', 'image_area', 0.25, 0.5, 0.75, 1.0, 2, 3, 4}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_sample_identifier_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_sample_identifier, read_sample_identifier

Alternativen

create_sample_identifier

Siehe auch

add_sample_identifier_preparation_data, prepare_sample_identifier, add_sample_identifier_training_data, train_sample_identifier, apply_sample_identifier, get_sample_identifier_param, get_sample_identifier_object_info, remove_sample_identifier_preparation_data, remove_sample_identifier_training_data, write_sample_identifier, serialize_sample_identifier, deserialize_sample_identifier, set_sample_identifier_object_info

Modul

Matching

train_sample_identifier (: : SampleIdentifier, GenParamName, GenParamValue :)

Trainieren eines Sample-Identifikators.

train_sample_identifier führt das Training des Sample-Identifikators SampleIdentifier durch. Hierzu werden die Trainingsdaten verwendet, die mit add_sample_identifier_training_data gesetzt wurden.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Um Objekte identifizieren zu können, muss der Sample-Identifikator SampleIdentifier mit einem repräsentativen Satz von Samplebildern der zu unterscheidenden Objekte trainiert werden (siehe add_sample_identifier_training_data).

Zum Training des Sample-Identifikators SampleIdentifier müssen diesem zuerst alle relevanten Samplebilder der zu unterscheidenden Objekte mit add_sample_identifier_training_data hinzugefügt werden. Alternativ dazu können die Vorbereitungsdaten wiederverwendet werden. Siehe prepare_sample_identifier für eine Beschreibung, wie die Vorbereitungsdaten für das Training des Sample-Identifikators wiederverwendet werden können. Dann wird das Training des Sample-Identifikators durch den Aufruf des Operators train_sample_identifier durchgeführt.

Das Training selbst läuft sehr schnell ab und es benötigt auch nur wenig Speicher. Daher ist es sehr gut möglich, den SampleIdentifier neu zu trainieren, z.B., wenn neue Objekte hinzugefügt oder bestehende Objekte gelöscht werden sollen.

Um einen bestehenden Sample-Identifikator mit neuen Objekten neu zu trainieren, müssen ihm lediglich die relevanten Samplebilder mit dem Operator add_sample_identifier_training_data hinzugefügt werden. Anschließend kann der Sample-Identifikator mit dem Operator train_sample_identifier neu trainiert

werden. Bestehende Objekte können mit dem Operator remove_sample_identifier_training_data entfernt werden. Anschließend kann der Sample-Identifikator mit dem Operator train_sample_identifier neu trainiert werden.

Das Training des Sample-Identifikators kann nur dann durchgeführt werden, wenn der Sample-Identifikator Trainingsdaten für mindestens zwei unterschiedliche Trainingsobjekte enthält.

Die generischen Parameter GenParamName und GenParamValue sind lediglich für eine mögliche zukünftige Verwendung vorhanden.

Parameter

- ▷ **SampleIdentifier** (input_control) ..... sample_identifier ⇝ *handle*
  Handle des Sample-Identifikators.
- ▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
  Parametername.
  Defaultwert: []
  Werteliste: GenParamName ∈ {[]}
- ▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *real* / string / integer
  Parameterwert.
  Defaultwert: []
  Werteliste: GenParamValue ∈ {[]}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert train_sample_identifier den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_identifier_training_data

Nachfolger

apply_sample_identifier, write_sample_identifier

Alternativen

read_sample_identifier

Siehe auch

create_sample_identifier, add_sample_identifier_preparation_data, prepare_sample_identifier, apply_sample_identifier, set_sample_identifier_param, get_sample_identifier_param, get_sample_identifier_object_info, remove_sample_identifier_preparation_data, remove_sample_identifier_training_data, serialize_sample_identifier, deserialize_sample_identifier, clear_sample_identifier, set_sample_identifier_object_info

Modul

Matching

write_sample_identifier (: : SampleIdentifier, FileName :)

Schreiben eines Sample-Identifikators in eine Datei.

write_sample_identifier schreibt einen Sample-Identifikator in die Datei FileName. Der Sample-Identifikator kann mit read_sample_identifier wieder eingelesen werden. Die Dateiendung für den Sample-Identifikator ist in HALCON 'sid'.

Das Konzept der samplebasierten Identifikation ist bei der Einleitung zu Kapitel Identifizierung / Samplebasiert beschrieben.

Parameter

▷ **SampleIdentifier** (input_control) . . . . . sample_identifier ⇝ *handle*
Handle des Sample-Identifikators.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.sid`

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert `write_sample_identifier` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`prepare_sample_identifier`, `train_sample_identifier`, `add_sample_identifier_preparation_data`, `add_sample_identifier_training_data`, `set_sample_identifier_param`

Nachfolger

`clear_sample_identifier`

Alternativen

`serialize_sample_identifier`

Siehe auch

`create_sample_identifier`, `prepare_sample_identifier`, `train_sample_identifier`, `apply_sample_identifier`, `get_sample_identifier_param`, `get_sample_identifier_object_info`, `remove_sample_identifier_preparation_data`, `remove_sample_identifier_training_data`, `read_sample_identifier`, `deserialize_sample_identifier`, `set_sample_identifier_object_info`

Modul

Matching

Kapitel 13

Inspektion

13.1 Kleberaupeninspektion

apply_bead_inspection_model (Image : LeftContour, RightContour, ErrorSegment : BeadInspectionModel : ErrorType)

Inspizieren einer Kleberaupe in einem Bild.

Der Operator apply_bead_inspection_model inspiziert die durch das Modell BeadInspectionModel definierte Kleberaupe im Bild Image. Gefundene Fehler werden als Kontursegmente in ErrorSegment zurückgegeben. Pro Segment wird in ErrorType der Typ des gefundenen Fehlers zurückgegeben. Die folgenden Fehler können auftreten (siehe auch create_bead_inspection_model):

'no bead': Für diese Position auf dem Modell konnte keine Kleberaupe gefunden werden.

'incorrect position': Das Zentrum der gefundenen Kleberaupe ist an dieser Postion weiter von der Modellkontur entfernt als PositionTolerance erlaubt.

'too thin': Die Kleberaupe ist dünner als erlaubt (siehe TargetThickness und ThicknessTolerance).

'too thick': Die Kleberaupe ist breiter als erlaubt (siehe TargetThickness und ThicknessTolerance).

Die Werte der generischen Parameter, die mit create_bead_inspection_model oder set_bead_inspection_param gesetzt wurden können die Ergebnisse der Inspektion stark beeinflussen. Siehe dazu auch create_bead_inspection_model.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Bild mit der zu inspizierenden Kleberaupe.
- ▷ **LeftContour** (output_object) ..... xld-array ⇝ *object*
  Detektierte linke Kontur.
- ▷ **RightContour** (output_object) ..... xld-array ⇝ *object*
  Detektierte rechte Kontur.
- ▷ **ErrorSegment** (output_object) ..... xld-array ⇝ *object*
  Detektierte Fehlersegmente.
- ▷ **BeadInspectionModel** (input_control) ..... bead_inspection_model ⇝ *handle*
  Handle des zu verwendenden Modells.
- ▷ **ErrorType** (output_control) ..... string-array ⇝ *string*
  Typ der gefundenen Fehler.

Ergebnis

Der Operator apply_bead_inspection_model gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BeadInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_bead_inspection_model`, `set_bead_inspection_param`

Nachfolger

`set_bead_inspection_param`, `clear_bead_inspection_model`

Siehe auch

`get_bead_inspection_param`, `create_bead_inspection_model`, `set_bead_inspection_param`

Modul

2D Metrology

`clear_bead_inspection_model` (: : BeadInspectionModel :)

Freigeben eines Modells zur Kleberaupeninspektion.

Der Operator `clear_bead_inspection_model` gibt das Modell `BeadInspectionModel` frei, welches von `create_bead_inspection_model` erzeugt wurde. Aller vom Modell verwendeter Speicher wird freigegeben. Nach dem Aufruf von `clear_bead_inspection_model` ist `BeadInspectionModel` ungültig.

Parameter

▷ **`BeadInspectionModel`** (input_control) ..... bead_inspection_model ⇝ *handle*
Handle des Modells zur Kleberaupeninspektion.

Ergebnis

Der Operator `clear_bead_inspection_model` gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BeadInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_bead_inspection_model`

Siehe auch

`create_bead_inspection_model`

Modul

2D Metrology

create_bead_inspection_model (BeadContour : : TargetThickness, ThicknessTolerance, PositionTolerance, Polarity, GenParamName, GenParamValue : BeadInspectionModel)

Erzeugen eines Modells zur Inspektion von Kleberaupen.

create_bead_inspection_model erzeugt ein neues Modell zur Inspektion von Kleberaupen.

Grundprinzip der Inspektion von Kleberaupen

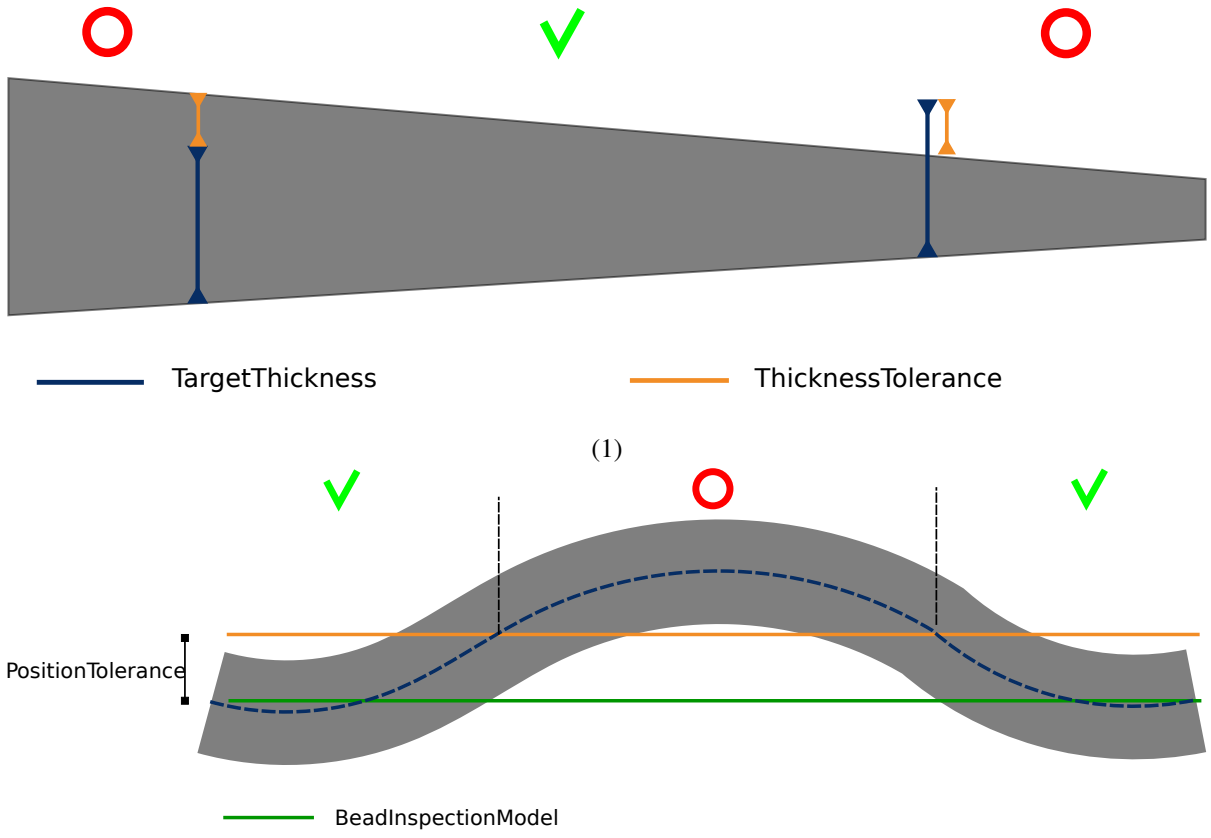
Die Operatoren zur Kleberaupeninspektion erlauben es eine Referenzkontur festzulegen, die die Position und Form einer Kleberaupe repräsentiert. In Aufnahmen entsprechender Kleberaupen kann damit verifiziert werden, ob sich eine Kleberaupe an den von der Referenzkontur festgelegten Positionen befindet und ob sie von akzeptabler Breite ist. Diese Bilder müssen bereits zur Modellkontur ausgerichtet sein.

Der Operator create_bead_inspection_model gibt einen Handle auf das Modell in BeadInspectionModel zurück, welcher später benutzt werden kann um weitere Operationen damit durchzuführen. Das beinhaltet das Setzen von Parametern und das Durchführen der Inspektion.

Modellkontur

Um Kleberaupen inspizieren zu können, wird eine Kontur in BeadContour benötigt, die als Modell dient. So eine Kontur kann beispielsweise mit den Operatoren gen_contour_polygon_xld oder gen_contour_nurbs_xld erzeugt werden. Mit draw_polygon oder draw_nurbs kann sie im Grafikfenster direkt über ein Referenzbild gezeichnet werden. Bestehen die zu inspizierenden Kleberaupen aus mehreren Teilen, müssen mehrere Modelle erzeugt werden.

Basisparameter



Die Basisparameter für ein Modell zur Inspektion von Kleberaupen

Die folgenden Parameter werden bei der Erzeugung des Modells benötigt:

TargetThickness: bestimmt die optimale Breite der Kleberaupe in Pixeln.

ThicknessTolerance: bestimmt die Toleranz bezüglich der Breite der Kleberaupe. Ein Teil einer Kleberaupe wird als korrekt identifiziert, wenn seine Breite innerhalb des folgenden Intervalls liegt: [TargetThickness-ThicknessTolerance, TargetThickness+ThicknessTolerance]

PositionTolerance: bestimmt die Toleranz bezüglich der Position der Kleberaupe in Pixeln. Die Position ist das Zentrum der gefundenen Kleberaupe. Ein Teil einer Kleberaupe wird als korrekt identifiziert, wenn die Abweichung dieser Position zur entsprechenden Position auf der Modellkontur kleiner als PositionTolerance ist.

Polarity: bestimmt die Polarität der Kleberaupe in den zu inspizierenden Bildern. *'light'* wenn die Kleberaupe heller im Bild erscheint als ihr Hintergrund, ansonsten *'dark'*.

Parameter für die Kantenextraktion

Die Inspektion von Kleberaupen verwendet Kantenextraktion um die Position der Kleberaupe im Bild zu bestimmen. Eine korrekte Wahl der Parameter für die Kantenextraktion führt zu besseren Ergebnissen. Die folgenden beiden Parameter können über die generischen Parameter gesetzt werden (siehe auch measure_pos):

'sigma': Wie in measure_pos ist *'sigma'* die Standardabweichung der zur Glättung verwendeten Gaußmaske. Die Wahl von *'sigma'* hängt von der Bildqualität ab, hauptsächlich vom Rauschanteil. Der Standard für *'sigma'* ist 2.0.

'threshold': Wie in measure_pos ist *'threshold'* die Schwelle für die Amplitude der extrahierten Kanten. Ein höherer Wert für *'threshold'* verringert die Extraktion von Kanten im Bildrauschen. *'threshold'* muss mindesten 1 betragen, der Standardwert ist 30.

Der Measure Assistent kann verwendet werden um geeignete Werte für diese beiden Parameter zu bestimmen.

Parameter

▷ **BeadContour** (input_object) ..... xld ⇝ *object*
XLD Kontur zur Festlegung der Position und Form der Kleberaupe.

▷ **TargetThickness** (input_control) ..... integer ⇝ *integer* / real
Optimale Breite der Kleberaupe.
Defaultwert: 50
Typischer Wertebereich: 5 ≤ TargetThickness

▷ **ThicknessTolerance** (input_control) ..... integer ⇝ *integer* / real
Toleranz bezüglich der Breite der Kleberaupe.
Defaultwert: 15
Typischer Wertebereich: 0 ≤ ThicknessTolerance

▷ **PositionTolerance** (input_control) ..... integer ⇝ *integer* / real
Toleranz bezüglich der Position der Kleberaupe.
Defaultwert: 15
Typischer Wertebereich: 0 ≤ PositionTolerance

▷ **Polarity** (input_control) ..... string ⇝ *string*
Polarität der Kleberaupe.
Defaultwert: 'light'
Werteliste: Polarity ∈ {'light', 'dark'}

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'sigma', 'threshold'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *integer* / real / string
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0.6, 1.0, 4.0, 5.0, 10.0, 50.0}

▷ **BeadInspectionModel** (output_control) ..... bead_inspection_model ⇝ *handle*
Handle für das neu erzeugte Modell.

Beispiel

```
* Read the image of the bead to be inspected.
read_image (Image, 'bead/adhesive_bead_01')
* Define the reference path of the adhesive beads.
gen_contour_nurbs_xld (ContourRef, \
                       [610.974,533.443,461.763,393.009,330.106,287.683, \
```

```
                          270.129,265.74,265.74,300.849,331.569,376.917, \
                          438.357,489.557,539.294], \
                         [418.581,424.27,439.441,473.574,526.67,574.078, \
                          644.241,708.715,765.604,818.7,866.107,915.411, \
                          966.611,998.848,993.159], \
                         'auto', \
                         [15,15,15,15,15,15,15,15,15,15,15,15,15,15,15], \
                         3, 1, 5)
*
* Create the bead inspection model, identifying the bead's parameters
create_bead_inspection_model (ContourRef, 14, 7, 30, 'dark', [], [], \
                              BeadInspectionModel)
*
* Apply the bead inspection model to the image
apply_bead_inspection_model (Image, LeftContour, RightContour, \
                             ErrorSegment, BeadInspectionModel, \
                             ErrorType)
```

Ergebnis

Der Operator `create_bead_inspection_model` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BeadInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`gen_contour_nurbs_xld`, `gen_contour_polygon_xld`

Nachfolger

`apply_bead_inspection_model`, `set_bead_inspection_param`

Siehe auch

`clear_bead_inspection_model`

Modul

2D Metrology

get_bead_inspection_param (: : BeadInspectionModel, GenParamName : GenParamValue)

Abfragen der Parameter eines Modells zur Inspektion von Kleberaupen.

Mit dem Operator `get_bead_inspection_param` können die Werte der Parameter des Modells `BeadInspectionModel` abgefragt werden.

Es können alle Parameter abgefragt werden, die mit `set_bead_inspection_param` gesetzt werden können. Mehrere Parameter können mit einem einzigen Aufruf abgefragt werden.

Parameter

▷ **BeadInspectionModel** (input_control) . . . . . bead_inspection_model ⇝ *handle*
Handle des Modells zur Kleberaupeninspektion.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name des abzufragenden Parameters.
Defaultwert: 'target_thickness'
Werteliste: `GenParamName` ∈ {'target_thickness', 'thickness_tolerance', 'position_tolerance', 'polarity', 'sigma', 'threshold'}

▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *integer* / real / string
Wert des Parameters.

Ergebnis

Der Operator `get_bead_inspection_param` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_bead_inspection_model`, `set_bead_inspection_param`

Nachfolger

`apply_bead_inspection_model`, `set_bead_inspection_param`

Siehe auch

`create_bead_inspection_model`, `set_bead_inspection_param`

Modul

2D Metrology

set_bead_inspection_param (: : BeadInspectionModel, GenParamName, GenParamValue :)

Setzen von Parametern des Modells zur Inspektion von Kleberaupen.

Mit dem Operator `set_bead_inspection_param` können die Parameter eines Modells zur Kleberaupeninspektion für eine spezielle Inspektionsaufgabe angepasst werden. Alle Parameter ausgenommen der Modellkontur können mit diesem Operator gesetzt werden. Die aktuell gesetzten Parameterwerte können mit dem Operator `get_bead_inspection_param` abgefragt werden.

Zu setzende Parameter

Die folgenden Parameter können gesetzt werden:

'target_thickness': setzt den Parameter `TargetThickness`.

'thickness_tolerance': setzt den Parameter `ThicknessTolerance`.

'position_tolerance': setzt den Parameter `PositionTolerance`.

'polarity': setzt den Parameter `Polarity`.

'sigma'

'threshold'

Diese Parameter sind ausführlich in der Referenz von `create_bead_inspection_model` beschrieben.

Parameter

▷ **BeadInspectionModel** (input_control) . . . . . bead_inspection_model ⇝ *handle*
Handle des Modells zur Kleberaupeninspektion.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name des zu setzenden Parameters.
Defaultwert: 'target_thickness'
Werteliste: `GenParamName` ∈ {'target_thickness', 'thickness_tolerance', 'position_tolerance', 'polarity', 'sigma', 'threshold'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Neuer Parameterwert.
Defaultwert: 40
Wertevorschläge: `GenParamValue` ∈ {40, 10, 15, 0.6, 1.0, 4.0, 5.0, 'light', 'dark'}

Ergebnis

Der Operator `set_bead_inspection_param` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BeadInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_bead_inspection_model`

Nachfolger

`apply_bead_inspection_model`, `get_bead_inspection_param`

Alternativen

`create_bead_inspection_model`

Siehe auch

`create_bead_inspection_model`, `clear_bead_inspection_model`

Modul

2D Metrology

13.2 OCV

close_ocv (: : OCVHandle :)

Freigeben eines OCV-Tools.

`close_ocv` gibt ein OCV-Tool frei, das mit `create_ocv_proj` oder `read_ocv` angelegt wurde. Das Handle ist nach dem Aufruf ungültig.

Parameter

▷ **OCVHandle** (input_control) . . . . . ocv ⇝ *handle*
Handle des OCV-Tools, das freigegeben werden soll.

Beispiel

```
read_ocv("ocv_file",&ocv_handle);
for (i=0; i<1000; i++)
{
  grab_image_async(&Image,fg_handle,-1);
  reduce_domain(Image,ROI,&Pattern);
  do_ocv_simple(Pattern,ocv_handle,"A",
                "true","true","false","true",10,
                &Quality);
}
close_ocv(ocv_handle);
```

Ergebnis

close_ocv liefert den Wert TRUE, falls das Handle gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCVHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

read_ocv, create_ocv_proj

Siehe auch

close_ocr

Modul

OCR/OCV

create_ocv_proj (: : PatternNames : OCVHandle)

Erzeugen eines neuen OCV-Tools basierend auf Grauwertprojektionen.

create_ocv_proj erzeugt ein neues OCV-Tool. Dieses kann danach mit „Gut-Zeichen“ trainiert werden. Hierzu steht der Operator traind_ocv_proj zur Verfügung. Dieser Operator wird daher normalerweise im Anschluss an create_ocv_proj aufgerufen.

Das Verfahren des Objektvergleiches basiert auf den Grauwertprojektionen. Hierzu werden beim Training und beim Vergleich die Grauwerte des Musters horizontal und vertikal aufsummiert. Die so erhaltenen Daten werden dann miteinander verglichen. Um so ähnlicher sich die Projektionen sind, um so höher ist die Güte des Objektes.

Übergeben werden die Namen der Zeichen (PatternNames), die trainiert werden sollen. Die Anzahl der Zeichen und auch die zu vergebenden Namen können frei gewählt werden. Im einfachsten Fall wird nur ein Name angegeben. Die Namen werden beim Training und in der Anwendung (Bewertung von Zeichen) zur Identifikation der „Gut-Zeichen“ verwendet. Die Namen können nicht geändert werden. Es können mehr Namen vergeben werden als später trainiert werden. Geschlossen wird das Tool mit dem Operator close_ocv. Hierdurch wird der Speicher wieder freigegeben.

Parameter

▷ **PatternNames** (input_control) ..... string(-array) ⇝ *string*
Liste von Namen der zu lernenden Zeichen.
Defaultwert: 'a'

▷ **OCVHandle** (output_control) ..... ocv ⇝ *handle*
Handle des erzeugten OCV-Tools.

Beispiel

```
create_ocv_proj("A",&ocv_handle);
draw_region(&ROI,window_handle);
reduce_domain(Image,ROI,&Sample);
traind_ocv_proj(Sample,ocv_handle,"A","single");
```

Ergebnis

`create_ocv_proj` liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`traind_ocv_proj`, `write_ocv`, `close_ocv`

Alternativen

`read_ocv`

Siehe auch

`create_ocr_class_box`

Modul

OCR/OCV

deserialize_ocv (: : SerializedItemHandle : OCVHandle)

Deserialisieren eines serialisierten OCV-Tools.

`deserialize_ocv` deserialisiert ein OCV-Tool, welches mit `serialize_ocv` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte OCV-Tool wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten OCV-Tool mit dem Handle `OCVHandle` gespeichert. Nach dem Deserialisieren kann entweder das Training fortgesetzt werden (`traind_ocv_proj`), sofern dies noch nicht abgeschlossen wurde, oder es kann direkt eine Bewertung von Zeichen ausgeführt werden (`do_ocv_simple`).

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **OCVHandle** (output_control) ..... ocv ⇝ *handle*
Handle des OCV-Tools.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_ocv` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_ocv`

Nachfolger

`do_ocv_simple`, `close_ocv`

Modul

OCR/OCV

do_ocv_simple (Pattern : : OCVHandle, PatternName, AdaptPos, AdaptSize, AdaptAngle, AdaptGray, Threshold : Quality)

Bewerten eines Zeichens mit einem OCV-Tool.

`do_ocv_simple` bewertet das übergebene Zeichen (`Pattern`). Das Zeichen muss vorher mit dem OCV-Tool trainiert worden sein. Die Region sollte dabei in etwa die gleiche (relative) Ausdehnung und Form haben wie beim Training. Zur Kennzeichnung des Zeichens wird der gleiche Name wie beim Training in `PatternName` übergeben. Über die nächsten vier Parameter kann dann Einfluss auf die automatische Anpassung genommen werden: `AdaptPos` und `AdaptSize` beziehen sich auf die Geometrie des Zeichens. Mit `AdaptPos` wird festgelegt, ob eine Verschiebung des Zeichens kompensiert werden soll. `AdaptSize` legt fest, ob eine Größenveränderung angepasst werden soll. `AdaptAngle` ist nicht implementiert. Der Parameter `AdaptGray` bezieht sich auf die Grauwertanpassung. Zum einen wird eine additive Grauwertänderung kompensiert, gleichzeitig wird eine multiplikative Grauwertänderung angepasst.

Der Parameter `Threshold` gibt den Mindestunterschied der Grauwerte an, der als Fehler interpretiert wird. Als Resultat ergibt sich dann der Prozentsatz der fehlerhaften Pixel. Wird mit dem Parameter `Threshold` eine Schwelle kleiner als 0 übergeben, besteht das Resultat stattdessen in der Summe aller Abweichungen normiert auf die Fläche der Region.

Das Ergebnis des Operators ist eine Bewertung des Zeichens. Dies ist der Unterschied der Grauwerte zwischen dem aktuellen Zeichen und dem Trainingszeichen nach den entsprechenden automatischen Anpassungen. Der Wert von `Quality` liegt zwischen 0 und 1. Der Wert 1 bedeutet, dass das Zeichen mit dem Muster identisch ist. Der Wert 0 steht für eine sehr große Abweichung.

Parameter

- ▷ **Pattern** (input_object) . . . . . singlechannelimage(-array) ⇝ *object : byte*
  Zu bewertendes Zeichen.
- ▷ **OCVHandle** (input_control) . . . . . ocv ⇝ *handle*
  Handle des OCV-Tools.
- ▷ **PatternName** (input_control) . . . . . string(-array) ⇝ *string*
  Name des Zeichens.
  Defaultwert: 'a'
- ▷ **AdaptPos** (input_control) . . . . . string ⇝ *string*
  Anpassung an horizontale und vertikale Verschiebung.
  Defaultwert: 'true'
  Werteliste: `AdaptPos` ∈ {'true', 'false'}
- ▷ **AdaptSize** (input_control) . . . . . string ⇝ *string*
  Anpassung an horizontale und vertikale Größenveränderung.
  Defaultwert: 'true'
  Werteliste: `AdaptSize` ∈ {'true', 'false'}
- ▷ **AdaptAngle** (input_control) . . . . . string ⇝ *string*
  Anpassung der Orientierung (nicht implementiert).
  Defaultwert: 'false'
  Werteliste: `AdaptAngle` ∈ {'false'}
- ▷ **AdaptGray** (input_control) . . . . . string ⇝ *string*
  Anpassung an additive und multiplikative Grauwertänderungen.
  Defaultwert: 'true'
  Werteliste: `AdaptGray` ∈ {'true', 'false'}

▷ **Threshold** (input_control) . . . . . number ⇝ *real*
Mindestdifferenz der Muster.
Defaultwert: 10
Wertevorschläge: `Threshold` ∈ {-1.0, 0.0, 1.0, 5.0, 10.0, 15.0, 20.0, 30.0, 40.0, 50.0, 60.0, 80.0, 100.0, 150.0}

▷ **Quality** (output_control) . . . . . real(-array) ⇝ *real*
Bewertung des Zeichens.
Typischer Wertebereich: 0.0 ≤ `Quality` ≤ 1.0

Ergebnis

`do_ocv_simple` liefert den Wert TRUE, falls das Handle und die Vergleichsdaten gültig sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`traind_ocr_class_box`, `trainf_ocr_class_box`, `read_ocv`, `threshold`, `connection`, `select_shape`

Nachfolger

`close_ocv`

Siehe auch

`create_ocv_proj`

Modul

OCR/OCV

read_ocv (: : FileName : OCVHandle)

Einlesen eines OCV-Tools von Datei.

`read_ocv` liest ein OCV-Tool von Datei. Der Zustand des Tools ist identisch mit dem Zustand beim Speichern (`write_ocv`). Nach dem Lesen kann entweder das Training fortgesetzt werden (`traind_ocv_proj`), sofern dies noch nicht abgeschlossen wurde, oder es kann direkt eine Bewertung von Zeichen ausgeführt werden (`do_ocv_simple`).

Als Extension der Datei wird '.ocv' verwendet. Falls diese bei dem Dateinamen nicht angegeben ist, wird sie automatisch angefügt.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei die gelesen werden soll.
Defaultwert: 'test_ocv'
Dateiendung: `.ocv`

▷ **OCVHandle** (output_control) . . . . . ocv ⇝ *handle*
Handle des gelesenen OCV-Tools.

Beispiel

```
read_ocv("ocv_file",&ocv_handle);
for (i=0; i<1000; i++)
{
  grab_image_async(&Image,fg_handle,-1);
  reduce_domain(Image,ROI,&Pattern);
  do_ocv_simple(Pattern,ocv_handle,"A",
                "true","true","false","true",10,
```

```
                &Quality);
}
close_ocv(ocv_handle);
```

Ergebnis

read_ocv liefert den Wert TRUE, falls die Datei korrekt ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

write_ocv

Nachfolger

do_ocv_simple, close_ocv

Siehe auch

read_ocr

Modul

OCR/OCV

serialize_ocv (: : OCVHandle : SerializedItemHandle)

Serialisieren eines OCV-Tools.

serialize_ocv serialisiert die Daten eines OCV-Tools (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_ocv in eine Datei geschrieben werden. Das OCV-Tool wird in dem Handle OCVHandle übergeben. Das serialisierte OCV-Tool wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_ocv wieder deserialisiert werden.

Parameter

▷ **OCVHandle** (input_control) ..... ocv ⇝ *handle*
Handle des OCV-Tools.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_ocv den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

traind_ocv_proj

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_ocv

Modul

OCR/OCV

traind_ocv_proj (Pattern : : OCVHandle, Name, Mode :)

Training eines OCV-Tools mit Zeichen.

traind_ocv_proj trainiert ein OCV-Tool das mit create_ocv_proj erzeugt oder mit read_ocv eingelesen wurde. Zum Training werden dem System die Zeichen als Regionen mit dem zugehörigen Bild übergeben. Es ist zu beachten, dass die Region nicht nur den Vordergrund des Zeichens (z.B. die dunklen Pixel) beinhaltet, sondern auch Pixel von der Umgebung des Zeichens. Dies kann z.B. das umschließende Rechteck des Zeichens sein. Ohne diesen Kontext kann ein Zeichen nicht beurteilt werden.

Falls mehr als ein Muster gelernt werden soll, kann das Training durch einen Aufruf mit mehreren Zeichen zusammen mit einem Tupel der zugehörigen Namen oder durch mehrfache Aufrufe des Operators mit jeweils einem Muster realisiert werden. Das Verhalten, d.h. die Bewertung eines Zeichens, wird hierdurch nicht beeinflusst. Die Laufzeit des Trainings wird jedoch bei einer Aufteilung auf mehrere Aufrufe etwas höher sein.

Parameter

- ▷ **Pattern** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Zu trainierende Zeichen.
- ▷ **OCVHandle** (input_control) ..... ocv ⇝ *handle*
  Handle des zu trainierenden OCV-Tools.
- ▷ **Name** (input_control) ..... string(-array) ⇝ *string*
  Name(n) der zu untersuchenden Objekte.
  Defaultwert: 'a'
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus für Training (nur ein Modus verfügbar).
  Defaultwert: 'single'
  Werteliste: Mode ∈ {'single'}

Beispiel

```
create_ocv_proj("A",&ocv_handle);
draw_region(&ROI,window_handle);
reduce_domain(Image,ROI,&Sample);
traind_ocv_proj(Sample,ocv_handle,"A","single");
```

Ergebnis

traind_ocv_proj liefert den Wert TRUE, falls das Handle und die Trainingsdaten gültig sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCVHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

write_ocr_trainf, create_ocv_proj, read_ocv, threshold, connection, select_shape

Nachfolger

close_ocv

Siehe auch

traind_ocr_class_box

Modul

OCR/OCV

write_ocv (: : OCVHandle, FileName :)

Sichern eines neuen OCV-Tools in eine Datei.

write_ocv schreibt ein OCV-Tool in eine Datei. Dies kann nach dem Operator traind_ocv_proj zum Sichern des Zustandes verwendet werden. Das OCV-Tool wird durch das Schreiben nicht verändert. Die Datei kann später mit read_ocv wieder eingelesen werden. Als Extension wird '.ocv' verwendet. Wenn diese Extension nicht mit dem Dateinamen angegeben ist, wird sie automatisch angefügt.

Parameter

▷ **OCVHandle** (input_control) ..... ocv ⇝ *handle*
Handle des zu schreibenden OCV-Tools.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei, in die das Tool gesichert werden soll.
Defaultwert: 'test_ocv'
Dateiendung: .ocv

Ergebnis

write_ocv liefert den Wert TRUE, falls die Daten korrekt sind und die Datei geschrieben werden kann. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

traind_ocv_proj

Nachfolger

close_ocv

Siehe auch

write_ocr

Modul

OCR/OCV

13.3 Strukturiertes Licht

Dieses Kapitel beinhaltet Operatoren für die Verwendung von strukturiertem Licht.

Konzept von strukturiertem Licht

Das grundlegende Konzept von strukturiertem Licht beruht auf der Darstellung bekannter Muster auf einem Monitor oder Projektor. Die Art und Weise, wie diese Muster nach dem Auftreffen auf eine Oberfläche in der Szene erscheinen, lässt eine weitere Analyse der Oberfläche zu.

Eine Voraussetzung für die Anwendung von strukturiertem Licht ist das Erzeugen eines Modells mittels create_structured_light_model. Der Modelltyp wird mit ModelType spezifiziert. Zurzeit wird nur der Typ *'deflectometry'* unterstützt. Unter Deflektometrie versteht man die Analyse der Reflexionen bekannter Muster von spiegelnden oder halbmatten Oberflächen. Für jedes Muster Bild, das auf dem Bildschirm angezeigt wird, wird ein entsprechendes Kamerabild von der Reflexion aufgenommen. Verformungen der Muster lassen auf einen Oberflächendefekt schließen.

Im Folgenden werden kurz die Schritte beschrieben, die zur Defekterkennung auf spiegelnden Oberflächen notwendig sind.

Erzeugen eines Modells für strukturiertes Licht: Im ersten Schritt wird das Modell für strukturiertes Licht mit

- create_structured_light_model

erzeugt oder mit

- read_structured_light_model

eingelesen.

Setzen der Modellparameter: Die unterschiedlichen Modellparameter können mit set_structured_light_model_param gesetzt oder mit get_structured_light_model_param abgefragt werden. Die Parameter *'pattern_width'*, *'pattern_height'*, *'pattern_orientation'* und *'pattern_type'* spezifizieren, mitsamt den Parametern *'min_stripe_width'* und *'single_stripe_width'* die Vorgaben der zu erzeugenden Muster-Bilder, welche auf die spiegelnde Oberfläche projiziert werden. Schließlich kann der Parameter *'persistence'* gesetzt werden, um die Zwischenergebnisse zu debuggen.

Erzeugen der Muster-Bilder: Die Muster-Bilder werden mit gen_structured_light_pattern erzeugt, nachdem alle relevanten Parameter gesetzt wurden. Es ist sicherzustellen, dass die erzeugten Muster-Bilder an den Bedarf des jeweiligen Setups angepasst sind.

Projizieren der Muster und Aufnahme der Kamerabilder: In dieser Phase werden die Muster-Bilder auf dem Monitor angezeigt und von der spiegelnden Oberfläche reflektiert. Jedes reflektierte Muster wird von der Kamera aufgenommen.

Dekodieren der aufgenommenen Bilder: Die erzeugten Kamerabilder CameraImages können mit decode_structured_light_pattern dekodiert werden. Nach Aufruf dieses Operators werden die Korrespondenzbilder erzeugt und in dem Modell StructuredLightModel gespeichert.

Abrufen der Ergebnisse: Nun können die dekodierten Korrespondenzbilder *'correspondence_image'* sowie die anderen Ergebnisse mit get_structured_light_object abgefragt werden. Das Defektbild *'defect_image'* kann erzeugt werden und mit get_structured_light_object abgerufen werden. Weitere Details über die verschiedenen Objekte, die abgefragt werden können, finden sich in der Referenzdokumentation des Operators.

Weitere Operatoren

Das Modell für strukturiertes Licht bietet diverse Operatoren, die dabei helfen, auf Modellparameter zuzugreifen und diese zu aktualisieren.

Mit write_structured_light_model kann das Modell für strukturiertes Licht in einer Datei gespeichert werden. Es ist sicherzustellen, dass zuvor generierte Muster-Bilder nicht in dieser Datei gespeichert werden. Mit read_structured_light_model kann die Modell-Datei gelesen werden.

Des Weiteren ist es möglich, das Modell mit den Operatoren serialize_structured_light_model und deserialize_structured_light_model zu serialisieren und zu deserialisieren.

Weiterführende Informationen

Weitere Details zu strukturiertem Licht finden sich im „Solution Guide Basics".

clear_structured_light_model (: : StructuredLightModel :)

Löschen eines Modells für strukturiertes Licht und Freigabe des verwendeten Speichers.

Der Operator clear_structured_light_model löscht ein Modell für strukturiertes Licht, das mit create_structured_light_model erzeugt wurde. Aller vom Modell verwendeter Speicher wird freigegeben. Das Handle des Modells wird in StructuredLightModel übergeben. Nach dem Aufruf von clear_structured_light_model ist das Handle ungültig.

Das Konzept des strukturierten Lichts sowie die unterstützen Anwendungsbereiche werden in der Einleitung des Kapitels Inspektion / Strukturiertes Licht beschrieben.

Parameter

▷ **StructuredLightModel** (input_control) . . . . . structured_light_model(-array) ⇝ *handle*
Handle des Modells für strukturiertes Licht.

Ergebnis

Der Operator `clear_structured_light_model` gibt TRUE zurück, falls ein gültiger Handle übergeben wurde und der entsprechende Handle korrekt freigegeben werden kann. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- StructuredLightModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

`create_structured_light_model`

Modul

3D Metrology

create_structured_light_model (
: : ModelType : StructuredLightModel)

Erzeugen eines Modells für strukturiertes Licht.

`create_structured_light_model` erzeugt ein neues Modell für strukturiertes Licht vom Typ `ModelType`. Aktuell ist nur der Typ *'deflectometry'* verfügbar.

Die Parameter des Modells können mit `get_structured_light_model_param` abgefragt und mit `set_structured_light_model_param` manipuliert werden.

Das Konzept des strukturierten Lichts sowie die unterstützen Anwendungsbereiche werden in der Einleitung des Kapitels Inspektion / Strukturiertes Licht beschrieben.

Parameter

▷ **ModelType** (input_control) . . . . . string ⇝ *string*
Typ des erzeugten Modells für strukturiertes Licht.
Defaultwert: 'deflectometry'

▷ **StructuredLightModel** (output_control) . . . . . structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

Beispiel

```
* Create the model
create_structured_light_model ('deflectometry', StructuredLightModel)
* Generate the patterns to project
gen_structured_light_pattern (PatternImages, StructuredLightModel)
* Decode the camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
```

Ergebnis

Der Operator `create_structured_light_model` gibt TRUE zurück, falls das Modell für strukturiertes Licht korrekt allokiert werden kann. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_structured_light_model_param`, `get_structured_light_model_param`

Modul

3D Metrology

decode_structured_light_pattern (
`CameraImages : : StructuredLightModel : )`

Dekodieren der aufgenommenen Kamerabilder bei einem Verfahren mit strukturiertem Licht.

`decode_structured_light_pattern` dekodiert die Kamerabilder `CameraImages`, die zuvor bei einem Verfahren mit strukturiertem Licht aufgenommen wurden. Die bei der Dekodierung erzeugten Korrespondenzbilder und andere Zwischenergebnisse werden im Model `StructuredLightModel` gespeichert und können im Anschluss mit dem Operator `get_structured_light_object` abgerufen werden.

Im Folgenden wird der Dekodiervorgang ausführlich erklärt:

Wie in `gen_structured_light_pattern` angemerkt, ist es im ersten Schritt notwendig zu entscheiden, ob ein Kamerapixel eine Region beobachtet, die einen weißen Streifen reflektiert oder einen schwarzen Streifen reflektiert. Um diesen Entscheidungsprozess zu vereinfachen, werden Normalisierungsbilder benutzt und ein lokal wechselnder Schwellenwert bestimmt, der variierende Reflexionsgrade des Objekts und variierende Lichtverhältnisse verarbeiten kann. Während des Dekodiervorgangs der aufgenommenen Kamerabilder werden alle Gray-Code-Bilder mit diesem vorberechneten Schwellenwert verglichen. Ein Kamerapixel wird als hell klassifiziert, wenn sein Grauwert größer oder gleich diesem Schwellenwert ist.

Ferner wird während des Dekodiervorgangs die Musterregion bestimmt. Die Segmentierung wird vom Parameter *'min_gray_difference'* (siehe `set_structured_light_model_param`) kontrolliert.

Angenommen, es wurden n Gray-Code-Bilder verarbeitet, erhält man für jedes Pixel einen binären n-bit Code. Aus dieser Folge lassen sich Zeile und Spalte des Monitors bis auf $'min_stripe_width'/2$ bestimmen.

Besteht das `StructuredLightModel` nicht nur aus Gray-Code-Bildern, sondern ist ein Hybrid-System mit Phasenbildern (see `gen_structured_light_pattern`), so werden im nächsten Schritt die Phasenbilder dekodiert. Das Ergebnis ist ein subpixelgenaues Korrespondenzbild zwischen Monitor- und Kamerakoordinaten, das die Information beinhaltet, welches Kamerapixel die Reflexion welches Monitorpixels beobachtet.

Wenn der Parameter *'pattern_type'* des Modells `StructuredLightModel` auf *'single_stripe'* gesetzt wird, wird im ersten Schritt des Dekodierprozesses entschieden, welcher Einzelstreifen sein Licht auf einen Kamerapixel wirft. Die Gray-Code-Folge und die Phase werden dann verwendet, um die Position innerhalb des gefundenen Einzelstreifens zu verfeinern.

Es kann vorkommen, dass die detektierte Gray-Code-Folge eines Pixels falsch ist. Dies kann zu Werten im Korrespondenzbild führen, die Monitor- Zeilen und -Spalten darstellen, die größer als die Monitor-Breite und -Höhe sind. Um diese Probleme zu vermeiden, ist der letzte Schritt des Dekodiervorgangs die Entfernung dieser Werte aus dem Korrespondenzbild.

Parameter

▷ **CameraImages** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
Aufgenommene Kamerabilder.

▷ **StructuredLightModel** (input_control) ..... structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

Beispiel

```
* Create the model
```

```
create_structured_light_model ('deflectometry', StructuredLightModel)
* Set the size of the monitor
set_structured_light_model_param (StructuredLightModel, \
                                  'pattern_width', 1600)
set_structured_light_model_param (StructuredLightModel, \
                                  'pattern_height', 1200)
* Set the smallest width of the stripes in the pattern
set_structured_light_model_param (StructuredLightModel, \
                                  'min_stripe_width', 8)
* Generate the patterns to project
gen_structured_light_pattern (PatternImages, StructuredLightModel)
* Set the expected black/white contrast in the region of interest
set_structured_light_model_param (StructuredLightModel, \
                                  'min_gray_difference', 70)
* Decode the camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
* Get the computed correspondences and defects
get_structured_light_object (CorrespondenceImages, StructuredLightModel, \
                             'correspondence_image')
set_structured_light_model_param (StructuredLightModel, 'derivative_sigma', \
                                  Sigma)
get_structured_light_object (DefectImage, StructuredLightModel, \
                             'defect_image')
```

Ergebnis

Der Operator `decode_structured_light_pattern` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- StructuredLightModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`gen_structured_light_pattern`

Nachfolger

`get_structured_light_object`

Siehe auch

`create_structured_light_model`, `set_structured_light_model_param`

Modul

3D Metrology

deserialize_structured_light_model (
: : SerializedItemHandle : StructuredLightModel)

Deserialisieren eines Modells für strukturiertes Licht.

`deserialize_structured_light_model` deserialisiert ein Modell für strukturiertes Licht, welches mit `serialize_structured_light_model` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Modell wird in dem

Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden im Modell StructuredLightModel gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **StructuredLightModel** (output_control) ..... structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

Beispiel

```
* Deserialize the model
deserialize_structured_light_model (SerializedItemHandle, \
                                    StructuredLightModel)
* Get a previously decoded result
get_structured_light_object (CorrespondenceImages, StructuredLightModel, \
                             'correspondence_image')
* Decode new camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
* Get the decoded result
get_structured_light_object (CorrespondenceImagesNew, StructuredLightModel, \
                             'correspondence_image')
```

Ergebnis

Der Operator deserialize_structured_light_model gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- StructuredLightModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_structured_light_model

Siehe auch

create_structured_light_model, write_structured_light_model, serialize_structured_light_model

Modul

3D Metrology

gen_structured_light_pattern (: PatternImages : StructuredLightModel :)

Erstellen der Muster-Bilder zur Anzeige bei einem Verfahren mit strukturiertem Licht.

gen_structured_light_pattern erzeugt die Muster-Bilder, die bei einem Verfahren mit strukturiertem Licht angezeigt werden. Verschiedene Parameter wie Breite und Höhe der Bilder, der Typ der Muster und die minimale Streifenbreite müssen zuvor mit dem Operator set_structured_light_model_param für das Modell StructuredLightModel festgelegt werden. Je nach den gesetzten Parametern in StructuredLightModel kann Anzahl und Aussehen der Muster-Bilder variieren.

Generell können die Muster-Bilder in vier Gruppen unterteilt werden:

- Normalisierungsbilder
- Gray-Code-Bilder
- Phasenbilder
- Einzelstreifen-Bilder

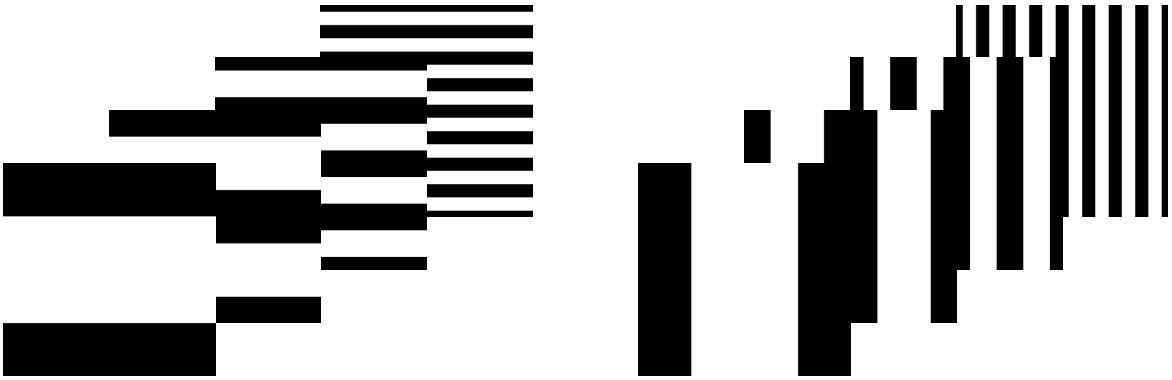
Allgemeine Information zu den Bildgruppen

Normalisierungsbilder:

Zum Dekodieren der Gray-Code-Bilder ist es notwendig zu entscheiden, ob ein Kamerapixel eine Region beobachtet, die einen weißen Streifen reflektiert oder einen schwarzen Streifen reflektiert. Um auch bei großer Variation des Reflexionsgrads der Oberfläche robust dekodieren zu können, werden Normalisierungsbilder benutzt um hell und dunkel beleuchtete Regionen besser zu identifizieren. Der einfachste, in den meisten Fällen aber auch ausreichende, Ansatz ist, ein vollständig schwarzes und ein vollständig weißes Bild zu erzeugen. Während des Dekodiervorgangs der aufgenommenen Kamerabilder (siehe `decode_structured_light_pattern`) werden dann alle Gray-Code-Bilder mit dem Mittelwert des dunklen und hellen Bildes verglichen. Ein Kamerapixel wird als hell klassifiziert, wenn sein Grauwert größer oder gleich seinem zuvor berechneten Mittelwert ist. Ein weiterer Ansatz besteht darin, für jedes erzeugte Gray-Code-Bild wie im nächsten Absatz beschrieben auch eine invertierte Version des Gray-Code-Bilds zu erzeugen. Ein Kamerapixel wird dann als hell klassifiziert, wenn sein Grauwert im ersten Bild größer als sein Grauwert im invertierten Bild ist.

Gray-Code-Bilder:

Vor anwendungsnahen Arbeiten wie dem Detektieren von Defekten ist der erste Schritt in jeder Anwendung von strukturiertem Licht, eine Abbildung zwischen den Kamerapixeln und der Pixelebene des Monitors zu finden. Dies wird durch Anzeigen einer Bildfolge erreicht, die die Zeilen und Spalten des Monitors eindeutig kodieren. Wegen einiger Vorteile gegenüber anderen Mustertypen werden typischerweise Gray-Code-Bilder zur Kodierung verwendet. Ein Beispiel für eine Abfolge vertikaler und horizontaler Gray-Code-Bilder findet sich in der folgenden Abbildung.



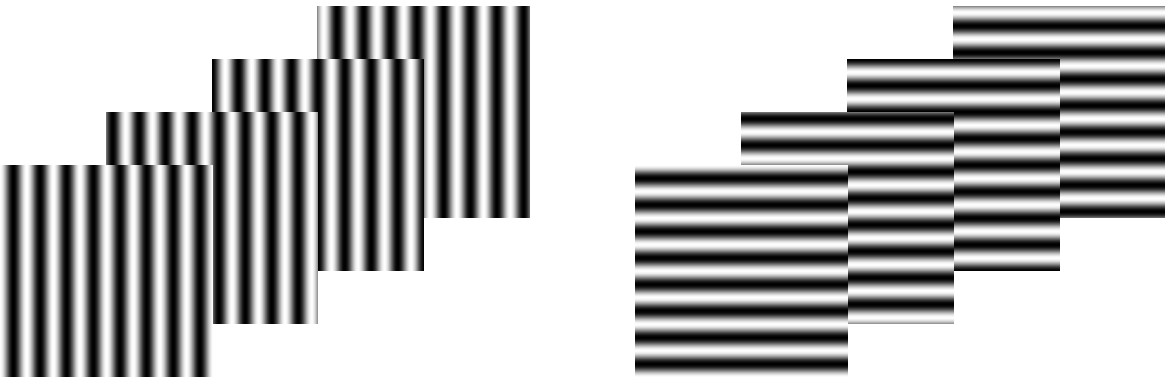
Bildfolge vertikaler und horizontaler Gray-Code-Bilder wie sie von `gen_structured_light_pattern` erzeugt werden. Die Muster werden vom Monitor typischerweise vom gröbsten Muster zum feinsten angezeigt, wobei sich die Streifenbreite jeweils halbiert.

Obwohl die Benutzung von Gray-Code-Bildern eine einfache und zweckmäßige Herangehensweise darstellt, weist sie gewisse Beschränkungen auf. Ein Nachteil tritt auf, wenn viele Gray-Code-Bilder benötigt werden, um eine für die praktische Anwendung nötige räumliche Auflösung zu erreichen. Bei zunehmend feinerer Streifenauflösung wird das korrekte Dekodieren wegen Unschärfeeffekten immer schwieriger. Die dritte Bildgruppe (Phasenbilder) kann erzeugt werden, um diese Beschränkung zu überwinden.

Phasenbilder:

Phasenbilder benutzen gewöhnlich periodische Muster wie Sinus- oder Kosinus-Wellen, um später einen Winkel des Phasendurchlaufs zu rekonstruieren. In Kombination mit den Gray-Code-Bildern liefern sie dann die Korrespondenzen zwischen Monitor- und Kamerakoordinaten.

Um eine robuste Dekodierung in Hinblick auf Variation des Reflexionsgrads der Oberfläche und Rauschanfälligkeit zu ermöglichen, werden pro Orientierung (vertikal oder horizontal) vier Phasenbilder erzeugt. Die Bilder sind Kosinus-Wellen mit Phasenverschiebungen von $k\pi/2$, wobei $k = 0, ..., 3$.

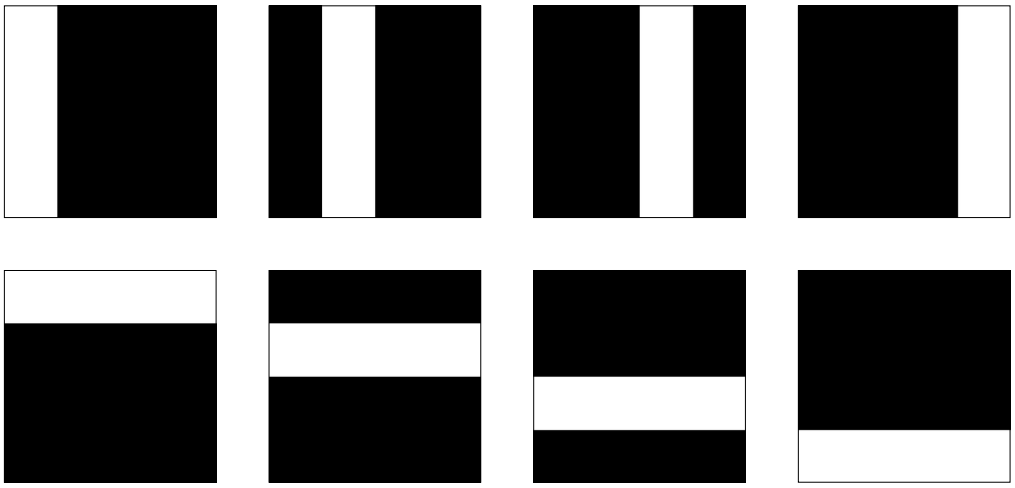


Bildfolge vertikaler und horizontaler Phasenbilder wie sie von `gen_structured_light_pattern` erzeugt werden.

Der Hauptvorteil der Phasenbilder gegenüber den Gray-Code-Bildern besteht in subpixelgenauen Korrespondenzen zwischen Monitor- und Kamerakoordinaten. Wegen der Periodizität ist die Kodierung nur eindeutig bis auf ganzzahlige Vielfache der Periodenlänge. Um eine eindeutige Kodierung über das ganze Bild zu erreichen, ist die Kombination mit Gray-Code-Bildern notwendig.

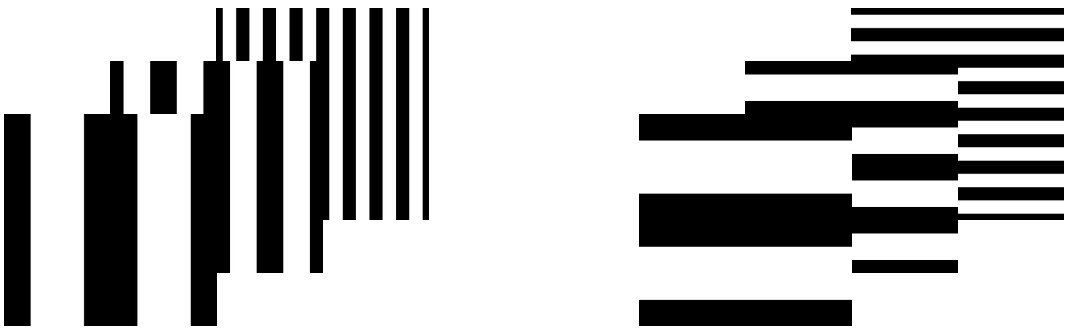
Einzelstreifen-Bilder:

Einzelstreifen-Bilder bestehen aus einem hellen Streifen auf dunklem Hintergrund. Der helle Streifen wird derart über die Bilder verschoben, dass jeder Monitorpixel in genau einem Streifen enthalten ist. Um zu entscheiden, welcher helle Streifen in dem vom Kamerapixel beobachteten Bereich reflektiert wird, wird während der Dekodierung der hellste Pixelwert innerhalb der Sequenz einzelner Streifenbilder gewählt. Dies ermöglicht eine Dekodierung, die trotz Schwankungen der Oberflächenreflexion robust ist.



Bildfolge vertikaler und horizontaler Streifenbilder wie sie von `gen_structured_light_pattern` erzeugt werden, wenn *'pattern_type'* auf *'single_stripe'* gesetzt wird.

Die Gray-Code-Folge und die Phase werden dann verwendet, um die Position innerhalb des gefundenen Einzelstreifens zu verfeinern. Somit nimmt die Anzahl der erzeugten Gray-Code-Bilder typischerweise etwas ab, während zusätzliche Einzelstreifen-Bilder erzeugt werden.



Bildfolge vertikaler und horizontaler Gray-Code-Bilder wie sie von gen_structured_light_pattern erzeugt werden, wenn *'pattern_type'* auf *'single_stripe'* gesetzt wird.

Information über die Erzeugung der Bildgruppen

Erzeugung der Normalisierungsbilder:

Wie oben beschrieben, bietet HALCON zwei verschiedene Arten von Normalisierungsbildern. Standardmäßig ist *'normalization'* auf *'global'* gesetzt, was zur Erzeugung eines vollständig schwarzen und eines vollständig weißen Bilds führt. Zum Ändern der Methode so dass jedes Gray-Code-Bild einzeln invertiert wird, muss mit set_structured_light_model_param *'normalization'* auf *'inverted_pattern'* gesetzt werden. Der Hauptvorteil der globalen Normalisierung ist die signifikant kleinere Bildanzahl während noch zufriedenstellende Ergebnisse erreicht werden. Nur für teilspiegelnde Oberflächen könnte die andere Methode robuster sein und sollte gewählt werden, falls die globale Normalisierung zu falschen Dekodier-Ergebnissen führt.

Erzeugung der Gray-Code-Bilder:

Größe, Anzahl und Aussehen der Gray-Code-Bilder hängt von folgenden Parametern ab (siehe set_structured_light_model_param): *'pattern_width'*, *'pattern_height'*, *'min_stripe_width'* and *'pattern_orientation'*.

Die ersten beiden Parameter legen die Größe der erzeugten Bilder fest und sollten normalerweise auf die Größe des Monitors gesetzt werden, der zum Anzeigen der Muster-Bilder verwendet wird. Der Parameter *'min_stripe_width'* legt die feinste Streifenbreite (in Pixel) der erzeugten Muster fest. Außerdem haben alle drei Parameter großen Einfluss auf die Anzahl der erzeugten Bilder.

Sei $max_stripe_width = 2^{\lceil \log_2 (pattern_width) \rceil}$

die maximale vertikale Streifenbreite und

$max_stripe_width = 2^{\lceil \log_2 (pattern_height) \rceil}$ die maximale horizontale Streifenbreite. Dann ist die Anzahl der Gray-Code-Bilder num_images in einer Orientierung (horizontal oder vertikal) gegeben durch

$$num_images = \lceil \log_2 (2 \cdot max_stripe_width / min_stripe_width) \rceil$$

Der Parameter *'pattern_orientation'* setzt die Orientierung der erzeugten Gray-Code-Bilder (und Phasenbilder). Standardmäßig werden Bilder mit vertikalen und Bilder mit horizontalen Streifen erzeugt. Es ist auch möglich, nur Bilder einer Richtung zu erzeugen. Dies wird jedoch nicht empfohlen, da Änderungen der Oberfläche dann nur in einer Richtung detektiert werden können. Nur falls Änderungen der Oberfläche bewusst ignoriert werden sollten oder Defekte nur in einer Richtung zu erwarten sind, sollte dieser Parameter mit set_structured_light_model_param verändert werden.

Erzeugung der Phasenbilder:

Da Phasenbilder geeignet sind, die praktischen Beschränkungen der Gray-Code-Bilder zu überwinden, ist ihre Erzeugung empfohlen und geschieht standardmäßig.

Falls die erreichte Genauigkeit der Gray-Code-Bilder ausreicht, kann die Erzeugung der Phasenbilder durch set_structured_light_model_param ausgeschaltet werden, wobei *'pattern_type'* auf *'gray_code'* gesetzt wird.

Erzeugung der Einzelstreifen-Bilder:

Wenn die alleinige Verwendung von Gray-Code-Bildern und Phasenbildern nicht robust genug ist (dies kann zum Beispiel auf halbspiegelnden Oberflächen der Fall sein), kann die Erstellung von Einzelstreifen-Bildern mit set_structured_light_model_param aktiviert werden, indem *'pattern_type'* auf *'single_stripe'* gesetzt wird.

Anzahl sowie Aussehen der Einzelstreifen-Bilder hängen von dem Parameter *'single_stripe_width'* ab, welcher die Streifenbreite in Pixeln festlegt (siehe set_structured_light_model_param).

Während zusätzliche Einzelstreifen-Bilder erzeugt werden, nimmt die Anzahl der erzeugten Gray-Code-Bilder typischerweise etwas ab. Die Anzahl der Gray-Code-Bilder kann mit der obigen Formel berechnet werden, indem *'max_stripe_width'* durch *'single_stripe_width'* ersetzt wird.

Die Verwendung dieser Bilder wird in der Einleitung des Kapitels Inspektion / Strukturiertes Licht beschrieben.

Parameter

▷ **PatternImages** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
Generierte Muster-Bilder.

▷ **StructuredLightModel** (input_control) ..... structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

Beispiel

```
* Create the model
create_structured_light_model ('deflectometry', StructuredLightModel)
* Set the size of the monitor
set_structured_light_model_param (StructuredLightModel, \
                                  'pattern_width', 1600)
set_structured_light_model_param (StructuredLightModel, \
                                  'pattern_height', 1200)
* Set the smallest width of the stripes in the pattern
set_structured_light_model_param (StructuredLightModel, \
                                  'min_stripe_width', 8)
* Generate the patterns to project
gen_structured_light_pattern (PatternImages, StructuredLightModel)
* Decode the camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
```

Ergebnis

Der Operator `gen_structured_light_pattern` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- StructuredLightModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`set_structured_light_model_param`

Nachfolger

`decode_structured_light_pattern`

Siehe auch

`create_structured_light_model`, `get_structured_light_model_param`, `get_structured_light_object`

Modul

3D Metrology

get_structured_light_model_param (: : StructuredLightModel, GenParamName : GenParamValue)

Abfragen von Parametern eines Modells für strukturiertes Licht.

Der Operator `get_structured_light_model_param` wird verwendet, um verschiedene Parameter eines Modells für strukturiertes Licht (`StructuredLightModel`) abzufragen. Die Namen der gewünschten Parameter werden in `GenParamName` übergeben und die entsprechenden Werte werden in

GenParamValue zurückgegeben. Es ist möglich, mehrere Parameter mit einem einzigen Aufruf des Operators get_structured_light_model_param abzufragen. Dabei können alle Parameter, die sich mit set_structured_light_model_param manipulieren lassen, abgefragt werden. Erläuterungen zu den individuellen Parametern finden sich in der Dokumentation zu set_structured_light_model_param.

Parameter

- ▷ **StructuredLightModel** (input_control) ..... structured_light_model ⇝ *handle*
  Handle des Modells für strukturiertes Licht.
- ▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
  Name des abgefragten Modellparameters.
  Defaultwert: 'min_stripe_width'
  Werteliste: GenParamName ∈ {'derivative_sigma', 'min_gray_difference', 'min_stripe_width', 'normalization', 'pattern_height', 'pattern_orientation', 'pattern_type', 'pattern_width', 'single_stripe_width', 'persistence'}
- ▷ **GenParamValue** (output_control) ..... attribute.value(-array) ⇝ *integer* / real / string
  Wert des abgefragten Modellparameters.

Beispiel

```
* Create the model
create_structured_light_model ('deflectometry', StructuredLightModel)
* Get the default value
get_structured_light_model_param (StructuredLightModel, \
                                  'min_stripe_width', Default)
* Set the value
set_structured_light_model_param (StructuredLightModel, \
                                  'min_stripe_width', 64)
* Get the value
get_structured_light_model_param (StructuredLightModel, 'min_stripe_width', \
                                  MinStripeWidth)
* Generate the patterns to project
gen_structured_light_pattern (PatternImages, StructuredLightModel)
* Decode the camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
```

Ergebnis

Der Operator get_structured_light_model_param gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_structured_light_model

Siehe auch

set_structured_light_model_param

Modul

3D Metrology

get_structured_light_object (: Object : StructuredLightModel, ObjectName :)

Ikonische (Zwischen-)Ergebnisse vom Modell für strukturiertes Licht abfragen.

Der Operator get_structured_light_object greift auf ikonische (Zwischen-)Ergebnisse eines Modells für strukturiertes Licht zu, das bereits mit decode_structured_light_pattern dekodiert wurde.

Einerseits kann man so die Ergebnisse des Dekodiervorgangs (*'correspondence_image'*) abfragen, sowie das weiterverarbeitete Defektbild (*'defect_image'*). Andererseits ist get_structured_light_object auch hilfreich bei einer Problembehandlung des Dekodiervorgangs. Zu beachten ist, dass Zwischenergebnisse nur aus dem StructuredLightModel abgefragt werden können, wenn der *'persistence'*-Modus des Modells *vor* dem Dekodieren mit set_structured_light_model_param eingeschaltet wurde.

Verfügbare Objekte ohne persistence-Modus:

'correspondence_image': Diese Bilder beschreiben die Pixelkorrespondenzen zwischen Monitor und Kamera. Bei Dekodieren in vertikaler und horizontaler Richtung werden beide Korrespondenzbilder konkateniert zurückgegeben. Ansonsten wird nur das Bild der jeweiligen Richtung zurückgegeben.

'pattern_region': In dieser Region wurden die Kamerabilder dekodiert. Diese Region ist identisch zum Definitionsbereich der Korrespondenzbilder.

'defect_image': Dieses Bild beschreibt mögliche Defekte der Oberfläche, die die projizierten Muster zur Kamera reflektiert. Prinzipiell weisen hohe Grauwerte im Defektbild darauf hin, dass die Gradienten der Korrespondenzbilder signifikant von den Gradienten in ihrer Nachbarschaft abweichen.

Nur im persistence-Modus verfügbare Objekte:

Bei Dekodieren in vertikaler und horizontaler Richtung werden Bilder beider Richtungen konkateniert zurückgegeben. Ansonsten werden nur die Bilder der jeweiligen Richtung zurückgegeben.

'binarized_image': Diese Bilder beschreiben die Gray-Code-Folge, wobei weiße und schwarze Grauwerte anzeigen, ob das Kamerapixel Reflexionen weißer oder schwarzer Regionen des projizierten Musters beobachtete. Wurde *'normalization'* mit set_structured_light_model_param auf *'inverted_pattern'* gesetzt, bezieht sich der Grauwert auf das erste Bild des invertierten Bildpaars.

'decoded_phase_shift_image': Diese Bilder enthalten die zu den projizierten Phasenbildern gehörigen Winkel. Eine Periode deckt in den projizierten Mustern Streifen der Ausdehnung *'min_stripe_width'* ab.

'decoded_single_stripe_image': Dekodiertes Einzelstreifen-Bild, das ausschließlich aus Einzelstreifen-Bildern berechnet wird. Wenn *'pattern_type'* auf *'single_stripe'* gesetzt wird, ergibt sich das Korrespondenzbild aus der Kombination des dekodierten Einzelstreifen-Bildes mit den dekodierten Gray-Code-Bildern und den dekodierten Phasenbildern.

'gray_code_correspondence_image': Aus Gray-Code-Bildern berechnetes Korrespondenzbild. Wenn *'pattern_type'* auf *'gray_code_and_phase_shift'* gesetzt wird, ergibt sich das aktuelle Korrespondenzbild aus dem *'gray_code_correspondence_image'* und den dekodierten Phasenbildern. Wenn *'pattern_type'* auf *'single_stripe'* gesetzt wird, ergibt sich das aktuelle Korrespondenzbild aus dem *'gray_code_correspondence_image'*, kombiniert mit den Gray-Code-Bildern und den Phasenbildern.

Parameter

▷ **Object** (output_object) ..... (multichannel-)object(-array) ⇝ *object*
Ikonisches Ergebnis.

▷ **StructuredLightModel** (input_control) ..... structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

▷ **ObjectName** (input_control) ..... string(-array) ⇝ *string*
Name des ikonischen Ergebnisses.
Defaultwert: 'correspondence_image'
Wertevorschläge: ObjectName ∈ {'correspondence_image', 'pattern_region', 'decoded_single_stripe_image', 'binarized_image', 'decoded_phase_shift_image', 'gray_code_correspondence_image', 'defect_image'}

Beispiel

```
* Create the model
create_structured_light_model ('deflectometry', StructuredLightModel)
* Generate the patterns to project
gen_structured_light_pattern (PatternImages, StructuredLightModel)
* Decode the camera images
```

```
decode_structured_light_pattern (CameraImages, StructuredLightModel)
* Get the computed correspondences and defects
get_structured_light_object (CorrespondenceImages, StructuredLightModel, \
                             'correspondence_image')
set_structured_light_model_param (StructuredLightModel, \
                                  'derivative_sigma', Sigma)
get_structured_light_object (DefectImage, StructuredLightModel, \
                             'defect_image')
```

Ergebnis

Der Operator `get_structured_light_object` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`decode_structured_light_pattern`

Nachfolger

`clear_structured_light_model`

Siehe auch

`gen_structured_light_pattern`

Modul

3D Metrology

read_structured_light_model (: : FileName : StructuredLightModel)

Lesen eines Modells für strukturiertes Licht aus einer Datei.

`read_structured_light_model` liest ein Modell für strukturiertes Licht, welches mit `write_structured_light_model` gespeichert wurde. Die Modelle für strukturiertes Licht haben in HALCON die Standard-Dateiendung 'hslm'.

Es kann hilfreich sein ein Modell zu speichern, nachdem alle optimalen Parameter für eine Messanordnung gesetzt wurden. Das Modell kann dann zu einem späteren Zeitpunkt wieder eingelesen werden, um ohne Umstände Muster-Bilder neu zu generieren. Objekte, die bereits vor dem Speichern des Modells generiert wurden, können auch nach dem Einlesen eines Modells wieder aufgegriffen werden.

Das Konzept des strukturierten Lichts sowie die unterstützten Anwendungsbereiche werden in der Einleitung des Kapitels Inspektion / Strukturiertes Licht beschrieben.

Parameter

▷ **FileName** (input_control) ..... filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.hslm`

▷ **StructuredLightModel** (output_control) ..... structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

Beispiel

```
* Read the model
read_structured_light_model ('ExampleModel.hslm', StructuredLightModel)
* Get a previously decoded result
get_structured_light_object (CorrespondenceImages, StructuredLightModel, \
                             'correspondence_image')
```

```
* Decode new camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
* Get the decoded result
get_structured_light_object (CorrespondenceImagesNew, StructuredLightModel, \
                             'correspondence_image')
```

Ergebnis

Der Operator `read_structured_light_model` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`write_structured_light_model`

Siehe auch

`create_structured_light_model`, `clear_structured_light_model`

Modul

3D Metrology

serialize_structured_light_model (: : StructuredLightModel : SerializedItemHandle)

Serialisieren eines Modells für strukturiertes Licht.

`serialize_structured_light_model` serialisiert ein Modell für strukturiertes Licht (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das Modell wird in dem Handle `StructuredLightModel` übergeben. Das serialisierte Modell wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_structured_light_model` wieder deserialisiert werden.

Parameter

▷ **StructuredLightModel** (input_control) ..... structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Beispiel

```
* Create the model
create_structured_light_model ('deflectometry', StructuredLightModel)
* Generate the patterns to project
gen_structured_light_pattern (PatternImages, StructuredLightModel)
* Decode the camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
* Serialize the model
serialize_structured_light_model (StructuredLightModel, \
                                  SerializedItemHandle)
```

Ergebnis

Der Operator `serialize_structured_light_model` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`clear_structured_light_model`, `fwrite_serialized_item`, `send_serialized_item`, `deserialize_structured_light_model`

Siehe auch

`create_structured_light_model`, `read_structured_light_model`, `deserialize_structured_light_model`

Modul

3D Metrology

set_structured_light_model_param (: : StructuredLightModel, GenParamName, GenParamValue :)

Setzen von Parametern eines Modells für strukturiertes Licht.

Der Operator `set_structured_light_model_param` wird verwendet, um die Parameter eines Modells für strukturiertes Licht `StructuredLightModel` zu manipulieren. Die aktuellen Parametereinstellungen können mit dem Operator `get_structured_light_model_param` abgefragt werden.

Das Konzept des strukturierten Lichts sowie die unterstützen Anwendungsbereiche werden in der Einleitung des Kapitels Inspektion / Strukturiertes Licht beschrieben.

Allgemeine Parameter:

'persistence': Aktiviert (`GenParamValue`=*'true'*) oder deaktiviert (`GenParamValue`=*'false'*) den Modus *'persistence'* des Modells für strukturiertes Licht. Im Persistenz-Modus speichert das Modell Zwischenergebnisse der Dekodierung, welche später mit `get_structured_light_object` inspiziert werden können.

Es ist zu beachten, dass das Modell in diesem Modus signifikanten Speicherplatz benötigen könnte. Deswegen empfehlen wir, den Persistenz-Modus nur während der Einrichtung oder zur Fehlerbehebung zu aktivieren.

Wird der Wert von *'persistence'* verändert, werden alle Ergebnisse des Modells gelöscht.

Werte: 'true', *'false'*, 1, 0

Defaultwerte: 'false'

Parameter, die das Erscheinungsbild der Muster-Bilder beeinflussen:

'pattern_width': Setzt die Breite der Muster-Bilder, welche durch den Aufruf von `gen_structured_light_pattern` erzeugt werden. Normalerweise sollte *'pattern_width'* der Breite des Monitors entsprechen, der zur Anzeige der Muster-Bilder verwendet wird.

Der Wert von *'pattern_width'* muss größer als *'min_stripe_width'*/2 sein. Für den Fall, dass *'pattern_type'* auf *'single_stripe'* gesetzt ist, muss *'pattern_width'* zusätzlich größer als *'single_stripe_width/2'* sein.

Wird der Wert von *'pattern_width'* verändert, ist das Modell nicht mehr dekodierbar und alle möglicherweise verfügbaren Ergebnisse werden gelöscht. Deshalb muss `gen_structured_light_pattern` aufgerufen werden, bevor das Modell mit `decode_structured_light_pattern` dekodiert wird.

Werte: Ganze Zahlen größer oder gleich 1

Defaultwert: 1024

'pattern_height': Setzt die Höhe der Muster-Bilder, welche durch den Aufruf von `gen_structured_light_pattern` erzeugt werden. Normalerweise sollte *'pattern_height'* der Höhe des Monitors entsprechen, der zur Anzeige der Muster-Bilder verwendet wird.

Der Wert von *'pattern_height'* muss größer als *'min_stripe_width'*/2 sein. Für den Fall, dass *'pattern_type'* auf *'single_stripe'* gesetzt ist, muss *'pattern_height'* zusätzlich größer als *'single_stripe_width/2'* sein.

Wird der Wert von *'pattern_height'* verändert, ist das Modell nicht mehr dekodierbar und alle möglicherweise verfügbaren Ergebnisse werden gelöscht. Deshalb muss gen_structured_light_pattern aufgerufen werden, bevor das Modell mit decode_structured_light_pattern dekodiert wird.

Werte: Ganze Zahlen größer oder gleich 1

Defaultwert: 1024

'min_stripe_width': Setzt die Breite (in Pixeln) des schmalsten Streifens der Muster Bilder, welche durch den Aufruf von gen_structured_light_pattern erzeugt werden.

Der Wert von *'min_stripe_width'* muss eine Zweierpotenz sein. Außerdem muss *'min_stripe_width'* kleiner oder gleich dem Minimum der Parameter *'pattern_width'* und *'pattern_height'* sein, nachdem beide bis zur nächsten Zweierpotenz aufgerundet wurden. Für den Fall, dass *'pattern_type'* auf *'single_stripe'* gesetzt ist, muss *'min_stripe_width'* zusätzlich kleiner oder gleich *'single_stripe_width'* sein.

Wird der Wert von *'min_stripe_width'* verändert, ist das Modell nicht mehr dekodierbar und alle möglicherweise verfügbaren Ergebnisse werden gelöscht. Deshalb muss gen_structured_light_pattern aufgerufen werden, bevor das Modell mit decode_structured_light_pattern dekodiert wird.

Werte: Ganzzahlige Zweierpotenzen größer oder gleich 4

Defaultwert: 32

'normalization': Setzt den Modus der Normalisierung des Modells für strukturiertes Licht. Wenn der Parameter *'normalisation'* auf *'global'* gesetzt ist, generiert gen_structured_light_pattern zwei zusätzliche Muster-Bilder bestehend aus einem weißen und einem schwarzen Bild. Diese Bilder werden dann während der Dekodierung verwendet, um die eigentliche Muster Region zu ermitteln. Wird stattdessen der Wert *'inverted_pattern'* gewählt, wird jedes Gray-Code-Bild zusätzlich invertiert und die Muster Region wird, unter Verwendung der Gray-Code-Bilder und ihrer invertierten Version, iterativ segmentiert.

Wird der Wert von *'normalization'* verändert, ist das Modell nicht mehr dekodierbar und alle möglicherweise verfügbaren Ergebnisse werden gelöscht. Deshalb muss gen_structured_light_pattern aufgerufen werden, bevor das Modell mit decode_structured_light_pattern dekodiert wird.

Werte: 'global', *'inverted_pattern'*

Defaultwert: 'global'

'pattern_orientation': Setzt die Orientierung der Muster-Bilder, welche durch den Aufruf von gen_structured_light_pattern erzeugt werden. Wenn der Parameter *'pattern_orientation'* auf *'both'* gesetzt ist, werden Muster Bilder mit vertikalen Streifen sowie Muster-Bilder mit horizontalen Streifen erzeugt. Bei *'vertical'* bestehen die Muster-Bilder nur aus vertikalen Streifen, wohingegen bei *'horizontal'* nur horizontale Streifen generiert werden.

Wird der Wert von *'pattern_orientation'* verändert, ist das Modell nicht mehr dekodierbar und alle möglicherweise verfügbaren Ergebnisse werden gelöscht. Deshalb muss gen_structured_light_pattern aufgerufen werden, bevor das Modell mit decode_structured_light_pattern dekodiert wird.

Werte: 'both', *'vertical'*, *'horizontal'*

Defaultwerte: 'both'

'pattern_type': Setzt den Typ der Muster-Bilder, welche durch den Aufruf von gen_structured_light_pattern erzeugt werden.

'pattern_type'	**Erzeugte Bilder**
'gray_code'	Gray-Code-Bilder
'gray_code_and_phase_shift'	Gray-Code-Bilder und Phasenbilder
'single_stripe'	Einzelstreifen-Bilder, Gray-Code-Bilder und Phasenbilder

Normalerweise liefert der Typ *'gray_code_and_phase_shift'* präzisere Ergebnisse, da die Kombination aus Gray-Code-Bildern und Phasenbildern sub-pixel genaue Koordinaten liefert. Der Typ *'single_stripe'* erhöht die Robustheit im Fall von partiell spiegelnden Oberflächen.

Wird *'pattern_type'* verändert, ist das Modell nicht mehr dekodierbar und alle möglicherweise verfügbaren Ergebnisse werden gelöscht. Deshalb muss gen_structured_light_pattern aufgerufen werden, bevor das Modell mit decode_structured_light_pattern dekodiert wird.

Werte: 'gray_code_and_phase_shift', *'gray_code'*, *'single_stripe'*

Defaultwerte: 'gray_code_and_phase_shift'

'single_stripe_width': Legt die Breite (in Pixeln) des Einzelstreifens fest, welcher durch den Aufruf von `gen_structured_light_pattern` mit dem *'pattern_type'* *'single_stripe'* generiert wird.

Der Wert muss von *'single_stripe_width'* muss eine Zweierpotenz sein und muss größer oder gleich *'min_stripe_width'* sein. Außerdem muss *'single_stripe_width'* kleiner oder gleich dem Minimum der Parameter *'pattern_width'* und *'pattern_height'* sein, nachdem beide bis zur nächsten Zweierpotenz aufgerundet wurden.

Beachten Sie, dass der Parameter *'single_stripe_width'* nur gesetzt werden kann, wenn *'pattern_type'* der Wert *'single_stripe'* zugewiesen wurde. Wird *'pattern_type'* in *'single_stripe'* geändert, wird *'single_stripe_width'* auf eine Zweierpotenz ungefähr in der Mitte zwischen *'min_stripe_width'* und dem Minimum von *'pattern_width'* und *'pattern_height'* gesetzt.

Wird *'single_stripe_width'* verändert, ist das Modell nicht mehr dekodierbar und alle möglicherweise verfügbaren Ergebnisse werden gelöscht. Daher muss `gen_structured_light_pattern` aufgerufen werden, bevor das Modell mit `decode_structured_light_pattern` dekodiert wird.

Values: Ganzzahlige Zweierpotenzen größer oder gleich 4

Default: Zweierpotenz ungefähr in der Mitte zwischen dem aktuell gesetzten Wert *'min_stripe_width'* und *'pattern_width'*, *'pattern_height'*

Parameter, welche die Segmentierung der Muster Region beeinflussen:

'min_gray_difference': Setzt die kleinste Grauwertdifferenz für die Gray-Code-Bilder. Dieser Wert wird innerhalb des Operators `decode_structured_light_pattern` verwendet, um binarisierte Bilder zu ermitteln. Dadurch wird für jeden Pixel des Kamerabildes die Grauwertdifferenz zwischen belichtetem und unbelichtetem Fall berechnet. Wenn die Grauwertdifferenz eines Pixels kleiner als *'min_gray_difference'* ist, wird der Pixel von der Domäne der binarisierten Bilder (siehe `get_structured_light_object`) ausgeschlossen und somit auch von weiteren Berechnungen.

Wenn *'min_gray_difference'* auf 0 gesetzt wird, ist die segmentierte Region identisch zur Eingaberegion der Kamerabilder, mit dem einzigen Unterschied, dass Pixel mit einem Dekodierergebnis außerhalb der Muster-Bilder ausgeschlossen werden.

Werte: Ganze Zahlen großer oder gleich 0

Defaultwerte: 30

Parameter, die das Defektbild beeinflussen:

'derivative_sigma': Setzt das Sigma der Gaußfunktion (d.h. das Ausmaß der Glättung) zur Faltung der Korrespondenzbilder, um das Defektbild zu berechnen.

Werte: Gleitkommazahlen oder ganze Zahlen größer oder gleich 0.01

Defaultwert: 2

Parameter

▷ **`StructuredLightModel`** (input_control) ..... structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
Name des zu ändernden Parameters.
Defaultwert: 'min_stripe_width'
Werteliste: `GenParamName` ∈ {'derivative_sigma', 'min_gray_difference', 'min_stripe_width', 'normalization', 'pattern_height', 'pattern_orientation', 'pattern_type', 'pattern_width', 'single_stripe_width', 'persistence'}

▷ **`GenParamValue`** (input_control) ..... attribute.value(-array) ⇝ *integer* / real / string
Neuer Wert des Parameters.
Defaultwert: 32
Wertevorschläge: `GenParamValue` ∈ {0, 0.01, 0.5, 0.7, 1, 1.4, 5, 50.0, 64, 128, 256, 1024, 'true', 'false', 'both', 'vertical', 'horizontal', 'global', 'inverted_pattern', 'gray_code_and_phase_shift', 'gray_code', 'single_stripe'}

Beispiel

```
* Create the model
create_structured_light_model ('deflectometry', StructuredLightModel)
```

```
* Set the size of the monitor
set_structured_light_model_param (StructuredLightModel, \
                                  'pattern_width', 1600)
set_structured_light_model_param (StructuredLightModel, \
                                  'pattern_height', 1200)
* Set the smallest width of the stripes in the pattern
set_structured_light_model_param (StructuredLightModel, \
                                  'min_stripe_width', 8)
* Generate the patterns to project
gen_structured_light_pattern (PatternImages, StructuredLightModel)
* Set the expected black/white contrast in the region of interest
set_structured_light_model_param (StructuredLightModel, \
                                  'min_gray_difference', 70)
* Decode the camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
* Get the computed correspondences and defects
get_structured_light_object (CorrespondenceImages, StructuredLightModel, \
                             'correspondence_image')
set_structured_light_model_param (StructuredLightModel, 'derivative_sigma', \
                                  Sigma)
get_structured_light_object (DefectImage, StructuredLightModel, \
                             'defect_image')
```

Ergebnis

Der Operator `set_structured_light_model_param` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- StructuredLightModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_structured_light_model`

Siehe auch

`get_structured_light_model_param`

Modul

3D Metrology

write_structured_light_model (: : StructuredLightModel, FileName :)

Abspeichern eines Modells für strukturiertes Licht in einer Datei.

`write_structured_light_model` schreibt ein Modell für strukturiertes Licht in die Datei `FileName`. Die Modelle für strukturiertes Licht haben in HALCON die Standard-Dateiendung 'hslm'. Das Modell kann dann mit `read_structured_light_model` gelesen werden. Dabei ist zu berücksichtigen, dass `write_structured_light_model` keine zuvor generierten Muster-Bilder im Modell abspeichert. Mit anderen Worten, nur die Parameter und Objekte des Modells die bereits generiert wurden, werden in der Datei gespeichert. Die Liste mit Objekten sowie eine Beschreibung, wie diese zu generieren sind, kann in der Referenz zu `get_structured_light_object` gefunden werden.

Das Konzept des strukturierten Lichts sowie die unterstützten Anwendungsbereiche werden in der Einleitung des Kapitels Inspektion / Strukturiertes Licht beschrieben.

Parameter

▷ **StructuredLightModel** (input_control) ..... structured_light_model ⇝ *handle*
Handle des Modells für strukturiertes Licht.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.hslm`

Beispiel

```
* Create the model
create_structured_light_model ('deflectometry', StructuredLightModel)
* Generate the patterns to project
gen_structured_light_pattern (PatternImages, StructuredLightModel)
* Decode the camera images
decode_structured_light_pattern (CameraImages, StructuredLightModel)
* Write the model
write_structured_light_model (StructuredLightModel, 'ExampleModel.hslm')
```

Ergebnis

Der Operator `write_structured_light_model` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`clear_structured_light_model`

Siehe auch

`create_structured_light_model`, `clear_structured_light_model`

Modul

3D Metrology

13.4 Texturinspektion

Dieses Kapitel beschreibt Operatoren für die Texturinspektion.

Konzept der Texturinspektion

Mit der Texturinspektion können mit wenigen Parametereinstellungen texturierte Oberflächen auf Fehler überprüft werden. Der Algorithmus benötigt ein Training mit Bildern von fehlerfreier Textur. Bei dem Training werden Texturmerkmale aus den Trainingsbildern extrahiert und dazu genutzt, auf der Basis eines Gaussian Mixture Model (GMM) Klassifikators ein Texturinspektionsmodell zu trainieren. Nach dem erfolgreichen Training können Bilder der gleichen Textur mit dem Texturinspektionsmodell verglichen werden und somit potentielle Fehler identifiziert werden. Texturinspektion arbeitet auf einer Bildpyramide, so dass unterschiedliche Frequenzbereiche der Textur untersucht werden.

Es ist möglich Mehrkanalbilder in der Texturinspektion-Pipeline zu verwenden.

Im Folgenden werden die einzelnen Schritte, die zur Texturinspektion benötigt werden, kurz beschrieben.

Anlegen eines Texturinspektionsmodells: Zuerst wird ein Texturinspektionsmodell mit

- `create_texture_inspection_model`

angelegt oder mit

- read_texture_inspection_model

eingelesen.

Hinzufügen von Trainingsdaten: Der Operator

- add_texture_inspection_model_image

fügt dem Texturinspektionsmodell neue Trainingsbilder hinzu.

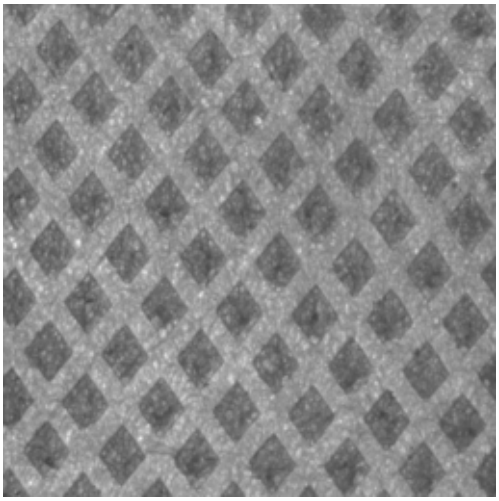


Bild einer texturierten Oberfläche, dass zum Trainieren eines Texturinspektionsmodells genutzt wird.

Die mit add_texture_inspection_model_image zugefügten Bilder können mit einem Aufruf von get_texture_inspection_model_image wieder abgefragt werden. Falls alle oder ein Teil der Bilder nicht mehr zur Texturinspektion verwendet werden sollen, so können diese Bilder mit Hilfe des Operators remove_texture_inspection_model_image aus dem Modell entfernt werden.

Trainieren des Texturinspektionsmodells: Das Texturinspektionsmodell wird mit

- train_texture_inspection_model

trainiert.

Während des Trainings wird eine Bildpyramide erzeugt. Für jede Pyramidenstufe wird ein Gaussian Mixture Model (GMM) trainiert und eine Fehlerschwelle (*'novelty_threshold'*) bestimmt. Diese Schwelle ermöglicht beim Anwenden der Texturinspektion die Klassifikation von jedem Pixel in fehlerhafte und fehlerfreie Textur. Nach dem Training können die Fehlerschwellen mit get_texture_inspection_model_param abgefragt werden.

Verschiedene Parameter beeinflussen das Training. Sie können mit

- set_texture_inspection_model_param

gesetzt werden.

Das folgende Vorgehen für das Training wird empfohlen:

- Setzen von initialen Parametern, wenn nötig:
  - *'patch_normalization'*: Wird dieser Parameter auf *'weber'* gesetzt, werden die Texturmerkmale normalisiert und das Texturinspektionsmodell wird robust gegen Beleuchtungsänderungen. Hier ist zu beachten, dass eine Normalisierung nicht immer gewünscht ist, z.B. wenn die Helligkeit ein wichtiges Merkmal bei der Unterscheidung zwischen fehlerhafter und fehlerfreier Textur darstellt. In diesem Fall ist eine kontrollierte Beleuchtung für eine erfolgreiche Klassifizierung zwingend notwendig.

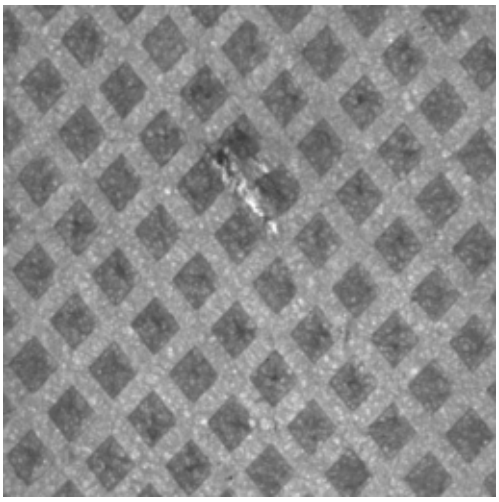
  - *'patch_rotational_robustness'*: Wird dieser Parameter auf *'true'* gesetzt, werden die Texturmerkmale so sortiert, dass das Texturinspektionsmodell robust gegen Rotationsänderungen wird. Dies kann hilfreich sein, wenn es schwierig ist, alle möglichen Orientierungen der Textur durch Trainingsbilder abzudecken.

- Trainieren des Texturinspektionsmodells (mit den Default-Parametern): Das initiale Training des Texturinspektionsmodells kann viel Zeit in Anspruch nehmen. Die benötigte Zeit ist abhängig von der Anzahl der Trainingsbilder und deren Auflösung. Mögliche Strategien zum Verringern der Trainingszeit:
  - Zoomen der Bilder: Häufig funktioniert die Texturinspektion sogar besser mit niedrig aufgelösten Bildern. Ist bekannt, dass nicht die volle Auflösung der Bilder benötigt wird, kann das Zoomen der Bilder einen signifikanten Zeitvorteil bringen. Zoomt man ein Bild z.B. mit Faktor 2, kann dies die Trainingszeit um Faktor 4 beschleunigen.
  - Beschleunigen des Trainings: Ein weiterer, aber auch riskanterer Ansatz ist es, den Schwellenwert, der bestimmt, wann die Optimierung des Klassifikators beendet wird, mit dem Parameter *'gmm_em_threshold'* z.B. von *0.001* auf *0.1* zu erhöhen. Dies führt zu kürzeren Trainingszeiten aber auch zu einem möglicherweise weniger genauen Texturinspektionsmodell. Wird dieses Vorgehen gewählt, ist es strengstens empfohlen, den Schwellenwert nach dem Anpassen der meisten Parameter wieder zu reduzieren.
- Prüfen des Trainingsergebnisses: Mit einem Satz von Bildern mit fehlerhafter Textur kann das Ergebnis einer Klassifikation mit dem trainierten Texturinspektionsmodell geprüft werden. Dies wird näher im Abschnitt 'Anwenden des Texturinspektionsmodells' beschrieben. Ist das Ergebnis nicht zufriedenstellend, können weitere Parameter angepasst werden. Die wichtigsten Parameter sind:
  - *'num_levels'* oder *'levels'*: Die Anzahl der benutzten Pyramidenstufen bzw. die expliziten genutzten Pyramidenstufen. Höhere Pyramidenstufen funktionieren gut für gröbere Texturmerkmale. Ist die Textur in den Bildern sehr grob, werden die niedrigeren Pyramidenstufen eventuell gar nicht für eine erfolgreiche Inspektion benötigt. Um deutlich an Laufzeit zu sparen, können die entsprechenden Skalen durch das explizite Setzen der Skalen mittels *'levels'* ausgeschlossen werden, Wird zum Beispiel *'levels'* auf [1,3] gesetzt, werden im Training und zum Klassifizieren nur die erste und die dritte Level verwendet.
  - *'sensitivity'*: Dieser Parameter kontrolliert, wie streng die Fehlerschwellen (*'novelty_threshold'*) gesetzt werden. Somit kann die Empfindlichkeit gegenüber Texturfehlern eingestellt werden. Generell führen positive Werte zu einer höheren Empfindlichkeit des Verfahrens und dadurch zu mehr Texturfehlern. Umgekehrt führen negative Werte zu einer niedrigeren Empfindlichkeit des Verfahrens und dadurch zu weniger Texturfehlern.
  - *'novelty_threshold'*: Dieser Parameter setzt die Fehlerschwellen manuell. Dies kann zur Feinabstimmung genutzt werden, sollten die Ergebnisse der automatischen Schätzung nicht optimal sein.
- Wiederholen des Trainings, wenn nötig: Wurden Parameter angepasst, muss das Texturinspektionsmodell für gewöhnlich neu trainiert werden.

Anwenden des Texturinspektionsmodells: Nach dem erfolgreichen Training kann das Texturinspektionsmodell dazu verwendet werden, texturierte Oberflächen zu klassifizieren. Jedes Testbild kann mit

- `apply_texture_inspection_model`

mit dem Texturinspektionsmodell verglichen werden.



Testbild mit Fehlern.

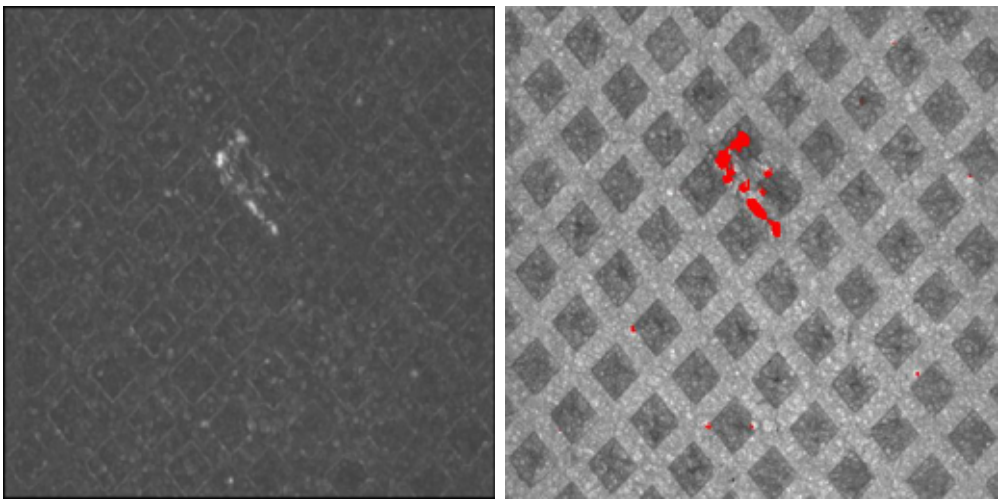
Für jedes Testbild gibt der Operator eine Fehlerregion (*'novelty_region'*) zurück, die anzeigt, wo das Bild größere Abweichungen von den trainierten Trainingsdaten aufweist. Die Fehlerregion wird durch die Kombination der Fehlerregionen der unterschiedlichen Bildpyramidenstufen erzeugt. Dabei werden die Fehlerregionen von benachbarten Skalen miteinander geschnitten. Dadurch wird die Robustheit gegenüber Rauschen erhöht. Die so entstandenen Schnittflächen werden in der *'novelty_region'* vereinigt. Falls es keine benachbarte Level gibt, wird die Fehlerregion der entsprechenden Level direkt zu der *'novelty_region'* hinzugefügt. Setzt man zum Beispiel *'num_levels'* auf 1, so gibt es keine benachbarten Skalen und die Fehlerregion vom ersten Level wird in *'novelty_region'* zurückgegeben.

Zum Debuggen kann der Parameter *'gen_result_handle'* mit `set_texture_inspection_model_param` auf *'true'* gesetzt werden. Auf diese Weise werden die Novelty-Score-Bilder und die Fehlerregionen der einzelnen Bildpyramidenstufen in einem Ergebnis-Handle gespeichert, aus welchem sie wieder mit

- `get_texture_inspection_result_object`

ausgelesen werden können. Generell beschreiben die Novelty-Score-Bilder wie gut jedes Pixel zu dem Texturinspektionsmodell passt, welches im Trainingsprozess entstanden ist. Die Fehlerregionen der einzelnen Pyramidenstufen werden dann durch das Anwenden der Fehlerschwelle (*'novelty_threshold'*) auf die jeweiligen Novelty-Score-Bilder berechnet.

Das Inspizieren der Novelty-Score-Bilder und der Fehlerregionen kann Aufschluss darüber geben, welche Art von Fehlern auf welchen Pyramidenstufen gefunden werden können. Darauf basierend kann entschieden werden, ob Parameter für das Training und die Klassifizierung angepasst werden sollten, z.B., ob es sinnvoll ist, explizit nur bestimmte Pyramidenstufen durch das Setzen des Parameters *'levels'* auszuwählen.



(1) (2)

Das resultierende Novelty-Score Bild (1) und die resultierende Fehlerregion (2) für das obige Testbild.

Weitere Operatoren

Die Funktionalität der Texturinspektion wird durch verschiedene Parameter beeinflusst. Die Parameterwerte können mit `get_texture_inspection_model_param` abgefragt und mit `set_texture_inspection_model_param` geändert werden.

Ein Texturinspektionsmodell kann mit dem Operator `write_texture_inspection_model` abgespeichert werden. Dabei wird nur das Texturinspektionsmodell und nicht dem Modell hinzugefügte Bilder gespeichert. Die hinzugefügten Bilder können mit `get_texture_inspection_model_image` abgefragt und mit einem Aufruf von `write_object` separat abgespeichert werden.

Um den Speicherbedarf des Texturinspektionsmodells zu verringern, können dem Model bereits hinzugefügte Bilder wieder herausgelöscht werden. Dies erfolgt durch den Operator `remove_texture_inspection_model_image`.

Außerdem kann das Modell mit `serialize_texture_inspection_model` und `deserialize_texture_inspection_model` serialisiert bzw. deserialisiert werden.

Glossar

Im Folgenden werden die wichtigsten Begriffe, die im Umfeld der Texturinspektion verwendet werden, beschrieben:

Trainingsbilder Bilder, die für das Training verwendet werden.

Testbilder Bilder, die mit dem trainierten Texturinspektionsmodell verglichen werden.

Patch Menge von benachbarten Pixeln.

Texturmerkmal Pixelwerte innerhalb eines Patches.

Trainingsdaten Texturmerkmale, die in einem Trainingsprozess genutzt werden.

Neuheitswert Im Testprozess werden die Texturmerkmale der Testbilder mit dem Texturinspektionsmodell verglichen und ihr Neuheitswert (*novelty score*) wird berechnet. Je höher dieser Wert ist, desto wahrscheinlicher ist es, dass das Texturmerkmal nicht zu dem Texturinspektionsmodell passt.

Fehlerschwelle Die Fehlerschwelle (Novelty Threshold) wird im Trainingsprozess geschätzt. Texturmerkmale mit einem Neuheitswert unter der Fehlerschwelle passen zum Texturinspektionsmodell, Texturmerkmale mit einem höheren Neuheitswert passen nicht.

```
add_texture_inspection_model_image ( 
    Image : : TextureInspectionModel : Indices )
```

Hinzufügen von Trainingsbildern zu einem Texturinspektionsmodell.

`add_texture_inspection_model_image` fügt Trainingsbilder zum Texturinspektionsmodell `TextureInspectionModel` hinzu. Für die weitere Verarbeitung wird für jedes Bild die Information innerhalb der Domäne des Bildes berücksichtigt. Da Merkmale, welche im Trainingsprozess für jedes Pixel extrahiert werden, in Relation zu ihrer Nachbarschaft berechnet werden, können für Pixel am Rand der Domäne keine Merkmale extrahiert werden. Die Größe des Randes hängt von der Größe der Merkmale (*'patch_size'*, siehe `set_texture_inspection_model_param`) ab und ergibt sich als:

$$Rand = patch_size/2 - 1.$$

Alle Bilder, die mit diesem Operator dem Modell hinzugefügt werden, werden zum Trainieren des Texturinspektionsmodells (siehe `train_texture_inspection_model`) verwendet. Des Weiteren ist es möglich, die hinzugefügten Bilder mit `get_texture_inspection_model_image` abzufragen.

Jedes hinzugefügte Bild erhält einen Index (`Indices`), der es dem Nutzer ermöglicht, eine einfache Zuordnung der hinzugefügten Bilddaten zu seiner Bilddatenbank herzustellen. Der Index kann außerdem dazu verwendet werden, um hinzugefügte Bilder die nicht mehr benötigt werden, mit `remove_texture_inspection_model_image` gezielt aus aus dem Modell zu entfernen.

Es ist möglich Grauwertbilder oder Mehrkanalbilder hinzuzufügen. Allerdings müssen alle übergebenen Bilder die gleiche Anzahl von Kanälen haben.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild der zu trainierenden Textur.
- ▷ **TextureInspectionModel** (input_control) . . . . . texture_inspection_model ⇝ *handle*
  Handle des Texturinspektionsmodells.
- ▷ **Indices** (output_control) . . . . . integer(-array) ⇝ *integer*
  Indices der Bilder die dem Texturinspektionsmodell hinzugefügt wurden.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TrainImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModel, \
                                TextureInspectionResultID)
```

Ergebnis

Der Operator `add_texture_inspection_model_image` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextureInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_texture_inspection_model`, `set_texture_inspection_model_param`, `get_texture_inspection_model_param`

Nachfolger

`train_texture_inspection_model`, `clear_texture_inspection_model`, `remove_texture_inspection_model_image`, `get_texture_inspection_model_image`

Modul

Matching

apply_texture_inspection_model (Image : NoveltyRegion : TextureInspectionModel : TextureInspectionResultID)

Inspizieren der Textur in einem Bild.

Der Operator apply_texture_inspection_model vergleicht die Textur in einem Bild Image mit dem trainierten Texturinspektionsmodell TextureInspectionModel. Image kann ein einzelnes oder ein Tupel von Bildern sein. Dabei ist es möglich Grauwertbilder oder Mehrkanalbilder zu übergeben. Der Operator erwartet, dass die übergebenen Bilder in Image die gleiche Anzahl von Kanälen haben, wie die Bilder mit denen das Modell trainiert wurde. Die Laufzeit von apply_texture_inspection_model skaliert etwa linear mit der Anzahl der Bildkanäle. Modelle, die farbige Bilder verwenden, sind jedoch im Allgemeinen besser in der Lage, farbige Texturdefekte zu erkennen.

Regionen im Bild die nicht zum Texturinspektionsmodell passen, werden in der Fehlerregion NoveltyRegion zurückgegeben. Außerdem gibt der Operator, falls *'gen_result_handle'* mit set_texture_inspection_model_param auf *'true'* gesetzt wurde, im Handle TextureInspectionResultID detailliertere Informationen zur Texturklassifikation zurück. Ist *'gen_result_handle'* auf *'false'* gesetzt, bleibt TextureInspectionResultID leer.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Für jede Pyramidenstufe werden Texturmerkmale extrahiert und mit dem entsprechenden GMM-Klassifikator klassifiziert. Der sich ergebende Neuheitswert (*'novelty_score'*) wird mit der Fehlerschwelle (*'novelty_threshold'*) dieser Pyramidenstufe verglichen und als fehlerhaft oder fehlerfrei klassifiziert. Die fehlerhaften Pixel werden zunächst in einer Fehlerregion für jede Pyramidenstufe gesammelt und anschließend zu der endgültigen Fehlerregion kombiniert, die in NoveltyRegion zurückgegeben wird. Dabei werden die Fehlerregionen von benachbarten Skalen miteinander geschnitten. Dadurch wird die Robustheit gegenüber Rauschen erhöht. Die so entstandenen Schnittflächen werden in der *'novelty_region'* vereinigt. Falls es keine benachbarte Level gibt, wird die Fehlerregion der entsprechenden Level direkt zu der *'novelty_region'* hinzugefügt. Setzt man zum Beispiel *'num_levels'* auf 1, so gibt es keine benachbarten Skalen und die Fehlerregion vom ersten Level wird in *'novelty_region'* zurückgegeben.

Ist *'gen_result_handle'* mit set_texture_inspection_model_param auf *'true'* gesetzt worden, enthält TextureInspectionResultID Novelty-Score-Bilder und die daraus resultierenden Fehlerregionen für jede Pyramidenstufe.

Diese Informationen können beim Debuggen und beim Anpassen der Modellparameter (z.B. der Fehlerschwellen) helfen und werden mit get_texture_inspection_result_object abgefragt.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image-array ⇝ *object :* byte / uint2 / real
  Bild(er) der zu inspizierenden Textur.
- ▷ **NoveltyRegion** (output_object) ..... region-array ⇝ *object*
  Detektierte Fehlerregion.
- ▷ **TextureInspectionModel** (input_control) ..... texture_inspection_model ⇝ *handle*
  Handle des Texturinspektionsmodells.
- ▷ **TextureInspectionResultID** (output_control) ..... texture_inspection_result ⇝ *handle*
  Handle des Inspektionsergebnisses.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Set parameters
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gen_result_handle', 'true')
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
```

```
                                    TextureInspectionModel, \
                                    TextureInspectionResultID)
```

Ergebnis

Der Operator `apply_texture_inspection_model` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextureInspectionResultID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`train_texture_inspection_model`

Nachfolger

`get_texture_inspection_result_object`, `get_texture_inspection_model_param`, `clear_texture_inspection_result`, `clear_texture_inspection_model`

Modul

Matching

`clear_texture_inspection_model` (: : TextureInspectionModel :)

Löschen eines Texturinspektionsmodells und Freigabe des verwendeten Speichers.

Der Operator `clear_texture_inspection_model` löscht ein Texturinspektionsmodell, das mit `create_texture_inspection_model` erzeugt wurde. Aller vom Modell verwendeter Speicher wird freigegeben. Das Handle des Modells wird in `TextureInspectionModel` übergeben. Nach dem Aufruf von `clear_texture_inspection_model` ist das Handle ungültig.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **TextureInspectionModel** (input_control) ..... texture_inspection_model(-array) ⇝ *handle*
Handle des Texturinspektionsmodells.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModel, \
                                TextureInspectionResultID)
```

Ergebnis

Der Operator `clear_texture_inspection_model` gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextureInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

`create_texture_inspection_model`

Modul

Matching

clear_texture_inspection_result (: : TextureInspectionResultID :)

Löschen eines Texturinspektionsergebnisses und Freigabe des verwendeten Speichers.

Der Operator `clear_texture_inspection_result` gibt das Handle `TextureInspectionResultID`für die Ergebnisse einer Texturinspektion frei, das durch den Aufruf von `apply_texture_inspection_model` erzeugt wurde. Aller vom Handle verwendeter Speicher wird freigegeben. Nach dem Aufruf von `clear_texture_inspection_result` ist das Handle ungültig.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **TextureInspectionResultID** (input_control) . . . . . texture_inspection_result(-array) ⇝ *handle*
Handle der Ergebnisse einer Texturinspektion.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Set parameter to generate a result handle
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gen_result_handle', 'true')
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModel, \
                                TextureInspectionResultID)
```

Ergebnis

Der Operator `clear_texture_inspection_result` gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextureInspectionResultID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`apply_texture_inspection_model`

Modul

Matching

`create_texture_inspection_model` (
`: : ModelType : TextureInspectionModel`)

Erzeugen eines Texturinspektionsmodells.

`create_texture_inspection_model` erzeugt ein neues Texturinspektionsmodell `TextureInspectionModel` vom Typ `ModelType`. Aktuell wird nur der Modelltyp *'basic'* unterstützt.

Die Parameter des Modells können mit `get_texture_inspection_model_param` abgefragt und mit `set_texture_inspection_model_param` manipuliert werden.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **`ModelType`** (input_control) ..... string ⇝ *string*
Typ des erzeugten Texturinspektionsmodells.
Defaultwert: 'basic'
Werteliste: `ModelType` ∈ {'basic'}

▷ **`TextureInspectionModel`** (output_control) ..... texture_inspection_model ⇝ *handle*
Handle des Texturinspektionsmodells.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModel, \
                                TextureInspectionResultID)
```

Ergebnis

Der Operator `create_texture_inspection_model` gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_texture_inspection_model_param`, `get_texture_inspection_model_param`, `add_texture_inspection_model_image`

Modul

Matching

deserialize_texture_inspection_model (: : SerializedItemHandle : TextureInspectionModel)

Deserialisieren eines Texturinspektionsmodells.

`deserialize_texture_inspection_model` deserialisiert ein Texturinspektionsmodell, welches mit `serialize_texture_inspection_model` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Texturinspektionsmodell wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden im Texturinspektionsmodell `TextureInspectionModel` gespeichert.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **TextureInspectionModel** (output_control) ..... texture_inspection_model ⇝ *handle*
Handle des Texturinspektionsmodells.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Set parameters
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gen_result_handle', 'true')
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Serialize texture inspection model
serialize_texture_inspection_model (TextureInspectionModel, \
                                    SerializedItemHandle)
* Deserialize in Model
deserialize_texture_inspection_model (SerializedItemHandle, \
```

```
                                        TextureInspectionModelSerialized)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModelSerialized, \
                                TextureInspectionResultID)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_texture_inspection_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextureInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_texture_inspection_model`

Nachfolger

`apply_texture_inspection_model`

Siehe auch

`create_texture_inspection_model`, `write_texture_inspection_model`, `serialize_texture_inspection_model`

Modul

Matching

get_texture_inspection_model_image

```
get_texture_inspection_model_image (
    : ModelImages : TextureInspectionModel : )
```

Abfragen der Bilder eines Texturinspektionsmodells.

Der Operator `get_texture_inspection_model_image` gibt alle Bilder, die dem Texturinspektionsmodell `TextureInspectionModel` mit `add_texture_inspection_model_image` hinzugefügt wurden, zurück.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **ModelImages** (output_object) ..... (multichannel-)image-array ⇝ *object :* byte / uint2
Im Texturinspektionsmodell enthaltene Trainingsbilder.

▷ **TextureInspectionModel** (input_control) ..... texture_inspection_model ⇝ *handle*
Handle des Texturinspektionsmodells.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
```

```
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (ModelImages, TextureInspectionModel, \
                                    Indices)
* Get added training images
get_texture_inspection_model_image (ModelImagesOut, TextureInspectionModel)
```

Ergebnis

Der Operator `get_texture_inspection_model_image` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_texture_inspection_model`, `set_texture_inspection_model_param`, `get_texture_inspection_model_param`, `add_texture_inspection_model_image`

Nachfolger

`train_texture_inspection_model`, `clear_texture_inspection_model`, `remove_texture_inspection_model_image`

Modul

Matching

get_texture_inspection_model_param (: : TextureInspectionModel, GenParamName : GenParamValue)

Abfragen von Parametern eines Texturinspektionsmodells.

Der Operator `get_texture_inspection_model_param` wird genutzt, um die Werte der verschiedenen Parameter des Texturinspektionsmodells `TextureInspectionModel` abzufragen. Die Namen der gewünschten Parameter werden im Parameter `GenParamName` übergeben. Die zugehörigen Werte werden in `GenParamValue` zurückgegeben. Es können mehrere Parameter mit einem einzigen Aufruf von `get_texture_inspection_model_param` abgefragt werden. Es können alle Parameter abgefragt werden, die mit `set_texture_inspection_model_param` gesetzt werden können. Für die Beschreibung der einzelnen Parameter siehe die Dokumentation von `set_texture_inspection_model_param`. Darüber hinaus ist es möglich, die folgenden Parameter abzurufen:

'gmm_centers': Gibt die Anzahl der gaußschen Verteilungen zurück, die im Trainingsprozess als optimal ermittelt wurde. Diese Rückgabe kann als Referenz für zukünftig zu erstellende Texturmodelle dienen. Der Wertebereich für die mögliche Anzahl an Verteilungen wird durch die Festlegung der Parameter *'gmm_pmincenters'* und *'gmm_pmaxcenters'* bestimmt. Jede Bildpyramidenstufe kann eine unterschiedliche Anzahl an gaußschen Verteilungen aufweisen. Durch das Anhängen eines Indizes an den Parameter lässt sich die Anzahl der gaußschen Verteilungen für eine bestimme Bildpyramidenstufe anzeigen. *'gmm_centers_3'* liefert beispielsweise die Anzahl der gaußschen Verteilungen für die dritte Bildpyramidenstufe.

'image_indices': Gibt die Indices der Bilder zurück, die dem Texturinspektionsmodell mit `add_texture_inspection_model_image` hinzugefügt wurden. Es werden nur die Indices der Bilder zurückgegeben, die sich aktuell im Modell befinden, also nicht mit `remove_texture_inspection_model_image` entfernt wurden.

'trained_covartype': Liefert den Typ der Kovarianzmatrix, der während des Trainings verwendet wurde. Der Benutzer hat die Möglichkeit die bevorzugte Art der Kovarianzmatrix mit dem Setzen von *'gmm_covartype'* in `set_texture_inspection_model_param` einzustellen. Wenn jedoch das Training mit dem bevorzugten Kovarianz-Typ fehlschlägt, wird das Training mit einem Kovarianz-Typ mit weniger Freiheitsgeraden neu initialisiert. Wenn zum Beispiel das Training mit *'diag'* scheitert, wird ein weiteres Training mit *'spherical'* ausgeführt.

Zur besseren Übersicht können Parametern die tupelwertig sein können nur einzeln und nicht zusammen mit anderen Parametern abgefragt werden. Dazu gehören die Parameter *'image_indices'*, *'gmm_centers'*, *'levels'* und *'novelty_threshold'*.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **TextureInspectionModel** (input_control) . . . . . texture_inspection_model ⇝ *handle*
Handle des Texturinspektionsmodells.

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Name des abgefragten Modellparameters.
Defaultwert: 'novelty_threshold'
Werteliste: `GenParamName` ∈ {'num_levels', 'levels', 'gen_result_handle', 'novelty_threshold', 'sensitivity', 'patch_normalization', 'patch_rotational_robustness', 'patch_size', 'gmm_centers', 'gmm_sigma', 'gmm_pmincenters', 'gmm_pmaxcenters', 'gmm_preprocessing', 'gmm_ncomp', 'gmm_randseed', 'gmm_em_max_iter', 'gmm_em_threshold', 'gmm_em_regularize', 'gmm_covartype', 'image_indices', 'trained_covartype'}

▷ **GenParamValue** (output_control) . . . . . string(-array) ⇝ *integer* / real / string
Wert des abgefragten Modellparameters.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Get parameters
get_texture_inspection_model_param (TextureInspectionModel, 'patch_size', \
                                    PatchSize)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModel, \
                                TextureInspectionResultID)
```

Ergebnis

Der Operator `get_texture_inspection_model_param` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_texture_inspection_model`, `train_texture_inspection_model`

Nachfolger

`add_texture_inspection_model_image`, `train_texture_inspection_model`

Siehe auch

`set_texture_inspection_model_param`

Modul

Matching

get_texture_inspection_result_object (
: Object : TextureInspectionResultID, ResultName :)

Abfragen von ikonischen Ergebnissen einer Texturinspektion.

get_texture_inspection_result_object fragt das ikonische Ergebnis ResultName im von apply_texture_inspection_model zurückgegebenen Handle TextureInspectionResultID ab. TextureInspectionResultID wurde nur dann erzeugt, wenn *'gen_result_handle'* mit set_texture_inspection_model_param auf *'true'* gesetzt wurde.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Im Folgenden sind die möglichen Parameterwerte für ResultName gelistet:

'novelty_region': Gibt die Fehlerregionen für alle Bildpyramidenstufen in einem Objekttupel zurück. Um die Fehlerregion einer einzelnen Bildpyramidenstufe abzufragen, muss der jeweilige Index angehängt werden, z.B. *'novelty_region_1'*. Wenn ein Tupel von Bildern an apply_texture_inspection_model übergeben wurde, können nur die Fehlerregionen für das erste Bild dieses Tupels abgefragt werden.

'novelty_score_image': Gibt die Novelty-Score-Bilder für alle Bildpyramidenstufen in einem Objekttupel zurück. Um die Novelty-Score-Bilder einer einzelnen Bildpyramidenstufe abzufragen, muss der jeweilige Index angehängt werden, z.B. *'novelty_score_image_1'*. Wenn ein Tupel von Bildern an apply_texture_inspection_model übergeben wurde, können nur die Novelty-Score-Bilder für das erste Bild dieses Tupels abgefragt werden.

Parameter

▷ **Object** (output_object) ..... object(-array) ⇝ *object*
Zurückgegebene ikonische Ergebnisse.

▷ **TextureInspectionResultID** (input_control) ..... texture_inspection_result ⇝ *handle*
Handle des Texturinspektionsergebnisses.

▷ **ResultName** (input_control) ..... string(-array) ⇝ *string* / integer
Name der zurückzugebenden ikonischen Ergebnisse.
Defaultwert: 'novelty_region'
Werteliste: ResultName ∈ {'novelty_score_image', 'novelty_region'}

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Set parameters
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gen_result_handle', 'true')
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModel, \
                                TextureInspectionResultID)
* Get result objects
get_texture_inspection_result_object (Object, TextureInspectionResultID, \
                                      'novelty_score_image')
```

Ergebnis

Sind alle Parameter gültig, gibt get_texture_inspection_result_object TRUE zurück. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`apply_texture_inspection_model`

Nachfolger

`clear_texture_inspection_result`, `clear_texture_inspection_model`

Modul

Matching

read_texture_inspection_model (
: : FileName : TextureInspectionModel)

Lesen eines Texturinspektionsmodells aus einer Datei.

`read_texture_inspection_model` liest aus der durch `FileName` gegebenen Datei ein mit `write_texture_inspection_model` abgespeichertes Texturinspektionsmodell ein. Da das Trainieren eines Texturinspektionsmodells relativ lange dauern kann, wird das Texturinspektionsmodell typischerweise in einem Offline-Prozess trainiert und mit `write_texture_inspection_model` gespeichert. Im Online-Prozess wird das Texturinspektionsmodell mit `read_texture_inspection_model` gelesen und danach zur Evaluierung mit `apply_texture_inspection_model` verwendet. Die Dateiendung für das Texturinspektionsmodell ist in HALCON 'htim'.

Nach dem Einlesen eines bereits trainierten Texturinspektionsmodell, kann der Operator `apply_texture_inspection_model` aufgerufen werden, um Testbilder zu klassifizieren. Ein erneutes Trainieren des Modells kann nur erfolgen, wenn das eingelesene Modell über Bilder verfügt. Außerdem ist zu beachten, dass die Modifikation von Parametern des Modells ein komplettes Neutraining zur Folge hat.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **FileName** (input_control) ..... filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.htim`

▷ **TextureInspectionModel** (output_control) ..... texture_inspection_model ⇝ *handle*
Handle des Texturinspektionsmodells.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)

* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Write out texture inspection model
write_texture_inspection_model (TextureInspectionModel, \
                                'ExampleModel.htim')
* Read in Model
```

```
read_texture_inspection_model ('ExampleModel.htim', \
                               TextureInspectionModelRead)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModelRead, \
                                TextureInspectionResultID)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_texture_inspection_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`write_texture_inspection_model`

Nachfolger

`apply_texture_inspection_model`

Siehe auch

`create_texture_inspection_model`, `clear_texture_inspection_model`, `set_texture_inspection_model_param`, `get_texture_inspection_model_param`, `add_texture_inspection_model_image`

Modul

Matching

remove_texture_inspection_model_image (: : TextureInspectionModel, Indices : RemainingIndices)

Löschen ausgewählter oder aller Bilder eines Texturinspektionsmodells.

`remove_texture_inspection_model_image` löscht alle oder einzelne, ausgewählte Bilder eines Texturinspektionsmodells `TextureInspectionModel`.

Eine Verwendung von `remove_texture_inspection_model_image` ist einerseits zu empfehlen, um den Speicherplatz eines abgespeicherten Modells zu reduzieren. Dies ist vor allem dann von Vorteil, wenn das Texturinspektionsmodell offline trainiert und mit `write_texture_inspection_model` abgespeichert wurde, da in diesem Fall die Testbilder mit Hilfe von `apply_texture_inspection_model` weiterhin klassifiziert werden können, der benötigte Speicherplatz jedoch auf ein Minimum reduziert wurde. Andererseits können damit auf einfach Weise Bilder, die für eine spätere Klassifizierung von Testbildern nicht (mehr) benötigt werden, aus dem Modell entfernt werden. Dies ermöglicht es den Nutzer mit einem vollständig konfigurierten Texturmodell weiterzuarbeiten während er lediglich die verwendeten Bilddaten auf das zu lösende Inspektionsproblem abstimmen muss. Nach dem Entfernen von Bildern aus dem Texturinspektionsmodell muss es neu trainiert werden.

Um gezielt Bilder aus dem Texturinspektionsmodells zu entfernen, können im Parameter `Indices` die Indices der entsprechenden Bilder angegeben werden. Eine Überprüfung, welche Bilder sich noch im Texturinspektionsmodell befinden, kann mit dem Operator `get_texture_inspection_model_param` unter Angabe des Wertes *'image_indices'* erfolgen. Sollte der Nutzer noch nicht vergebene Indices zu Löschen von Bildern verwenden, so werden diese ignoriert. `remove_texture_inspection_model_image` bietet außerdem die Möglichkeit an, `Indices` auf den Wert *'all'* zu setzen. Damit werden automatisch alle Bilder aus dem Texturinspektionsmodells entfernt.

Falls mit `remove_texture_inspection_model_image` alle hinzugefügten Bilder aus dem Modell entfernt wurden, so kann das Modell erst (wieder) trainiert werden, wenn mit `add_texture_inspection_model_image` neue Bilder hinzugefügt wurden. Falls nur ein Teil der Bilder aus dem Modell gelöscht wurde, so werden bei einem Aufruf von `train_texture_inspection_model` alle Schritte des Trainings ausgeführt, da die ggf. noch vorhandenen Trainingsdaten nicht mehr mit der zu diesem Zeitpunkt verwendeten Datengrundlage übereinstimmen.

`remove_texture_inspection_model_image` gibt im Parameter `RemainingIndices` die Indices der Bilder zurück die nach dem Aufruf noch im Texturinspektionsmodells vorhanden sind.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

- ▷ **`TextureInspectionModel`** (input_control) . . . . . texture_inspection_model(-array) ⇝ *handle*
  Handle des Texturinspektionsmodells.
- ▷ **`Indices`** (input_control) . . . . . integer-array ⇝ *integer*
  Indices der Bilder, die aus dem Texturinspektionsmodell entfernt werden sollen.
- ▷ **`RemainingIndices`** (output_control) . . . . . integer-array ⇝ *integer*
  Indices der Bilder, die weiterhin vom Texturinspektionsmodell verwendet werden.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Get added training images
get_texture_inspection_model_image (TrainImageOut, TextureInspectionModel)
* Remove training images
remove_texture_inspection_model_image (TextureInspectionModel, Indices, \
                                       RemainingIndices)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `remove_texture_inspection_model_image` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextureInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`train_texture_inspection_model`

Siehe auch

`create_texture_inspection_model`, `clear_texture_inspection_model`, `set_texture_inspection_model_param`, `get_texture_inspection_model_param`, `add_texture_inspection_model_image`

Modul

Matching

serialize_texture_inspection_model (
: : TextureInspectionModel : SerializedItemHandle)

Serialisieren eines Texturinspektionsmodells.

serialize_texture_inspection_model serialisiert ein Texturinspektionsmodell (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das Texturinspektionsmodell wird in dem Handle TextureInspectionModel übergeben. Das serialisierte Texturinspektionsmodell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_texture_inspection_model wieder deserialisiert werden.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

- ▷ **TextureInspectionModel** (input_control) ..... texture_inspection_model ⇝ *handle*
  Handle des Texturinspektionsmodells.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Set parameters
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gen_result_handle', 'true')
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Serialize texture inspection model
serialize_texture_inspection_model (TextureInspectionModel, \
                                    SerializedItemHandle)
* Deserialize in Model
deserialize_texture_inspection_model (SerializedItemHandle, \
                                      TextureInspectionModelSerialized)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModelSerialized, \
                                TextureInspectionResultID)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_texture_inspection_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_texture_inspection_model

Nachfolger

clear_texture_inspection_model, fwrite_serialized_item, send_serialized_item, deserialize_texture_inspection_model

Siehe auch

create_texture_inspection_model, read_texture_inspection_model, deserialize_texture_inspection_model

Modul

Matching

set_texture_inspection_model_param (: : TextureInspectionModel, GenParamName, GenParamValue :)

Setzen von Parametern eines Texturinspektionsmodells.

Der Operator set_texture_inspection_model_param wird benutzt, um die Parameter des Texturinspektionsmodells TextureInspectionModel anzupassen. Die aktuell gesetzten Parameterwerte können mit dem Operator get_texture_inspection_model_param abgefragt werden.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Allgemeine Parameter:

'num_levels': Bestimmt die Anzahl an Bildpyramidenstufen die zur Inspektion von Texturen verwendet werden. Die Anzahl der verwendeten Bildpyramidenstufen kann sowohl das Ergebnis der Texturinspektion als auch die Laufzeit von train_texture_inspection_model und apply_texture_inspection_model beeinflussen. Standardmäßig wird die Anzahl der Bildpyramidenstufen über die Größe der Trainingsbilder bestimmt. Hierbei ist zu beachten dass nicht mehr als 5 Bildpyramidenstufen verwendet werden.

Das Ändern dieses Parameters erfordert ein erneutes Trainieren des Texturinspektionsmodells wenn die angegebenen Skalen vorher nicht schon trainiert wurden. Reduziert man bei einem trainierten Modell zum Beispiel die Anzahl der Skalen von 4 auf 3 muss nicht neu trainiert werden.

Ist die Textur in den Bildern sehr grob, werden die niedrigeren Bildpyramidenstufen unter Umständen nicht für eine erfolgreiche Inspektion benötigt. Dann kann die Laufzeit deutlich optimiert werden indem man diese Skalen explizit durch das Setzen von *'levels'* ausschließt.

Werteliste: integer Werte größer oder gleich 1 oder *'auto'*

Defaultwert: 'auto'

'levels': Bestimmt die Bildpyramidenstufen die zur Inspektion von Texturen verwendet werden. Die Anzahl der verwendeten Bildpyramidenstufen kann sowohl das Ergebnis der Texturinspektion als auch die Laufzeit von train_texture_inspection_model und apply_texture_inspection_model beeinflussen. Standardmäßig werden die Bildpyramidenstufen durch den Wert von *'num_levels'* bestimmt: Wird zum Beispiel *'num_levels'* auf 4 gesetzt wird das Training und die Klassifikation auf vier Skalen durchgeführt. Um alle verfügbaren Skalen zu trainieren, sollte der Wert auf *'auto'* gesetzt werden.

Das Ändern dieses Parameters erfordert ein erneutes Trainieren des Texturinspektionsmodells wenn die angegebene Skalen vorher nicht schon trainiert wurde.

Ist die Textur in den Bildern sehr grob, werden die niedrigeren Bildpyramidenstufen unter Umständen nicht für eine erfolgreiche Inspektion benötigt. Dann kann die Laufzeit deutlich optimiert werden indem man diese Skalen explizit ausschließt.

Um Verwirrungen über die Länge des Tupels zu vermeiden, kann *'levels'* nur einzeln und nicht zusammen mit anderen Parametern gesetzt werden.

Werteliste: ein monoton steigendes Tupel von integer Werten größer oder gleich 1, oder *'auto'*

Defaultwert: 'auto'

'gen_result_handle': Dieser Parameter bestimmt, ob beim Aufruf von apply_texture_inspection_model ein Handle zur detaillierteren Inspektion der Ergebnisse erstellt wird. Setzt man *'gen_result_handle'* auf *'true'*, lassen sich mit get_texture_inspection_result_object aus dem Handle Ergebnisse für die einzelnen Bildpyramidenstufen abfragen. Dies kann beim Debuggen und bei der Parameterfeinabstimmung helfen.

Für den Wert *'false'* wird beim Aufruf von apply_texture_inspection_model lediglich die detektierte Fehlerregion ausgegeben.

Werteliste: 'true', 'false'

Defaultwert: 'false'

Parameter, die die Berechnung der automatischen Fehlerschwelle beeinflussen:

'novelty_threshold': Dieser Parameter setzt die Werte für die Fehlerschwellen, welche zwischen fehlerfreier und fehlerhafter Textur unterscheiden. Die Fehlerschwellen werden als Tupel übergeben und werden für alle Bildpyramidenstufen gleichzeitig gesetzt. Dementsprechend muss die Anzahl der Elemente mit der Anzahl der Pyramidenstufen (*'num_levels'*) übereinstimmen.

Das Setzen dieses Parameters vor dem Training hat keinen Effekt, da das Training die Fehlerschwellen automatisch bestimmt. Ein Anpassen von *'novelty_threshold'* sollte daher nach dem Training erfolgen. Eine typische Anwendung ist die Feinabstimmung zum Optimieren der Ergebnisse von apply_texture_inspection_model. Bitte beachten Sie, ein Aufruf von train_texture_inspection_model schätzt die Fehlerschwellen automatisch und überschreibt somit explizit gesetzte Fehlerschwellen.

Durch das Anhängen eines Indizes lässt sich die Fehlerschwelle für eine bestimme Bildpyramidenstufe setzen. *'novelty_threshold_3'* beispielsweise legt die Fehlerschwelle für die dritte Bildpyramidenstufe fest.

Werteliste: integer oder float Werte größer oder gleich 0 und kleiner oder gleich 710

Defaultwert: Wird in train_texture_inspection_model berechnet

'sensitivity': Dieser Parameter beeinflusst die Empfindlichkeit der im Training mit train_texture_inspection_model automatisch bestimmten Fehlerschwellen. Standardmäßig ist der Wert 0.0 gesetzt, was dazu führt, dass die Fehlerschwellen aus dem Training übernommen werden. Bei einem Aufruf von apply_texture_inspection_model wird eine negative Sensitivität auf die im Training bestimmten Fehlerschwellen addiert. Dementsprechend führt eine Sensitivität unter 0.0 zu höheren Fehlerschwellen und dadurch zu potentiell weniger gefundenen Fehlern. Positive Werte hingegen führen zu niedrigeren Fehlerschwellen und dadurch zu potentiell mehr entdeckten Fehlern.

Die automatisch festgesetzten Fehlerschwellen können nach dem Training mit dem Parameter *'novelty_threshold'* geändert werden.

Werteliste: float oder integer Wert

Defaultwert: 0.0

Parameter, die die Texturmerkmale beeinflussen:

'patch_normalization': Bestimmt die Methode mit der die Texturmerkmale normiert werden. Eine Normierung ist dann sinnvoll, wenn zwischen den verschiedenen Aufnahmen keine konsistenten Beleuchtungsbedingungen gegeben sind. Wird der Parameter auf *'weber'* gesetzt, werden die Merkmale gemäß dem Weber-Fechner-Gesetz normiert. Wird der Parameter auf *'none'* gesetzt, werden die Merkmale nicht normiert.

Eine Änderung dieses Parameters erfordert ein erneutes vollständiges Training des Texturinspektionsmodells.

Werteliste: 'none', 'weber'

Defaultwert: 'none'

'patch_rotational_robustness': Wird dieser Parameter auf *'true'* gesetzt, werden die Texturmerkmale sortiert und dadurch zu einem gewissen Grad Rotationsinvarianz erreicht. Das Sortieren der Merkmale führt jedoch zu einem Informationsverlust bei den Merkmalen und die Texturinspektion wird generell weniger sensitiv. Außerdem ist die Laufzeit etwas länger. Wird der Parameter auf *'false'* gesetzt, werden die Texturmerkmale nicht sortiert.

Eine Änderung dieses Parameters erfordert ein erneutes vollständiges Training des Texturinspektionsmodells.

Alternativ ist es auch möglich, alle möglichen Rotationen in den Trainingsbildern abzudecken. Die Erhöhung der Anzahl der Trainingsbilder führt zu einer längeren Trainingszeit.

Werteliste: 'true', 'false'

Defaultwert: 'false'

'patch_size': Dieser Parameter bestimmt den Durchmesser der extrahierten Texturmerkmalspatches in Pixeln. Größere Patchgrößen erhöhen die Laufzeit deutlich. Daher sollte dieser Parameter nur mit großer Vorsicht angepasst werden. Zuerst sollte versucht werden, die Auflösung der Bilder oder die Anzahl der benutzten Pyramidenstufen zu reduzieren.

Eine Änderung dieses Parameters erfordert ein erneutes vollständiges Training des Texturinspektionsmodells.

Werteliste: ungerade integer Werte zwischen 1 und 11

Defaultwert: 5

Weitergehende Parameter, die die GMM-Klassifikatoren beeinflussen:

Intern nutzt die Texturinspektion einen auf Gaussian Mixture Models (GMM) basierenden Klassifikator. Generell wird die Änderung der GMM-Parameter nicht empfohlen, ist für erfahrene Nutzer aber trotzdem möglich.

Eine Änderung dieser Parameter erfordert ein erneutes vollständiges Training des Texturinspektionsmodells.

Im Folgenden werden die Parameter, die den GMM-Klassifikator beeinflussen, beschrieben. Für weitere Informationen zum GMM-Klassifikator siehe auch `create_class_gmm` und `train_class_gmm`.

'gmm_sigma': Dieser Parameter bestimmt die Standardabweichung des gaußschen Rauschens, welches zu den Trainingsdaten hinzugefügt wird (siehe `add_sample_class_gmm`).

Werteliste: integer oder float Werte größer gleich 0

Defaultwert: 0.0

'gmm_pmincenters': Dieser Parameter bestimmt die minimale Anzahl der gaußschen Verteilungen pro Klasse. *'gmm_pmincenters'* kann nicht auf einen Wert gesetzt werden, der größer ist als der aktuelle Wert von *'gmm_pmaxcenters'*.

Werteliste: integer Wert größer gleich 1 und kleiner gleich *'gmm_pmaxcenters'*

Defaultwert: 12

'gmm_pmaxcenters': Dieser Parameter bestimmt die maximale Anzahl der gaußschen Verteilungen pro Klasse. *'gmm_pmaxcenters'* kann nicht auf einen Wert gesetzt werden, der kleiner ist als der aktuelle Wert von *'gmm_pmincenters'*.

Werteliste: integer Wert größer gleich *'gmm_pmincenters'*

Defaultwert: 12

'gmm_preprocessing': Dieser Parameter bestimmt die Art der Vorverarbeitung, die zur Transformation der Merkmalsvektoren genutzt wird.

Werteliste: 'principal_components', *'normalization'*, *'none'*

Defaultwert: 'normalization'

'gmm_ncomp': Dieser Parameter steuert die Vorverarbeitung im Falle dass eine Hauptkomponentenanalyse (*'principal_components'*) als Vorverarbeitung ausgewählt wurde. Speziell legt es die Dimension des transformierten Merkmalsvektors fest. *'gmm_ncomp'* muss immer kleiner als die Merkmalsdimension sein, die (*'patch_size'* $\times$ *'patch_size'*) beträgt. Wird *'patch_size'* so gesetzt, dass *'patch_size'* $\times$ *'patch_size'* $<$ *'gmm_ncomp'*, so wird der Parameter entsprechend angepasst.

Werteliste: integer Werte größer oder gleich 1

Defaultwert: 15

'gmm_randseed': Dieser Parameter bestimmt den Initialisierungswert des Zufallszahlengenerators, der zur Initialisierung des GMM mit zufälligen Werten verwendet wird.

Werteliste: integer Wert

Defaultwert: 42

'gmm_em_max_iter': Dieser Parameter bestimmt die maximale Anzahl von Iterationen des Expectation-Maximization-Algorithmus.

Werteliste: integer Werte größer gleich 0

Defaultwert: 100

'gmm_em_threshold': Dieser Parameter bestimmt den Schwellenwert für die relative Änderung des Fehlers, die nötig ist, damit der Expectation-Maximization-Algorithmus endet.

Werteliste: integer oder float Werte größer gleich 0 und kleiner als 1.0

Defaultwert: 0.001

'gmm_em_regularize': Dieser Parameter bestimmt den Regularisierungswert zur Vermeidung einer Singularität der Kovarianzmatrizen im Trainingsprozess.

Werteliste: integer oder float Werte größer gleich 0 und kleiner als 1.0

Defaultwert: 0.0001

'gmm_covartype': Dieser Parameter bestimmt die Art der Kovarianzmatrix die intern verwendet wird.
Werteliste: 'spherical', 'diag', 'full'
Defaultwert: 'diag'

Ändern von Parametern eines trainierten Texturinspektionsmodells

Werden nach dem erfolgreichen Aufruf von `train_texture_inspection_model` die Werte von Parametern umgesetzt, so ist in den meisten Fällen ein erneutes Training notwendig. Je nachdem welche Parameter geändert werden, müssen verschiedene Teile des Trainings neu ausgeführt werden.

Ein Anpassen der Parameter *'gmm_*'* erfordert ein erneutes Training der Klassifikatoren. Da ein neuer Klassifikator zu abweichenden Neuheitswerten (*novelty scores*) führt, müssen auch die Fehlerschwellen (Novelty Threshold) neu berechnet werden. Die Parameter *'patch_*'* erfordern ein komplettes erneutes Training des Texturinspektionsmodells. Das Umsetzten von *'num_levels'* oder *'levels'* erfordert in der Regel ein komplettes Training des Texturinspektionsmodells. Werden die verwendeten Skalen aber nur reduziert, zum Beispiel durch das Reduzieren von *'levels'* von [1,2,3] auf [1,3], wird kein neues Training benötigt, da die Skalen 1 und 3 bereits in einem vorherigen Schritt trainiert wurden.

Der Parameter *'gen_result_handle'* hat keinerlei Auswirkung auf ein trainiertes Texturinspektionsmodell, so dass er geändert werden kann, ohne dass ein erneutes Training nötig ist.

Parameter

- ▷ **`TextureInspectionModel`** (input_control) ..... texture_inspection_model ⇝ *handle*
  Handle des Texturinspektionsmodells.
- ▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
  Name des zu ändernden Parameters.
  Defaultwert: 'gen_result_handle'
  Werteliste: `GenParamName` ∈ {'num_levels', 'levels', 'gen_result_handle', 'novelty_threshold', 'sensitivity', 'patch_normalization', 'patch_rotational_robustness', 'patch_size', 'gmm_sigma', 'gmm_pmincenters', 'gmm_pmaxcenters', 'gmm_preprocessing', 'gmm_ncomp', 'gmm_randseed', 'gmm_em_max_iter', 'gmm_em_threshold', 'gmm_em_regularize', 'gmm_covartype'}
- ▷ **`GenParamValue`** (input_control) ..... attribute.value(-array) ⇝ *integer* / real / string
  Neuer Wert des Parameters.
  Defaultwert: 'true'
  Werteliste: `GenParamValue` ∈ {5, 'auto', 'true', 'false', 'none', 'weber', 'principal_components', 'normalization'}

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Set parameters
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gen_result_handle', 'true')
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModel, \
                                TextureInspectionResultID)
```

Ergebnis

Der Operator `set_texture_inspection_model_param` gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextureInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_texture_inspection_model`

Nachfolger

`add_texture_inspection_model_image`, `train_texture_inspection_model`

Siehe auch

`get_texture_inspection_model_param`

Modul

Matching

`train_texture_inspection_model` (: : `TextureInspectionModel` :)

Trainieren eines Texturinspektionsmodells.

`train_texture_inspection_model` trainiert ein Texturinspektionsmodell mit allen Trainingsbildern, die dem Modell mit `add_texture_inspection_model_image` zugefügt wurden.

Der komplette Texturinspektionsprozess arbeitet mit Bildpyramiden. Die Tiefe der Pyramide wird durch den Parameter *'num_levels'* (oder falls explizite Skalen gesetzt sind, von *'levels'*) bestimmt. Im ersten Trainingsschritt werden für jede Pyramidenstufe Texturmerkmale extrahiert und zum Set der Trainingsdaten zugefügt. Für jede Pyramidenstufe wird mit allen Trainingsdaten der jeweiligen Pyramidenstufe ein auf Gaussian Mixture Models (GMM) basierender Klassifikator bestimmt. In einem dritten Schritt werden die Trainingsdaten dann genutzt, um die Fehlerschwelle für jede Bildpyramidenstufe zu ermitteln. Im Folgenden werden die drei Schritte des Trainings im Detail beschrieben:

1. **Die Merkmalsextraktion** extrahiert für jedes Pixel in den Trainingsbildern ein Texturmerkmal. Die Texturmerkmale werden entsprechend der Einstellungen der *'patch_*'* Parameter berechnet, welche mit `set_texture_inspection_model_param` geändert werden können. Jedes Texturmerkmal wird dann zu den Trainingsdaten des GMMs zugefügt.
2. **Während des Trainings** des GMMs werden aus den Trainingsdaten die optimalen *'gmm_*'* Parameter abgeleitet. Die Dimensionen der einzelnen Texturmerkmale werden durch die gewählte Größe der Patches bestimmt (siehe `set_texture_inspection_model_param`). Für große Patchgrößen kann es extrem schwierig sein, eine gute Approximation an die optimalen GMM Parameter zu bekommen. Außerdem erhöhen sich die Laufzeit und der benötigte Speicher signifikant. Daher wird empfohlen, die Patchgröße nur mit großer Vorsicht zu ändern. Weitere Informationen zu GMM Klassifikatoren sind bei der Beschreibung des Operators `create_class_gmm` zu finden.
3. **Die Berechnung der Fehlerschwelle (Novelty Threshold)** ist nötig, um zwischen fehlerhafter und fehlerfreier Textur unterscheiden zu können. Zunächst wird mit den in Schritt 2 bestimmten GMMs der Neuheitswert (*novelty score*) für jedes im Training genutzte Texturmerkmal berechnet. Dann werden basierend auf den sich ergebenden Neuheitswerten die Fehlerschwellen für jede Bildpyramidenstufe bestimmt. Die automatische Bestimmung der Fehlerschwellen kann durch den Parameter *'sensitivity'* beeinflusst werden, der mit `set_texture_inspection_model_param` gesetzt werden kann.

Nachdem das Training des Modells erfolgreich durchgeführt wurde, können Bilder mit `apply_texture_inspection_model` klassifiziert werden. Hier wird für jedes Pixel ein Neuheitswert festgesetzt und mit der Fehlerschwelle abgeglichen, die im dritten Schritt des Trainings bestimmt

wurde. Für optimale Ergebnisse können die Fehlerschwellen der einzelnen Bildpyramidenstufen mit set_texture_inspection_model_param angepasst werden.

Im Allgemeinen hat das Verändern aller Parameter, bis auf *'gen_result_handle'* und *'sensitivity'*, zur Folge, dass das Texturinspektionsmodell neu trainiert werden muss. Je nachdem an welcher Stelle im Algorithmus ein neu eingestellter Parameter eingreift, wird intern entweder das gesamte Training oder nur der benötigte Teil davon ausgeführt. Aus diesem Grund kann es zu deutlichen Geschwindigkeitsunterschieden bei den Aufrufen von train_texture_inspection_model kommen. Welche Teile des Trainings beim Umsetzen eines bestimmten Parameters betroffen sind, wird in set_texture_inspection_model_param beschrieben.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **TextureInspectionModel** (input_control) ..... texture_inspection_model ⇝ *handle*
Handle des Texturinspektionsmodells.

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Read and apply a test image
read_image (TestImage, 'carpet/carpet_02')
apply_texture_inspection_model (TestImage, DefectCandidates, \
                                TextureInspectionModel, \
                                TextureInspectionResultID)
```

Ergebnis

Der Operator train_texture_inspection_model gibt TRUE zurück, sofern alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextureInspectionModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

add_texture_inspection_model_image, set_texture_inspection_model_param

Nachfolger

apply_texture_inspection_model, clear_texture_inspection_model, remove_texture_inspection_model_image, write_texture_inspection_model, serialize_texture_inspection_model

Literatur

X. Xianghua, M. Mirmehdi: „TEXEMS: Texture Exemplars for Defect Detection on Random Textured Surfaces“; IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 29, No. 8; August 2007.

T. Boettger, M. Ulrich: „Real-time Texture Detection on Textured Surfaces with Compressed Sensing“; Pattern Recognition and Image Analysis, Vol. 26, No. 1, pp. 88-94; January 2016.

Modul

Matching

write_texture_inspection_model (: : TextureInspectionModel, FileName :)

Abspeichern eines Texturinspektionsmodells in einer Datei.

write_texture_inspection_model speichert das Texturinspektionsmodell TextureInspectionModel in der Datei FileName ab. Die Dateiendung für das Texturinspektionsmodell ist in HALCON 'htim'. write_texture_inspection_model wird typischerweise aufgerufen, nachdem das Texturinspektionsmodell mit train_texture_inspection_model trainiert wurde, kann jedoch auch zum Speichern eines untrainierten Modells verwendet werden. Das Texturinspektionsmodell kann mit read_texture_inspection_model wieder eingelesen werden. write_texture_inspection_model schreibt die im Modell gesetzten Parameter sowie eventuell hinzugefügte Bilder und bei einem trainierten Modell die Ergebnisse des Trainings.

Das Konzept der Texturinspektion ist bei der Einleitung zum Kapitel Inspektion / Texturinspektion beschrieben.

Parameter

▷ **TextureInspectionModel** (input_control) ..... texture_inspection_model ⇝ *handle*
Handle des Texturinspektionsmodells.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei.
Dateiendung: .htim

Beispiel

```
* Create texture inspection model
create_texture_inspection_model ('basic', TextureInspectionModel)
* Make this short example fast:
set_texture_inspection_model_param (TextureInspectionModel, \
                                    'gmm_em_max_iter', 1)
* Read and add training images
read_image (TrainImage, 'carpet/carpet_01')
add_texture_inspection_model_image (TrainImage, TextureInspectionModel, \
                                    Indices)
* Train the model
train_texture_inspection_model (TextureInspectionModel)
* Write out texture inspection model
write_texture_inspection_model (TextureInspectionModel, 'ExampleModel.htim')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_texture_inspection_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_texture_inspection_model

Nachfolger

clear_texture_inspection_model

Siehe auch

create_texture_inspection_model, clear_texture_inspection_model, set_texture_inspection_model_param, get_texture_inspection_model_param, add_texture_inspection_model_image

Modul

Matching

13.5 Variationsmodell

clear_train_data_variation_model (: : ModelID :)

Freigabe des Speichers der Trainingsdaten eines Variationsmodells.

clear_train_data_variation_model gibt den Speicher der Trainingsdaten eines Variationsmodells wieder frei. clear_train_data_variation_model dient dazu, Speicherplatz (sowohl im Hauptspeicher als auch beim Schreiben des Variationsmodells mit write_variation_model) zu sparen. clear_train_data_variation_model kann nur aufgerufen werden, falls das Variationsmodell mit prepare_variation_model zum Vergleich vorbereitet wurde. Nach dem Aufruf von clear_train_data_variation_model kann das Variationsmodell nur noch zum Bildvergleich mit compare_variation_model oder compare_ext_variation_model verwendet werden. Ein weiteres Training ist ausgeschlossen. Weiterhin können die zum Bildvergleich verwendeten Bilder nicht mehr mit get_variation_model ausgelesen werden. Falls dies erwünscht ist, muss get_variation_model vor clear_train_data_variation_model aufgerufen werden.

Parameter

▷ **ModelID** (input_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

Ergebnis

clear_train_data_variation_model liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

prepare_variation_model

Nachfolger

compare_variation_model, compare_ext_variation_model, write_variation_model

Modul

Matching

clear_variation_model (: : ModelID :)

Freigabe des Speichers eines Variationsmodells.

clear_variation_model gibt den Speicher eines Variationsmodells, das mit create_variation_model angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle ModelID ist nach dem Aufruf ungültig.

Parameter

▷ **ModelID** (input_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

Ergebnis

`clear_variation_model` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_variation_model`

Modul

Matching

compare_ext_variation_model (Image : Region : ModelID, Mode :)

Vergleich eines Bildes mit einem Variationsmodell.

`compare_ext_variation_model` vergleicht das Eingabebild `Image` mit dem Variationsmodell `ModelID`. `compare_ext_variation_model` ist eine Erweiterung von `compare_variation_model`, die mehr Modi zum Bildvergleich bietet. Bevor `compare_ext_variation_model` aufgerufen werden kann, müssen mit `prepare_variation_model` oder `prepare_direct_variation_model` die beiden internen Schwellenwertbilder erzeugt worden sein. Sei das aktuelle Bild `Image` durch $c(x, y)$ bezeichnet und die beiden internen Schwellenwertbilder mit $t_{u,l}$ (siehe `prepare_variation_model` oder `prepare_direct_variation_model`). Dann wird für `Mode` = *'absolute'* die Ausgaberegion `Region` aus allen Punkten bestimmt, die sich stark vom Modell unterscheiden, d.h. für die gilt:

$$c(x, y) > t_u(x, y) \vee c(x, y) < t_l(x, y) \ .$$

Diese Modus ist also identisch zu `compare_variation_model`. Für `Mode` = *'light'* besteht `Region` aus allen Punkten, die zu hell sind:

$$c(x, y) > t_u(x, y) \ .$$

Für `Mode` = *'dark'* besteht `Region` aus allen Punkten, die zu dunkel sind:

$$c(x, y) < t_l(x, y) \ .$$

Schließlich werden für `Mode` = *'light_dark'* in `Region` zwei Regionen zurückgeliefert. In der ersten Region ist das Ergebnis von `Mode` = *'light'*, in der zweiten Region das Ergebnis von `Mode` = *'dark'*. Die jeweiligen Regionen können mit `select_obj` selektiert werden.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2
Bild des zu vergleichenden Objekts.

▷ **Region** (output_object) . . . . . region(-array) ⇝ *object*
Region der Punkte, die sich stark vom Modell unterscheiden.

▷ **ModelID** (input_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Art des Vergleichs mit dem Variationsmodells.
Defaultwert: 'absolute'
Wertevorschläge: `Mode` ∈ {'absolute', 'light', 'dark', 'light_dark'}

Beispiel

```
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, \
                   'default', -1, 'default', 'testimages.seq', 'default', \
                   -1, -1, AcqHandle)
read_region (Region, 'model.reg')
area_center (Region, Area, RowRef, ColumnRef)
read_shape_model ('model.shm', TemplateID)
read_variation_model ('model.var', ModelID)
for K := 1 to 10000 by 1
    grab_image (Image, AcqHandle)
    find_shape_model (Image, TemplateID, 0, rad(360), 0.5, 1, 0.5, \
                      'true', 4, 0.9, Row, Column, Angle, Score)
    dev_display (Image)
    if (|Score| == 1)
        vector_angle_to_rigid (Row, Column, Angle, RowRef, \
                               ColumnRef, 0, HomMat2D)
        affine_trans_image (Image, ImageTrans, HomMat2D, 'constant', \
                            'false')
        compare_ext_variation_model (ImageTrans, RegionDiff, ModelID, \
                                     'light')
        dev_display (RegionDiff)
    endif
endfor
close_framegrabber (AcqHandle)
```

Ergebnis

`compare_ext_variation_model` liefert TRUE zurück, wenn alle Parameter korrekt sind und die internen Schwellenwertbilder zuvor mit `prepare_variation_model` oder `prepare_direct_variation_model` erzeugt worden sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`prepare_variation_model`, `prepare_direct_variation_model`

Nachfolger

`select_obj`, `connection`

Alternativen

`compare_variation_model`, `dyn_threshold`

Siehe auch

`get_thresh_images_variation_model`

Modul

Matching

compare_variation_model (Image : Region : ModelID :)

Vergleich eines Bildes mit einem Variationsmodell.

`compare_variation_model` vergleicht das Eingabebild `Image` mit dem Variationsmodell `ModelID`. Bevor `compare_variation_model` aufgerufen werden kann, müssen mit `prepare_variation_model` oder

prepare_direct_variation_model die beiden internen Schwellenwertbilder erzeugt worden sein. Sei das aktuelle Bild Image durch $c(x, y)$ bezeichnet und die beiden internen Schwellenwertbilder mit $t_{u,l}$ (siehe prepare_variation_model und prepare_direct_variation_model). Dann wird die Ausgaberegion Region aus allen Punkten bestimmt, die sich stark vom Modell unterscheiden, d.h. für die gilt:

$$c(x, y) > t_u(x, y) \vee c(x, y) < t_l(x, y) \ .$$

Falls nur zu helle oder zu dunkle Fehler segmentiert werden sollen, kann der Operator compare_ext_variation_model verwendet werden.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2
  Bild des zu vergleichenden Objekts.
- ▷ **Region** (output_object) ..... region(-array) ⇝ *object*
  Region der Punkte, die sich stark vom Modell unterscheiden.
- ▷ **ModelID** (input_control) ..... variation_model ⇝ *handle*
  ID des Variationsmodells.

Beispiel

```
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, \
                   'default', -1, 'default', 'testimages.seq', 'default', \
                   -1, -1, AcqHandle)
read_region (Region, 'model.reg')
area_center (Region, Area, RowRef, ColumnRef)
read_shape_model ('model.shm', TemplateID)
read_variation_model ('model.var', ModelID)
for K := 1 to 10000 by 1
    grab_image (Image, AcqHandle)
    find_shape_model (Image, TemplateID, 0, rad(360), 0.5, 1, 0.5, \
                      'true', 4, 0.9, Row, Column, Angle, Score)
    dev_display (Image)
    if (|Score| == 1)
        vector_angle_to_rigid (Row, Column, Angle, RowRef, \
                               ColumnRef, 0, HomMat2D)
        affine_trans_image (Image, ImageTrans, HomMat2D, 'constant', \
                            'false')
        compare_variation_model (ImageTrans, RegionDiff, ModelID)
        dev_display (RegionDiff)
    endif
endfor
close_framegrabber (AcqHandle)
```

Ergebnis

compare_variation_model liefert TRUE zurück, wenn alle Parameter korrekt sind und die internen Schwellenwertbilder zuvor mit prepare_variation_model oder prepare_direct_variation_model erzeugt worden sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

prepare_variation_model, prepare_direct_variation_model

Nachfolger

connection

Alternativen

compare_ext_variation_model, dyn_threshold

Siehe auch

get_thresh_images_variation_model

Modul

Matching

create_variation_model (: : Width, Height, Type, Mode : ModelID)

Erzeugung eines Variationsmodells zum Bildvergleich.

create_variation_model erzeugt ein Variationsmodell, das zum Bildvergleich verwendet werden kann. Der Handle auf das generierte Variationsmodell wird in ModelID zurückgeliefert.

Das Variationsmodell wird typischerweise verwendet, um korrekt gefertigte Objekte („gute Objekte“) von falsch gefertigten Objekten („schlechte Objekte“) zu unterscheiden. Vorausgesetzt wird hierbei, dass sich die Unterscheidung allein aufgrund der Grauwerte des Objektes treffen lässt.

Das Variationsmodell besteht aus einem idealen Bild des Objektes, mit dem das Bild eines Objektes später mit compare_variation_model oder compare_ext_variation_model verglichen werden soll, und einem Bild, das angibt, wie sehr das Objekt an seinen verschiedenen Punkten im Grauwert variieren kann. Die Größe der Bilder, mit denen das Variationsmodell trainiert wird und die später zum Vergleich verwendet werden, wird mit Width und Height festgelegt. Der Bildtyp der Trainings- und Vergleichsbilder wird mit Type angegeben.

Das Variationsmodell wird aus mehreren Bildern guter Objekte trainiert. Daher ist es unerlässlich, dass die Trainingsbilder die Objekte in derselben Lage zeigen. Falls dies nicht garantiert werden kann, kann die Lage des Objektes z.B. mittels Matching (siehe find_shape_model) bestimmt werden. Hierauf kann das Bild mit affine_trans_image in eine Standardlage rotiert werden.

Mit dem Parameter Mode kann festgelegt werden, wie das Bild des idealen Objektes und das zugehörige Variationsbild berechnet werden. Für Mode=*'standard'* wird das ideale Bild des Objektes aus dem Mittelwert aller Trainingsbilder an den jeweiligen Bildpunkten berechnet. Das zugehörige Variationsbild wird aus der Standardabweichung der Trainingsbilder an den jeweiligen Bildpunkten berechnet. Dieser Modus hat den Vorteil, dass das Modell iterativ trainiert werden kann, d.h. sobald ein Bild eines guten Objektes vorliegt, kann es mit train_variation_model trainiert werden. Der Nachteil dieses Modus ist, dass sehr genau darauf geachtet werden muss, dass ausschließlich Bilder guter Objekte trainiert werden, da der Mittelwert und die Standardabweichung nicht robust gegenüber Ausreißern sind, d.h. wenn ein Bild eines schlechten Objektes aus Versehen mittrainiert wird, kann die Genauigkeit des idealen Objektbildes und des Variationsbildes beeinträchtigt werden.

Falls nicht verhindert werden kann, dass das Variationsmodell mit Bildern von Objekten trainiert wird, die Fehler enthalten, kann Mode=*'robust'* verwendet werden. In diesem Modus wird das ideale Bild des Objektes aus dem Median aller Trainingsbilder an den jeweiligen Bildpunkten berechnet. Das zugehörige Variationsbild wird aus dem Median der absoluten Abweichungen der Trainingsbilder vom Median an den jeweiligen Bildpunkten berechnet und geeignet skaliert. Dieser Modus hat den Vorteil, dass das berechnete Variationsmodell robust gegenüber Ausreißern ist. Es hat den Nachteil, dass es nicht iterativ trainiert werden kann, d.h. alle Trainingsbilder müssen mit concat_obj aufgesammelt werden und in einem Aufruf von train_variation_model trainiert werden.

In manchen Fällen ist es nicht möglich, mehrere Trainingsbilder aufzunehmen. In diesem Fall kann durch Trainieren des einzigen Trainingsbildes kein sinnvolles Variationsbild erzeugt werden. Um dies zu umgehen, können synthetische Variationen des Trainingsbildes erzeugt werden, z.B. durch Verschieben des Trainingsbildes um ± 1 Pixel in Zeilen- und Spaltenrichtung oder durch Grauwertmorphologie (z.B. mit gray_erosion_shape und gray_dilation_shape) und dann trainiert werden. Eine andere Möglichkeit, ein Variationsmodell aus einem Trainingsbild zu erzeugen, ist, das Modell mit Mode=*'direct'* zu erzeugen. In diesem Fall kann das Variationsmodell nur dadurch trainiert werden, dass das ideale Bild des Objektes und das Variationsbild direkt mit prepare_direct_variation_model spezifiziert wird. Da typischerweise die Variation an den Kanten des Objektes groß ist, ist es sinnvoll, das Variationsbild mit Kantenoperatoren wie sobel_amp, edges_image oder gray_range_rect zu erzeugen.

Parameter

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite der zu vergleichenden Bilder.
Defaultwert: 640
Wertevorschläge: `Width` ∈ {160, 192, 320, 384, 640, 768}

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe der zu vergleichenden Bilder.
Defaultwert: 480
Wertevorschläge: `Height` ∈ {120, 144, 240, 288, 480, 576}

▷ **Type** (input_control) . . . . . string ⇝ *string*
Typ der zu vergleichenden Bilder.
Defaultwert: 'byte'
Wertevorschläge: `Type` ∈ {'byte', 'int2', 'uint2'}

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Art der Berechnung des Variationsmodells.
Defaultwert: 'standard'
Wertevorschläge: `Mode` ∈ {'standard', 'robust', 'direct'}

▷ **ModelID** (output_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

Komplexität

Das mit `create_variation_model` erzeugte Modell benötigt für `Mode` = *'standard'* und `Mode` = *'robust'* für `Type` = *'byte'* 12 ∗ `Width` ∗ `Height` Bytes Speicher. Für `Type` = *'uint2'* und `Type` = *'int2'* werden 14 ∗ `Width` ∗ `Height` Bytes benötigt. Für `Mode` = *'direct'* und nachdem die Trainingsdaten mit `clear_train_data_variation_model` gelöscht worden sind, werden für `Type` = *'byte'* 2 ∗ `Width` ∗ `Height` Bytes benötigt. Für die anderen Bildtypen werden 4 ∗ `Width` ∗ `Height` benötigt.

Ergebnis

`create_variation_model` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`train_variation_model`, `prepare_direct_variation_model`

Siehe auch

`prepare_variation_model`, `clear_variation_model`, `clear_train_data_variation_model`, `find_shape_model`, `affine_trans_image`

Modul

Matching

deserialize_variation_model (: : SerializedItemHandle : ModelID)

Deserialisieren eines Variationsmodells.

`deserialize_variation_model` deserialisiert ein Variationsmodell, welches mit `serialize_variation_model` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Variationsmodell wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Variationsmodell mit dem Handle `ModelID` gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **ModelID** (output_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_variation_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_variation_model`

Nachfolger

`compare_variation_model`, `compare_ext_variation_model`

Siehe auch

`read_variation_model`, `write_variation_model`

Modul

Matching

get_thresh_images_variation_model (: MinImage, MaxImage : ModelID :)

Auslesen der zum Bildvergleich verwendeten Schwellenwertbilder.

`get_thresh_images_variation_model` gibt die in dem Variationsmodell `ModelID` gespeicherten Schwellenwertbilder in `MinImage` und `MaxImage` zurück. Die Schwellenwertbilder müssen mit `prepare_variation_model` oder `prepare_direct_variation_model` berechnet werden, bevor sie ausgelesen werden können. Die Formel zur Berechnung der Schwellenwertbilder kann bei `prepare_variation_model` oder `prepare_direct_variation_model` nachgelesen werden. Die Schwellenwertbilder werden bei `compare_variation_model` und `compare_ext_variation_model` verwendet, um zu große Abweichungen eines Bildes vom Modell zu erkennen. Wie bei `compare_variation_model` und `compare_ext_variation_model` beschrieben, werden Grauwerte außerhalb des Intervalls, das durch `MinImage` und `MaxImage` gegeben ist, als Fehler betrachtet.

Parameter

▷ **MinImage** (output_object) . . . . . image ⇝ *object :* byte / int2 / uint2
Schwellenwertbild für den unteren Schwellenwert.

▷ **MaxImage** (output_object) . . . . . image ⇝ *object :* real
Schwellenwertbild für den oberen Schwellenwert.

▷ **ModelID** (input_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

Ergebnis

`get_thresh_images_variation_model` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

prepare_variation_model, prepare_direct_variation_model

Siehe auch

compare_variation_model, compare_ext_variation_model

Modul

Matching

get_variation_model (: Image, VarImage : ModelID :)

Auslesen der zum Bildvergleich verwendeten Bilder.

get_variation_model gibt die in dem Variationsmodell ModelID gespeicherten Bilder des idealen Objektes in Image und des zugehörigen Variationsbildes in VarImage zurück. Die zurückgelieferten Bilder können verwendet werden, um zu überprüfen, ob das Bild eines schlechten Objektes mit train_variation_model trainiert worden ist. Dies kann typischerweise daran erkannt werden, dass an Stellen des idealen Objektes, die keine Variation aufweisen sollten, eine große Variation im Variationsbild zu erkennen ist.

Parameter

- ▷ **Image** (output_object) ..... image ⇝ *object :* byte / int2 / uint2
  Bild des trainierten Objekts.
- ▷ **VarImage** (output_object) ..... image ⇝ *object :* real
  Variationsbild des trainierten Objekts.
- ▷ **ModelID** (input_control) ..... variation_model ⇝ *handle*
  ID des Variationsmodells.

Ergebnis

get_variation_model liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_variation_model

Siehe auch

prepare_variation_model, compare_variation_model, compare_ext_variation_model

Modul

Matching

prepare_direct_variation_model (RefImage, VarImage : : ModelID, AbsThreshold, VarThreshold :)

Vorbereitung eines Variationsmodells zum Vergleich.

prepare_direct_variation_model bereitet ein Variationsmodell zum Vergleich mit compare_variation_model oder compare_ext_variation_model vor. Hierzu muss das Modell mit Mode=*'direct'* mit create_variation_model erzeugt worden sein. Im Unterschied zu prepare_variation_model werden das ideale Bild und das zugehörige Variationsbild nicht durch train_variation_model berechnet, sondern in RefImage und VarImage direkt übergeben. Dies ist sinnvoll, wenn, wie bei create_variation_model beschrieben, das Variationsmodell aus einem Bild erzeugt werden soll. Das Variationsbild VarImage sollte typischerweise mit Kantenoperatoren wie sobel_amp, edges_image oder gray_range_rect erzeugt werden.

prepare_direct_variation_model berechnet aus dem idealen Bild RefImage und dem Variationsbild VarImage zwei interne Schwellenwertbilder und speichert sie im Variationsmodell ab. Diese Schwellenwertbilder werden in compare_variation_model oder compare_ext_variation_model verwendet, um den Vergleich des aktuellen Bildes mit dem Variationsmodell durchzuführen.

Zur Berechnung der Schwellenwertbilder werden zwei Schwellenwerte verwendet. Der Schwellenwert AbsThreshold gibt an, um wie viele Graustufen sich das Bild des aktuellen Objektes mindestens vom idealen Bild unterscheiden muss. Der Schwellenwert VarThreshold gibt einen Faktor für den Unterschied des aktuellen Bildes relativ zu der im Variationsbild gespeicherten Variation des Objektes an. VarThreshold legt mittels VarImage Regionen fest, in denen Unterschiede in den Grauwerten auftreten dürfen, die nicht als Fehler gezählt werden sollen. Dadurch werden Abweichungen in bestimmten Bereichen des zu prüfenden Objekts erlaubt. AbsThreshold und VarThreshold können jeweils einen oder zwei Werte enthalten. Falls zwei Werte angegeben werden, können unterschiedliche Schwellen für zu helle und zu dunkle Pixel festgelegt werden. Dabei bezieht sich der erste Wert auf die zu hellen Grauwerte und der zweite Wert auf die zu dunklen Grauwerte. Falls nur ein Wert angegeben wird, bezieht sich der Wert sowohl auf die zu hellen als auch auf die zu dunklen Grauwerte. Sei $i(x,y)$ das ideale Bild RefImage, $v(x,y)$ das Variationsbild VarImage, a_u = AbsThreshold[0], a_l = AbsThreshold[1], b_u = VarThreshold[0] und b_l = VarThreshold[1] (bzw. a_u = AbsThreshold, a_l = AbsThreshold, b_u = VarThreshold und b_l = VarThreshold). Dann werden die beiden Schwellenwertbilder $t_{u,l}$ wie folgt berechnet:

$$t_u(x,y) = i(x,y) + \max\{a_u, b_u v(x,y)\} \qquad t_l(x,y) = i(x,y) - \max\{a_l, b_l v(x,y)\} \ .$$

Wenn das aktuelle Bild $c(x,y)$ mit compare_variation_model mit dem Variationsmodell verglichen wird, wird die Ausgaberegion aus allen Punkten bestimmt, die sich stark vom Modell unterscheiden, d.h. für die gilt:

$$c(x,y) > t_u(x,y) \vee c(x,y) < t_l(x,y) \ .$$

In compare_ext_variation_model stehen erweiterte Vergleichsmodi zur Verfügung, mit denen zusätzlich nur zu helle Fehler, nur zu dunkle Fehler und zu helle und zu dunkle Fehler als separate Regionen zurückgeliefert werden können.

Nachdem die Schwellenwertbilder erzeugt worden sind, können sie mit get_thresh_images_variation_model ausgelesen werden.

Es ist zu beachten, dass RefImage und VarImage nicht als ideales Bild bzw. als Variationsbild im Modell abgespeichert werden, um Speicherplatz im Modell zu sparen.

Parameter

- ▷ **RefImage** (input_object) ..... singlechannelimage ⇝ *object :* byte / int2 / uint2
  Referenzbild des Objektes.
- ▷ **VarImage** (input_object) ..... singlechannelimage ⇝ *object :* byte / int2 / uint2
  Variationsbild des Objekts.
- ▷ **ModelID** (input_control) ..... variation_model ⇝ *handle*
  ID des Variationsmodells.
- ▷ **AbsThreshold** (input_control) ..... number(-array) ⇝ *real* / integer
  Absoluter Mindestschwellenwert für die Unterschiede bezüglich des Variationsmodells.
  Defaultwert: 10
  Wertevorschläge: AbsThreshold ∈ {0, 5, 10, 15, 20, 30, 40, 50}
  Restriktion: AbsThreshold >= 0
- ▷ **VarThreshold** (input_control) ..... number(-array) ⇝ *real* / integer
  Schwellenwert für die Unterschiede bezüglich der Variation des Variationsmodells.
  Defaultwert: 2
  Wertevorschläge: VarThreshold ∈ {1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5}
  Restriktion: VarThreshold >= 0

Beispiel

```
read_image (Image, 'model')
sobel_amp (Image, VarImage, 'sum_abs', 3)
get_image_pointer1 (Image, Pointer, Type, Width, Height)
create_variation_model (Width, Height, Type, 'direct', ModelID)
prepare_direct_variation_model (Image, VarImage, ModelID, 20, 1)
write_variation_model (ModelID, 'model.var')
```

Ergebnis

`prepare_direct_variation_model` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`sobel_amp`, `edges_image`, `gray_range_rect`

Nachfolger

`compare_variation_model`, `compare_ext_variation_model`, `get_thresh_images_variation_model`, `write_variation_model`

Alternativen

`prepare_variation_model`

Siehe auch

`create_variation_model`

Modul

Matching

prepare_variation_model (: : ModelID, AbsThreshold, VarThreshold :)

Vorbereitung eines Variationsmodells zum Vergleich.

`prepare_variation_model` bereitet ein Variationsmodell zum Vergleich mit `compare_variation_model` oder `compare_ext_variation_model` vor. Hierzu werden aus dem mit `train_variation_model` trainierten idealen Bild und dem zugehörigen Variationsbild zwei interne Schwellenwertbilder berechnet und im Variationsmodell abgespeichert. Diese Schwellenwertbilder werden in `compare_variation_model` oder `compare_ext_variation_model` verwendet, um den Vergleich des aktuellen Bildes mit dem Variationsmodell zu beschleunigen.

Zur Berechnung der Schwellenwertbilder werden zwei Schwellenwerte verwendet. Der Schwellenwert `AbsThreshold` gibt an, um wie viele Graustufen sich das Bild des aktuellen Objektes mindestens vom idealen Bild unterscheiden muss. Der Schwellenwert `VarThreshold` gibt einen Faktor für den Unterschied des aktuellen Bildes relativ zu der im Variationsbild gespeicherten Variation des Objektes an. `AbsThreshold` und `VarThreshold` können jeweils einen oder zwei Werte enthalten. Falls zwei Werte angegeben werden, können unterschiedliche Schwellen für zu helle und zu dunkle Pixel festgelegt werden. Dabei bezieht sich der erste Wert auf die zu hellen Grauwerte und der zweite Wert auf die zu dunklen Grauwerte. Falls nur ein Wert angegeben wird, bezieht sich der Wert sowohl auf die zu hellen als auch auf die zu dunklen Grauwerte. Sei $i(x,y)$ das ideale Bild, $v(x,y)$ das Variationsbild, a_u = `AbsThreshold`[0], a_l = `AbsThreshold`[1], b_u = `VarThreshold`[0] und b_l = `VarThreshold`[1] (bzw. a_u = `AbsThreshold`, a_l = `AbsThreshold`, b_u = `VarThreshold` und b_l = `VarThreshold`). Dann werden die beiden Schwellenwertbilder $t_{u,l}$ wie folgt berechnet:

$$t_u(x,y) = i(x,y) + \max\{a_u, b_u v(x,y)\} \qquad t_l(x,y) = i(x,y) - \max\{a_l, b_l v(x,y)\} \ .$$

Wenn das aktuelle Bild $c(x,y)$ mit `compare_variation_model` mit dem Variationsmodell verglichen wird, wird die Ausgaberegion aus allen Punkten bestimmt, die sich stark vom Modell unterscheiden, d.h. für die gilt:

$$c(x,y) > t_u(x,y) \vee c(x,y) < t_l(x,y) \ .$$

In compare_ext_variation_model stehen erweiterte Vergleichsmodi zur Verfügung, mit denen zusätzlich nur zu helle Fehler, nur zu dunkle Fehler und zu helle und zu dunkle Fehler als separate Regionen zurückgeliefert werden können.

Nachdem die Schwellenwertbilder erzeugt worden sind, können sie mit get_thresh_images_variation_model ausgelesen werden. Außerdem können die Trainingsdaten mit clear_train_data_variation_model gelöscht werden, um Speicherplatz zu sparen.

Parameter

▷ **ModelID** (input_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

▷ **AbsThreshold** (input_control) . . . . . number(-array) ⇝ *real* / integer
Absoluter Mindestschwellenwert für die Unterschiede bezüglich des Variationsmodells.
Defaultwert: 10
Wertevorschläge: AbsThreshold ∈ {0, 5, 10, 15, 20, 30, 40, 50}
Restriktion: AbsThreshold >= 0

▷ **VarThreshold** (input_control) . . . . . number(-array) ⇝ *real* / integer
Schwellenwert für die Unterschiede bezüglich der Variation des Variationsmodells.
Defaultwert: 2
Wertevorschläge: VarThreshold ∈ {1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5}
Restriktion: VarThreshold >= 0

Ergebnis

prepare_variation_model liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

train_variation_model

Nachfolger

compare_variation_model, compare_ext_variation_model, get_thresh_images_variation_model, clear_train_data_variation_model, write_variation_model

Alternativen

prepare_direct_variation_model

Siehe auch

create_variation_model

Modul

Matching

read_variation_model (: : FileName : ModelID)

Lesen eines Variationsmodells von Datei.

read_variation_model liest ein Variationsmodell, das mit write_variation_model geschrieben wurde, aus der Datei FileName ein. Die Dateiendung für den ein Variationsmodell ist in HALCON 'vam'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.vam`

▷ **ModelID** (output_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

Ergebnis

Ist der Dateiname korrekt, dann liefert `read_variation_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`compare_variation_model`, `compare_ext_variation_model`

Siehe auch

`write_variation_model`

Modul

Matching

serialize_variation_model (: : ModelID : SerializedItemHandle)

Serialisieren eines Variationsmodells.

`serialize_variation_model` serialisiert die Daten eines Variationsmodells (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_variation_model` in eine Datei geschrieben werden. Das Variationsmodell wird in dem Handle `ModelID` übergeben. Das serialisierte Variationsmodell wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_variation_model` wieder deserialisiert werden.

Parameter

▷ **ModelID** (input_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_variation_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_variation_model`

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_variation_model

Siehe auch

read_variation_model, write_variation_model

Modul

Matching

train_variation_model (Images : : ModelID :)

Trainieren eines Variationsmodells.

train_variation_model trainiert das durch ModelID angegebene Variationsmodell mit einem oder mehreren Bildern, die in Images übergeben werden.

Wie bei create_variation_model beschrieben, kann ein Variationsmodell, das mit dem Modus *'standard'* erzeugt worden ist, iterativ trainiert werden kann, d.h. sobald ein oder mehrere Bilder eines guten Objektes vorliegen, können diese mit train_variation_model trainiert werden. Das ideale Bild des Objektes wird in diesem Modus aus dem Mittelwert aller bisherigen Trainingsbilder und den Bildern, die in Images übergeben werden, berechnet. Das zugehörige Variationsbild wird aus der Standardabweichung der Trainingsbilder und der Bilder, die in Images übergeben werden, berechnet.

Falls das Variationsmodell mit dem Modus *'robust'* erzeugt worden ist, kann das Modell nicht iterativ trainiert werden, d.h. alle Trainingsbilder müssen mit concat_obj aufgesammelt werden und in einem Aufruf von train_variation_model trainiert werden. Falls zuvor schon Bilder trainiert worden sind, geht die Trainingsinformation verloren. Das ideale Bild des Objektes wird in diesem Modus aus dem Median aller in Images übergebenen Trainingsbilder berechnet. Das zugehörige Variationsbild wird aus dem Median der absoluten Abweichungen der Trainingsbilder vom Median berechnet und geeignet skaliert.

Achtung

Es können maximal 65535 Trainingsbilder trainiert werden.

Parameter

- ▷ **Images** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2
  Bilder des zu trainierenden Objekts.
- ▷ **ModelID** (input_control) ..... variation_model ⇝ *handle*
  ID des Variationsmodells.

Beispiel

```
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, \
                   'default', -1, 'default', 'model.seq', 'default', \
                   -1, -1, AcqHandle)
grab_image (Image, AcqHandle)
get_image_pointer1 (Image, Pointer, Type, Width, Height)
dev_display (Image)
draw_region (Region, WindowHandle)
reduce_domain (Image, Region, ImageReduced)
area_center (Region, Area, RowRef, ColumnRef)
create_shape_model (ImageReduced, 4, 0, rad(360), rad(1), 'none', \
                    'use_polarity', 40, 10, TemplateID)
create_variation_model (Width, Height, Type, 'standard', ModelID)
for K := 1 to 100 by 1
    grab_image (Image, AcqHandle)
    find_shape_model (Image, TemplateID, 0, rad(360), 0.5, 1, 0.5, \
                      'true', 4, 0.9, Row, Column, Angle, Score)
    if (|Score| == 1)
        vector_angle_to_rigid (Row, Column, Angle, RowRef, \
                               ColumnRef, 0, HomMat2D)
        affine_trans_image (Image, ImageTrans, HomMat2D, 'constant', \
```

```
                              'false')
        train_variation_model (ImageTrans, ModelID)
    endif
endfor
prepare_variation_model (ModelID, 10, 4)
write_region (Region, 'model.reg')
write_shape_model (TemplateID, 'model.shm')
write_variation_model (ModelID, 'model.var')
close_framegrabber (AcqHandle)
```

Ergebnis

`train_variation_model` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_variation_model`, `find_shape_model`, `affine_trans_image`, `concat_obj`

Nachfolger

`prepare_variation_model`

Siehe auch

`prepare_variation_model`, `compare_variation_model`, `compare_ext_variation_model`, `clear_variation_model`

Modul

Matching

write_variation_model (: : ModelID, FileName :)

Schreiben eines Variationsmodells auf Datei.

`write_variation_model` schreibt ein Variationsmodell in die Datei `FileName`. Das Modell kann mit `read_variation_model` wieder eingelesen werden. Die Dateiendung für den ein Variationsmodell ist in HALCON 'vam'.

Parameter

▷ **ModelID** (input_control) . . . . . variation_model ⇝ *handle*
ID des Variationsmodells.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.vam`

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert `write_variation_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_variation_model`

Siehe auch

`read_variation_model`

Modul

Matching

Kapitel 14

Kalibrierung

Dieses Kapitel enthält Informationen zur Kalibrierung von Kameras.

Allgemein

Um maximale Messgenauigkeit für einen Kameraaufbau zu erzielen muss dieser entsprechend kalibriert werden. Daher wird ein Kameramodell berechnet, welches die Projektion eines 3D-Weltpunktes auf ein (Sub-)Pixel des Bildes beschreibt.

In HALCON wird eine Vielzahl an Operatoren zur Verfügung gestellt um diverse Aufgabenstellungen aus dem Bereich der Kalibrierung bewältigen zu können, darunter

- die Beschreibung und Erkennung von Kalibrierobjekten (Kalibrierung / Kalibrierkörper),
- die Projektion von Punkten aus 3D-Szenen auf die Bildebene und umgekehrt (Kalibrierung / Projektion, Kalibrierung / Inverse Projektion),
- die Kompensation perspektivischer und radialer Verzerrungen (Kalibrierung / Rektifizierung),
- die Handhabung der Kameraparameter Kalibrierung / Kameraparameter),
- die Durchführung einer Selbstkalibrierung (Kalibrierung / Selbst-Kalibrierung), und
- die Kalibrierung verschiedener Kameraaufbauten, bestehend aus
  - einer Kamera (Kalibrierung / Monokular),
  - mehrerer Kameras (Kalibrierung / Binokular, Kalibrierung / Mehrbild), oder
  - einer Kombination aus Kamera und Roboter (Kalibrierung / Hand-Auge).

Dieses Kapitel gibt Empfehlungen bezüglich des grundlegenden Konzepts zur Berechnung der internen und externen Parameter einer Kamera. Die folgenden Paragraphen beschreiben, wie eine Kamera erfolgreich kalibriert wird. Insbesondere werden darin

- das benötigte Kalibrierobjekt,
- die einzelnen Schritte der Kamerakalibrierung, darunter
  - wie die Vorbereitung der Eingabedaten erfolgt,
  - wie die Kalibrierung mittels `calibrate_cameras` durchgeführt wird und
  - wie der Erfolg der Kalibrierung überprüft werden kann
- die Kameraparameter,
- zusätzliche Informationen über den Kalibrierprozess, darunter
  - wie eine geeignete Kalibrierplatte bezogen werden kann,

  - wie eine Reihe geeigneter Bilder aufgenommen werden kann, und
  - welches Verzeichnungsmodell verwendet werden sollte,

- die zur Verfügung stehenden 3D-Kameramodelle und wie 3D-Punkte in das Bildkoordinatensystem transformiert werden, und
- Beschränkungen bezogen auf einzelne Kameratypen

beschrieben.

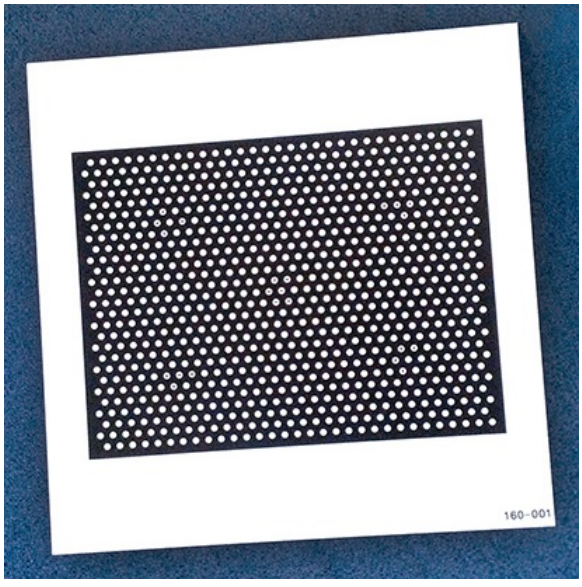
Kalibrierobjekt

Für eine erfolgreiche Kalibrierung wird mindestens ein Kalibrierobjekt mit genau bekannten metrischen Abmessungen benötigt, z.B. eine HALCON-Kalibrierplatte. Für die Kalibrierung nehmen Sie eine Serie von Bildern des Kalibrierobjektes in unterschiedlichen Orientierungen auf. Der Erfolg der Kalibrierung hängt in hohem Maße von der Qualität der Kalibrierbilder ab. Deshalb wird bei der Aufnahme der Kalibrierbilder hohe Sorgfalt empfohlen. Siehe den Abschnitt „Wie nehme ich geeignete Bilder auf?"für weitere Informationen.

Eine Kalibrierplatte ist mit einer Vielzahl von Kalibriermarken bedeckt, welche auf den Kalibrierbildern extrahiert werden um ihre Koordinaten zu bestimmen. Der Ursprung des lokalen Koordinatensystems der Platte liegt im Schwerpunkt der Marken. Die Orientierung der Platte muss genau bestimmbar sein, daher ist auch ein Suchmuster Teil des Aufdrucks.

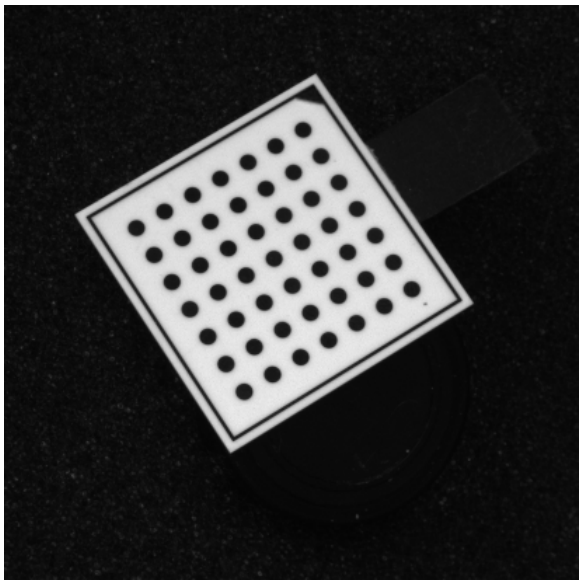
Ihr lokaler Vertrieb bietet zur Kamerakalibrierung zwei verschiedene HALCON-Kalibrierkörper an:

Kalibrierplatte mit hexagonal angeordneten Marken: Als Suchmuster weisen gewisse Marken kreisförmige Löcher auf (siehe `create_caltab`). Ein Suchmuster muss sichtbar sein um die Kalibrierplatte zu lokalisieren, um festzustellen ob sie invertiert ist müssen mindestens zwei Suchmuster zu sehen sein. Folglich muss die Platte nicht vollständig auf dem Bild sichtbar sein. Der Ursprung des Koordinatensystems liegt in der Mitte der zentralen Marke des ersten Suchmusters. Die z-Achse des Koordinatensystems zeigt in den Kalibrierkörper hinein, die x-Achse (in z-Richtung blickend) nach rechts und die y-Achse nach unten. Wird zur Kalibrierung `camera_calibration` statt `calibrate_cameras` genutzt, ist diese Platte nicht verwendbar.



HALCON Kalibrierplatte mit hexagonal angeordneten Marken.

Kalibrierplatte mit rechteckig angeordneten Marken: Der schwarze Rahmen der Kalibrierplatte und die in der linken oberen Ecke befindliche dreieckige Orientierungsmarkierung sind das Suchmuster für Kalibrierplatten mit rechteckig angeordneten Marken (siehe `gen_caltab`). Entsprechend muss die Kalibrierplatte vollständig im Bild sichtbar sein. Der Ursprung des Koordinatensystems liegt im Mittelpunkt der Kalibrierkörperoberfläche. Die z-Achse des Koordinatensystems zeigt in den Kalibrierkörper hinein, die x-Achse (in z-Richtung blickend) nach rechts und die y-Achse nach unten.



HALCON Kalibrierplatte mit rechteckig angeordneten Marken.

Vorbereitung der Eingabekalibrierdaten

Vor dem Aufruf des Operators `calibrate_cameras` sollten die benötigten Daten im Kalibrierdatenmodell gesammelt werden. Dabei müssen diese Schritte befolgt werden:

1. **Erstellen des Kalibrierdatenmodells** mit dem Operator `create_calib_data`, wobei die Anzahl der Kameras und die Anzahl der Kalibrierkörper im Modell festgelegt werden.
2. **Definieren des Kameratyps und der internen Kameraparameter** für alle Kameras mit dem Operator `set_calib_data_cam_param`.
3. **Definieren der Kalibrierkörperbeschreibung** für alle Kalibrierkörper mit dem Operator `set_calib_data_calib_object`.
4. **Sammeln von Beobachtungen** mit den Operatoren `find_calib_object` oder `set_calib_data_observ_points`, d.h. die Bildkoordinaten der extrahierten Kalibrierkörpermarker und eine grob geschätzte Pose des Kalibrierkörpers relativ zur beobachtenden Kamera.
5. **Kalibrierprozess konfigurieren**, z.B. bestimmte interne oder externe Kameraparameter von der Optimierung ausschließen. Diese Parameter können mit dem Operator `set_calib_data` gesetzt werden. Wenn bestimmte Kameraparameter genau bekannt sind, z.B. die Zellgrößen des Kamerasensors von Kamera 0, brauchen sie nicht kalibriert zu werden. Das kann mit folgendem Aufruf konfiguriert werden:

```
set_calib_data (CalibDataID, 'camera' , 0,
                'excluded_settings', ['sx','sy']).
```

Die tatsächliche Kalibrierung durchführen und die Ergebnisse abfragen

Mit Hilfe der Informationen die im Kalibrierdatenmodell gespeichert wurden, kann die tatsächliche Kalibrierung mit dem Operator `calibrate_cameras` durchgeführt werden. Dabei wird das Eingabemodell verändert indem die initialen internen Kameraparameter optimiert und weitere Daten wie externe Parameter und Standardabweichungen hinzugefügt werden. Außerdem werden die Standardabweichungen und Kovarianzen der kalibrierten internen Parameter und die Wurzel des mittleren quadratischen Rückprojektionsfehlers (RMSE) berechnet, um den Erfolg der Kalibrierung bewerten zu können.

Die Ergebnisse können dann mit `get_calib_data` abgefragt werden.

Ergebnisse der Kalibrierung überprüfen

Nach erfolgreicher Kalibrierung gibt der Operator `calibrate_cameras` die Wurzel des mittleren quadratischen Rückprojektionsfehlers (RMSE) der Optimierung im Parameter `Error` (in Pixel) zurück. Dieser Fehler gibt

einen allgemeinen Hinweis darauf, ob die Optimierung erfolgreich war, da er die durchschnittlichen Abweichungen (in Pixeln) zwischen den zurück projizierten Kalibrierpunkten und den errechneten Bildkoordinaten angibt.

Bei der Kalibrierung von nur einer einzelnen Kamera ist ein Fehler in der Größenordnung von 0.1 Pixel (der typische Detektionsfehler beim Extrahieren der Koordinaten der projizierten Kalibrierkörpermarken) ein Hinweis dafür, dass die Optimierung gut zu den Beobachtungsdaten passt. Wenn der Fehler deutlich über 0.1 Pixel liegt, wurde die Kalibrierung nicht gut ausgeführt. Die Gründe hierfür können z.B. eine schlechte Bildqualität, eine unzureichende Anzahl an Kalibrierbildern oder auch eine fehlerhafte Kalibrierplatte sein.

Für Informationen zur Überprüfung der Kalibrierung eines Mehrbild-Kameraaufbaus siehe die Kapitelreferenz Kalibrierung / Mehrbild.

Kameraparameter

Die Kameraparameter werden unterschieden nach internen und externen Kameraparametern.

Interne Kameraparameter: Die internen Kameraparameter beschreiben die Beschaffenheit der verwendeten Kamera, insbesondere die interne Sensorgröße und die Abbildungseigenschaften der verwendeten Kombination von Objektiv, Kamera und Framegrabber. Unten sehen Sie eine Übersicht über alle verfügbaren Kameratypen und deren jeweilige Parameter. In der Liste bezieht sich „projektive Kameras“ auf die Eigenschaft, dass das Objektiv eine perspektivische Projektion auf der Objektseite des Objektivs durchführt, während sich „telezentrische Kameras“ auf die Eigenschaft bezieht, dass das Objektiv eine telezentrische Projektion auf der Objektseite des Objektivs durchführt.

Kameras mit Flächensensor haben 9 bis 16 interne Parameter abhängig vom Kameratyp.

Aus weiter unten genannten Gründen werden die mit einem * Sternchen gekennzeichneten Parameter nicht vom Algorithmus geschätzt, sondern bleiben fest.

Flächenkameras mit normalen Objektiven

Projektive Flächenkameras mit normalen Objektiven

'area_scan_division' ['area_scan_division', Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_polynomial' ['area_scan_polynomial', Focus, K1, K2, K3, P1, P2, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

Telezentrische Flächenkameras mit normalen Objektiven

'area_scan_telecentric_division' ['area_scan_telecentric_division', Magnification, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_telecentric_polynomial' ['area_scan_telecentric_polynomial', Magnification, K1, K2, K3, P1, P2, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

Flächenkameras mit Tiltobjektiven

Projektive Flächenkameras mit Tiltobjektiven

'area_scan_tilt_division' ['area_scan_tilt_division', Focus, Kappa, ImagePlaneDist, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_tilt_polynomial' ['area_scan_tilt_polynomial', Focus, K1, K2, K3, P1, P2, ImagePlaneDist, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_tilt_image_side_telecentric_division' ['area_scan_tilt_image_side_telecentric_division', Focus, Kappa, Tilt, Rot, Sx, Sy*, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_tilt_image_side_telecentric_polynomial' ['area_scan_tilt_image_side_telecentric_polynomial', Focus, K1, K2, K3, P1, P2, Tilt, Rot, Sx, Sy*, Cx, Cy, ImageWidth, ImageHeight]*

Telezentrische Flächenkameras mit Tiltobjektiven

'area_scan_tilt_bilateral_telecentric_division' ['area_scan_tilt_bilateral_telecentric_division', Magnification, Kappa, Tilt, Rot, Sx, Sy*, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_tilt_bilateral_telecentric_polynomial' ['area_scan_tilt_bilateral_telecentric_polynomial', Magnification, K1, K2, K3, P1, P2, Tilt, Rot, Sx, Sy*, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_tilt_object_side_telecentric_division' ['area_scan_tilt_object_side_telecentric_division', Magnification, Kappa, ImagePlaneDist, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_tilt_object_side_telecentric_polynomial' ['area_scan_tilt_object_side_telecentric_polynomial', Magnification, K1, K2, K3, P1, P2, ImagePlaneDist, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

Flächenkameras mit hyperzentrischen Objektiven

Projektive Flächenkameras mit hyperzentrischen Objektiven

'area_scan_hypercentric_division' ['area_scan_hypercentric_division', Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

'area_scan_hypercentric_polynomial' ['area_scan_hypercentric_polynomial', Focus, K1, K2, K3, P1, P2, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]*

Beschreibung der internen Kameraparameter von Kameras mit Flächensensor:

CameraType: Typ der Kamera, wie oben aufgelistet.

Focus: Brennweite des Objektivs (nur für Kameras mit Objektiven, die auf der Objektseite eine perspektivische Projektion durchführen).
Startwert ist gleich der nominalen Brennweite des verwendeten Objektivs, also beispielsweise 0.008 m.

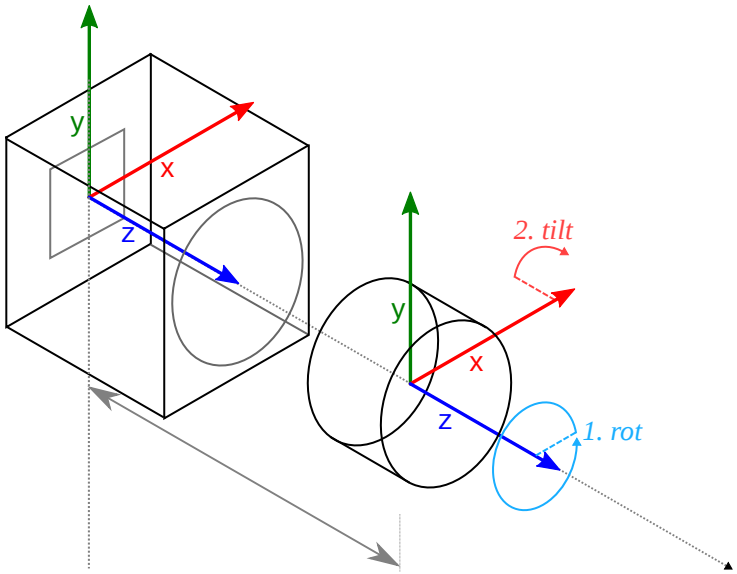
Magnification: Vergrößerungsmaßstab des Objektivs (nur für Kameras mit Objektiven, die auf der Objektseite eine telezentrische Projektion durchführen).
Startwert ist der nominale Vergrößerungsmaßstab des verwendeten telezentrischen Objektivs (Bildgröße geteilt durch Objektgröße), beispielsweise 0.2.

Kappa (κ): Verzeichnungskoeffizient zur Modellierung der radialen Verzeichnungen mit dem Divisionsmodell.
Startwert: 0.0 m^{-2}.

K1, K2, K3, P1, P2: Verzeichnungskoeffizienten zur Modellierung der radialen (K_1, K_2, K_3) und tangentialen (P_1, P_2) Verzeichnung mit dem Polynommodell.
Startwert für alle fünf Koeffizienten: 0.0

ImagePlaneDist: Abstand der Austrittspupille zur Bildebene. Die Austrittspupille ist das (virtuelle) Bild der Aperturblende (typischerweise die Irisblende), betrachtet von der Bildseite des Objektivs. Typische Werte liegen in der Größenordnung von wenigen Zentimetern bis zu sehr großen Werten, falls das Objektiv fast bildseitig telezentrisch ist.

Tilt, Rot: Der Tiltwinkel $tilt$ $(0° \leq tilt < 90°)$, um den die optische Achse im Vergleich zur Normalen der Bildebene verkippt ist (entspricht einer Drehung um die x-Achse), und der Rotationswinkel rot $(0° \leq rot < 360°)$, der beschreibt, in welche Richtung die optische Achse gekippt ist. Bei einer Rotation von $rot = 0°$ wird die optische Achse vertikal um den Tiltwinkel nach unten (bezüglich des Kameragehäuses) gekippt, $rot = 90°$ entspricht einer Verkippung nach links (Blickrichtung entlang der z-Achse), $rot = 180°$ nach oben, und bei $rot = 270°$ wird die optische Achse um den Winkel $tilt$ nach rechts gekippt.
Diese Parameter werden nur benötigt, falls ein Tiltobjektiv verwendet wird.



Der Tilt eines Objektivs wird durch die Parameter rot , $tilt$ und `ImagePlaneDist` beschrieben. rot beschreibt die Orientierung der Tiltachse relativ zur x-Achse des Sensors und muss zuerst angebracht werden. $tilt$ beschreibt den tatsächlichen Tiltwinkel des Objektivs. `ImagePlaneDist` beschreibt den Abstand der Austrittspupille zur Bildebene.

Diese Winkel sind typischerweise aus den Überlegungen, die zur Verwendung des Tiltobjektivs geführt haben, bekannt oder können von dem Mechanismus, der zur Verkippung des Objektivs verwendet wird, abgelesen werden.

Sx, Sy: Skalierungsfaktoren. Entsprechen dem horizontalen und vertikalen Abstand zweier benachbarter Zellen auf dem Sensor. Da in der Regel das Bildsignal zeilensynchron ausgelesen wird, ist S_y durch die Größe des Sensors exakt festgelegt und muss daher nicht durch die Kalibrierung bestimmt werden.
Die Startwerte für den Abstand zweier benachbarter Zellen hängen von der Größe des Chips der verwendeten Kamera ab und können den technischen Daten der Kamera entnommen werden. Vorsicht: Bei Unterabtastung des Bildes erhöhen sich diese Werte!
Da **projektive Flächenkameras** durch das Modell einer Lochkamera mit Flächensensor beschrieben werden, ist es unmöglich, `Focus`, S_x und S_y gleichzeitig zu bestimmen. Daher wird S_y nicht optimiert, sondern bleibt fest.
Für **telezentrische Objektive** ist es unmöglich, `Magnification`, S_x und S_y gleichzeitig zu bestimmen. Daher wird S_y nicht optimiert, sondern bleibt fest.
Bei **bildseitig telezentrischen Tiltobjektiven** ist es unmöglich, `Focus`, S_x, S_y und die Verkippungsparameter $tilt$ und rot gleichzeitig zu bestimmen. Daher wird zusätzlich S_x nicht optimiert, sondern bleibt fest.
Bei **beidseitig telezentrischen Tiltobjektiven** ist es unmöglich, `Magnification`, S_x, S_y und die Verkippungsparameter $tilt$ und rot gleichzeitig zu bestimmen. Daher wird zusätzlich S_x nicht optimiert, sondern bleibt fest.

Cx, Cy: Spalten- (C_x) und Zeilenkoordinate (C_y) des Kamerahauptpunktes (Zentrum der radialen Verzeichnung).
Startwerte für die Koordinaten des Kamerahauptpunktes sind die halbe Bildbreite bzw. -höhe. Hinweis: Bei Unterabtastung des Bildes verringern sich die Werte entsprechend!

ImageWidth, ImageHeight: Breite und Höhe des abgetasteten Bildes. Vorsicht: Bei Unterabtastung des Bildes verringern sich diese Werte!

Zeilenkameras haben 12 oder 16 interne Parameter abhängig vom Kameratyp.

Aus weiter unten genannten Gründen werden die mit einem * Sternchen gekennzeichneten Parameter nicht vom Algorithmus geschätzt, sondern bleiben fest.

Flächenkameras mit normalen Objektiven

Projektive Zeilenkameras mit normalen Objektiven

'line_scan_division' ['line_scan_division', Focus, Kappa, Sx, Sy*, Cx, Cy, ImageWidth, ImageHeight, Vx, Vy, Vz]*

'line_scan_polynomial' ['line_scan_polynomial', Focus, K1, K2, K3, P1, P2, Sx, Sy*, Cx, Cy, ImageWidth, ImageHeight, Vx, Vy, Vz]*

Telezentrische Zeilenkameras mit normalen Objektiven

'line_scan_telecentric_division' ['line_scan_telecentric_division', Magnification, Kappa, Sx, Sy*, Cx, Cy, ImageWidth, ImageHeight, Vx, Vy, Vz*]*

'line_scan_telecentric_polynomial' ['line_scan_telecentric_polynomail', Magnification, K1, K2, K3, P1, P2, Sx, Sy*, Cx, Cy, ImageWidth, ImageHeight, Vx, Vy, Vz*]*

Beschreibung der internen Kameraparameter von Zeilenkameras:

CameraType: Typ der Kamera, wie oben aufgelistet.

Focus: Brennweite des Objektivs (nur für Kameras mit Objektiven, die auf der Objektseite eine perspektivische Projektion durchführen).
Startwert ist gleich der nominalen Brennweite des verwendeten Objektivs, also beispielsweise 0.008 m.

Magnification: Vergrößerungsmaßstab des Objektivs (nur für Kameras mit Objektiven, die auf der Objektseite eine telezentrische Projektion durchführen).
Startwert ist der nominale Vergrößerungsmaßstab des verwendeten telezentrischen Objektivs (Bildgröße geteilt durch Objektgröße), beispielsweise 0.2.

Kappa (κ)**:** Verzeichnungskoeffizient zur Modellierung der radialen Verzeichnung mit dem Divisionsmodell.
Startwert: 0.0 m^{-2}.

K1, K2, K3, P1, P2: Verzeichnungskoeffizienten zur Modellierung der radialen (K_1, K_2, K_3) und tangentialen (P_1, P_2) Verzeichnung mit dem Polynommodell.
Startwert für alle fünf Koeffizienten: 0.0

Sx: Skalierungsfaktor. Entspricht dem horizontalen Abstand zweier benachbarter Zellen auf der Sensorzeile. Es ist zu beachten, dass `Focus` bzw. `Magnification` und S_x nicht gleichzeitig bestimmt werden können. Deshalb wird S_x in der Kalibrierung festgehalten. Der Startwert für den Abstand zweier benachbarter Zellen kann den technischen Daten der Kamera entnommen werden. Vorsicht: Bei Unterabtastung des Bildes erhöht sich dieser Wert!

Sy: Skalierungsfaktor, der im Rahmen der Kalibrierung nur in der Form $p_v = -S_y \cdot C_y$ auftritt. Daher können S_y und C_y nicht gleichzeitig bestimmt werden. Deshalb wird S_y in der Kalibrierung fest gehalten. p_v beschreibt den Abstand des Kamerahauptpunktes von der Sensorzeile in Metern. Der Startwert für die Größe einer Zelle senkrecht zur Richtung der Sensorzeile kann ebenfalls den technischen Daten der Kamera entnommen werden. Vorsicht: Bei Unterabtastung des Bildes erhöht sich der Wert von S_y!

Cx: Spaltenkoordinate des Kamerahauptpunktes (Zentrum der radialen Verzeichnung).
Der Startwert für die x-Koordinate des Kamerahauptpunktes ist die halbe Bildbreite, d.h. die halbe Anzahl von Pixeln des Zeilensensors. Hinweis: Bei Unterabtastung des Bildes verringert sich der Wert für C_x entsprechend!

Cy: Abstand des Kamerahauptpunktes (Zentrum der radialen Verzeichnung) von der Sensorzeile in Bildzeilen. Der Startwert für die y-Koordinate des Kamerahauptpunktes ist normalerweise 0.

ImageWidth, ImageHeight: Breite und Höhe des abgetasteten Bildes. Vorsicht: Bei Unterabtastung des Bildes verringern sich diese Werte!

Vx, Vy, Vz: Die x-, y- und z-Komponenten des Bewegungsvektors. Die Startwerte für die x-, y- und z-Komponente des Bewegungsvektors hängen vom Aufbau des gesamten Aufnahmesystems ab. Blickt die Kamera beispielsweise senkrecht auf ein Fließband, und ist dabei so um ihre optische Achse gedreht, dass die Sensorzeile senkrecht zur Bewegungsrichtung des Fließbandes steht, dass also die y-Achse des Kamerakoordinatensystems parallel zur Bewegungsrichtung des Fließbandes ausgerichtet ist, so sind die Startwerte $V_x = V_z = 0$. Der Startwert für V_y lässt sich dann aus der Aufnahme eines Objektes dessen Größe bekannt ist (z.B. Kalibrierplatte, Lineal) folgendermaßen bestimmen:

$$V_y = \frac{l[m]}{l[row]}$$

Mit:

$$\begin{aligned} l[m] &= \text{Länge des Objektes in Bewegungsrichtung in Objektkoordinaten [Meter]} \\ l[row] &= \text{Länge des Objektes in Bewegungsrichtung in Bildkoordinaten [Bildzeilen]} \end{aligned}$$

Wird die Kamera gegenüber diesem Aufbau z.B. um 30 Grad um die optische Achse, also die z-Achse des Kamerakoordinatensystems gedreht, so sind die oben bestimmten Startwerte folgendermaßen zu verändern:

$$\begin{aligned} V_x^z &= \sin(30^\circ) V_y \\ V_y^z &= \cos(30^\circ) V_y \\ V_z^z &= V_z = 0 \end{aligned}$$

Wenn die Kamera gegenüber dem ursprünglichen Aufbau z.B. um -20 Grad um die x-Achse des Kamerakoordinatensystems gedreht wird, so ergeben sich folgende Startwerte:

$$\begin{aligned} V_x^x &= V_x = 0 \\ V_y^x &= \cos(-20^\circ) V_y \\ V_z^x &= \sin(-20^\circ) V_y \end{aligned}$$

Die Qualität der Startwerte für V_x, V_y und V_z ist entscheidend für den Erfolg der gesamten Kalibrierung. Werden zu schlechte Startwerte verwendet, kann dies zum Abbruch der Kamerakalibrierung führen.
Es ist zu beachten, dass für telezentrische Zeilenkameras der Wert von V_z keinen Einfluss auf die Bildposition von 3D-Punkten hat und daher nicht bestimmt werden kann. Deswegen wird V_z für telezentrische Zeilenkameras nicht optimiert und auf dem Startwert belassen. Der Startwert von V_z sollte daher auf *0* gesetzt werden. Für Aufbauten mit mehreren telezentrischen Zeilenkameras, die einen gemeinsamen Bewegungsvektor besitzen (für eine genauere Erklärung siehe Kalibrierung / Mehrbild), kann V_z hingegen basierend auf der Pose der Kameras bestimmt werden. Daher wird in diesem Fall V_z optimiert.

Restriktionen für interne Kameraparameter Zu beachten ist, dass die Bezeichnung *Focus* im Allgemeinen nicht der realen Brennweite entspricht, wenn die Objektweite nicht unendlich ist. Der Einfachheit halber wird hier aber nicht zwischen Brennweite und Bildweite unterschieden.

Prüfen Sie für alle Operatoren, die Kameraparameter als Eingabe verwenden, ob die Werte der jeweiligen Kameraparameter folgende Restriktionen erfüllen:

$$\begin{aligned} S_x &> 0 \\ S_y &\geq 0 \\ Focus &> 0 \\ Magnification &> 0 \\ ImageWidth &> 0 \\ ImageHeight &> 0 \\ ImagePlaneDist &> 0 \\ 0 \leq tilt &< 90 \\ 0 \leq rot &< 360 \\ V_x^2 + V_y^2 + V_z^2 &\neq 0 \end{aligned}$$

Für manche Operatoren gibt es kleine Unterschiede in den Restriktionen. Im Speziellen für Operatoren, die Lochkameras mit Zeilensensor nicht unterstützen, gilt folgende Restriktion:

$$S_y > 0$$

Externe Kameraparameter: Die folgenden 6 Parameter beschreiben die 3D-Lage (Pose) des Weltkoordinatensystems relativ zum Kamerakoordinatensystem. Die x- und y-Achse des Kamerakoordinatensystems sind parallel zur Zeilen- und Spaltenachse der Abbildung, während die z-Achse rechtwinklig auf der Bildebene steht. Bei Kameras mit Zeilensensor bezieht sich die Pose des Weltkoordinatensystems auf das Kamerakoordinatensystem der ersten Bildzeile.

TransX: Translation entlang der x-Achse des Kamerakoordinatensystems.

TransY: Translation entlang der y-Achse des Kamerakoordinatensystems.

TransZ: Translation entlang der z-Achse des Kamerakoordinatensystems.

RotX: Rotationswinkel um die x-Achse des Kamerakoordinatensystems.

RotY: Rotationswinkel um die y-Achse des Kamerakoordinatensystems.

RotZ: Rotationswinkel um die z-Achse des Kamerakoordinatensystems.

Zusätzlich ist in dem Tupel, das die Pose beschreibt, als siebtes Element der Darstellungstyp enthalten. Dieser codiert die Kombination der Parameterwerte `OrderOfTransform`, `OrderOfRotation` und `ViewOfTransform`. Für detaillierte Informationen zu Posen siehe `create_pose`.

Bei Benutzung einer HALCON-Kalibrierplatte definiert deren Koordinatensystem das Weltkoordinatensystem. Siehe dazu den obigen Abschnitt „Kalibrierobjekt“.

Startwerte für alle Parameter können bei Verwendung einer HALCON-Kalibrierplatte mit dem Operator `find_calib_object` gewonnen werden. Alternativ kann bei HALCON-Kalibrierplatten mit rechteckiger Markenanordnung auch eine Kombination der beiden Operatoren `find_caltab` und `find_marks_and_pose` verwendet werden.

Einheiten der Parameter: Die HALCON-Kalibrierplatten verwenden Meter als Einheit. Die Kameraparameter gehen von entsprechenden Einheiten aus. Selbstverständlich kann die Kalibrierung auch in anderen Einheiten durchgeführt werden, doch müssen dann die zugehörigen Parameter adaptiert werden. Unten werden die HALCON Standard-Einheiten gelistet:

	Parameter	**Einheit**
Extern	RotX, RotY, RotZ	deg, deg, deg
	TransX, TransY, TransZ	m, m, m
Intern	Cx, Cy	px, px
	Focus	m
	ImagePlaneDist	m
	ImageWidth, ImageHeight	px, px
	K1, K2, K3	m^{-2}, m^{-4}, m^{-6}
	Kappa (κ)	m^{-2}
	P1, P2	m^{-1}, m^{-1}
	Magnification	- (Skalar)
	Sx, Sy	m/px, m/px
	Tilt, Rot	deg, deg
	Vx, Vy, Vz	$m/scanline$, $m/scanline$, $m/scanline$

Zusätzliche Information über die Kalibrierung

Die folgenden Abschnitte befassen sich jeweils mit einzelnen Fragen, die sich bei der Verwendung von `calibrate_cameras` stellen und sollen daher sowohl dem Verständnis als auch als Leitfaden für die eigene Anwendung dienen.

Was eignet sich als passender Kalibrierkörper? Hochgenaue Kalibrierkörper können Sie in verschiedenen Größen und Materialien über Ihren lokalen HALCON-Vertrieb erhalten. Diese Kalibrierplatten werden mit dazugehörigen Beschreibungsdateien ausgeliefert und können einfach mit `find_calib_object` extrahiert werden.

Es ist auch möglich, beliebige Objekte zur Kalibrierung zu verwenden. Einzige Voraussetzung ist, dass der Kalibrierkörper charakteristische Merkmalspunkte aufweist, die robust in Bildern erkannt werden können und deren 3D-Weltkoordination hochgenau bekannt sind. Siehe den „`Solution Guide III-C 3D Vision`" für weitere Informationen.

Selbst ausgedruckte Kalibrierplatten sind normalerweise nicht genau genug für Anwendungen mit hohen Präzisionsanforderungen.

Wie nehme ich geeignete Bilder auf? Mit der zu kalibrierenden Kombination aus Objektiv (mit fester Scharfstellung), Kamera und Framegrabber wird der Kalibrierkörper aufgenommen, vgl. `open_framegrabber` bzw. `grab_image`.

Ihr lokaler Vertrieb bietet zur Kamerakalibrierung zwei verschiedene HALCON-Kalibrierkörper an: Kalibrierplatten mit hexagonaler Markenanordnung (siehe `create_caltab`) und Kalibrierplatten mit rechteckiger Markenanordnung (siehe `gen_caltab`). Da sich die beiden Kalibrierkörper deutlich voneinander unterscheiden sind bei deren Verwendung neben den allgemeinen Hinweisen zusätzliche, kalibrierplattenspezifische Besonderheiten zu beachten (siehe unten):

Die Parameter und Hinweise in der folgenden Liste sollten bei der Aufnahme von Kalibrierbildern beachtet werden. Für eine erfolgreiche Kalibrierung müssen der Kameraaufbau und die Bilder gewissen Qualitätsansprüchen genügen. Diese Ansprüche sind von der Aufgabe und den damit verbunden Anforderungen abhängig. Um eine Orientierungshilfe zu geben werden hier Empfehlungen für einen einfachen Aufbau mit einer Kamera gegeben.

Es ist zu beachten, dass sich je nach Anwendungsbereich und verwendetem Kalibrierobjekt die optimalen Eigenschaften der Kalibrierbilder unterscheiden können. Die folgenden Werte und Hinweise sind Empfehlungen für einen Kameraaufbau mit einer Kamera.

Bezüglich des Kameraaufbaus:

- **Blende** Die Blendeneinstellung der Kamera darf während der Aufnahmen nicht verändert werden. Wird die Blende nach der Kalibrierung verändert muss die Kamera neu kalibriert werden.
- **Kamerapose** Die Position der Kamera darf während der Aufnahme der Bilder nicht verändert werden.
- **Fokus** Die Kalibrierbilder sollten möglichst scharfgestellt sein, d.h. Übergänge zwischen Objekten sollten klar abgetrennt sein. Der Fokus, beziehungsweise die Brennweite darf während der Aufnahme der Bilder nicht verändert werden.

Bezüglich der Platzierung der Kalibrierplatte:

- **Sichtfeldabdeckung und Ausrichtung** Durch die Serie von Kalibrierbildern sollte jeder Teil des Kamerasichtfeldes mindestens einmal durch eine Platte abgedeckt werden. Die Kalibrierplatte kann dabei auch das gesamte Sichtfeld ausfüllen. Die Orientierung der Platte sollte innerhalb der Serie von Bildern variieren.
- **Verkippung** Die Serie von Kalibrierbildern sollte außerdem Bilder von gekippten Kalibrierplatten enthalten. Dabei sollte die Platte in verschiedene Richtungen um einen Winkel von etwa 30-45° gekippt werden. Falls der empfohlene Winkel nicht erreicht werden kann, z.B. wegen begrenzter Tiefenschärfe, sollte die Platte zumindest so weit wie mit dem Kameraaufbau möglich gekippt werden.
- **Anzahl Bilder/ Posen der Kalibrierplatte**
  - Platte mit hexagonal angeordneten Marken: Mindestens 6 Bilder (Nicht alle, aber mindestens 4 davon mit gekippter Platte).
  - Platte mit rechteckig angeordneten Marken: Mindestens 15 Bilder.

  Nichtsdestotrotz müssen die Posen der Kalibrierplatten auch die weiteren Anforderungen erfüllen.
- **Seitenverkehrte Aufnahme der Platte** Die Kalibrierplatte darf nicht seitenverkehrt aufgenommen werden. Dies kann beispielsweise der Fall sein wenn eine Kalibrierplatte aus Glas verwendet wird und ihre Rückseite aufgenommen wird oder wenn eine Zeilenkamera sich nicht nach unten in Bezug auf das Bildkoordinatensystem bewegt (d.h. falls V_y negativ ist).

Bezüglich Bildeigenschaften und -inhalt:

- **Abdeckung der Kalibriermarken** Wie viel von den Kalibriermarken mindestens im Bild enthalten sein muss hängt von der verwendeten Kalibrierplate ab.
  - Platte mit hexagonal angeordneten Marken: Mindestens ein Suchmuster muss sichtbar sein. Wenn mindestens zwei Suchmuster zu sehen sind kann erkannt werden, ob die Kalibrierplatte gespiegelt ist. In diesem Fall wird ein entsprechender Fehler zurückgegeben.
  - Platte mit rechteckig angeordneten Marken: Platte muss komplett sichtbar sein, da das Suchmuster der Rahmen ist, der die Kalibriermarken umgibt.

  Dennoch ist es besser, wenn mehr Kalibriermarken für die Kamera sichtbar sind und mehr vom Sichtfeld durch die Kalibrierplatte ausgefüllt ist.
- **Markendurchmesser** Die Marken der Kalibrierplatte sollten in jedem Bild mindestens einen Durchmesser von 20 Pixeln aufweisen. Diese Anforderung ist essentiell für eine erfolgreiche Kalibrierung.
- **Kontrast** Der Kontrast zwischen hellen und dunklen Bereichen der Kalibrierplatte sollte mindestens 100 Grauwerte betragen (bezogen auf Byte-Bilder).
- **Überbelichtung** Um Überbelichtung zu vermeiden sollten Grauwerte einen Wert von 240 (bezogen auf Byte-Bilder), besonders in der Nähe von Kalibriermarken, nicht überschreiten.
- **Homogenität** Die Kalibrierplatte sollte homogen beleuchtet werden und Reflexionen sollten vermieden werden. Als grobe Faustregel sollte der Wertebereich der hellen Plattenbereiche 45 Grauwerte nicht überschreiten (bezogen auf Byte-Bilder).

Welches Verzeichnungsmodell soll ich verwenden? Es stehen zwei unterschiedliche Verzeichnungsmodelle zur Verfügung: Das Divisionsmodell und das Polynommodell. Das Divisionsmodell verwendet einen Parameter, um die radiale Verzeichnung zu modellieren, während das Polynommodell fünf Parameter verwendet, um die radiale Verzeichnung und die tangentiale Verzeichnung zu modellieren (siehe die Abschnitte „Kameraparameter“ und „Zugrundeliegendes 3D-Kameramodell“).

Der Vorteil des Divisionsmodells ist, dass die Korrektur der Verzeichnung schneller berechnet werden kann, insbesondere, wenn Weltkoordinaten in ein verzeichnetes Bild projiziert werden und die Verzeichnung somit in der umgekehrten Richtung angebracht werden muss. Des Weiteren liefert das Divisionsmodell typischerweise stabilere Ergebnisse, wenn nur wenige Kalibrierbilder verwendet werden oder das Sichtfeld nicht ausreichend durch Kalibrierkörper abgedeckt ist. Der Hauptvorteil des Polynommodells ist, dass es die Verzeichnung genauer modellieren kann, weil einerseits Terme höherer Ordnung zur Modellierung der radialen Verzeichnung verwendet werden und andererseits auch die tangentiale Verzeichnung modelliert wird.

Normalerweise sollte das Divisionsmodell für die Kalibrierung verwendet werden. Reicht die Genauigkeit der Kalibrierung nicht aus, kann das Polynommodell verwendet werden. Hierbei ist dann aber besonders darauf zu achten, dass durch die Folge von Kalibrierbildern das gesamte Gebiet abgedeckt werden muss, in dem später gemessen werden soll. Die Verzeichnungskorrektur kann außerhalb des Gebietes, das durch Kalibrierkörper abgedeckt ist, fehlerhaft sein. Dies gilt sowohl für die Bildränder als auch für Bereiche innerhalb des Bildes, die nicht durch einen Kalibrierkörper abgedeckt sind.

Zugrundeliegendes 3D-Kameramodell

Ganz allgemein bedeutet Kamerakalibrierung, diejenigen Parameter exakt zu bestimmen, die die (optische) Abbildung eines beliebigen 3D-Weltpunktes $\mathbf{p}^w$ im Raum auf ein (Sub-)Pixel (r, c) im Bild modellieren. Dies ist insbesondere von Bedeutung, wenn aus dem Kamerabild die ursprüngliche räumliche Lage von Gegenständen im Bild bestimmt werden soll, beispielsweise bei der Vermessung von Werkstücken. Das passende Projektionsmodell ist von den verwendeten Kameratypen abhängig.

Für die Modellierung dieses Abbildungsvorganges, der durch die Kombination von Kamera, Objektiv und Framegrabber festgelegt ist, stellt HALCON die folgenden 3D-Kameramodelle zur Verfügung:

Lochkamera mit Flächensensor: Die Kombination einer Kamera mit Flächensensor und einem Objektiv, das auf der Objektseite eine perspektivische Projektion durchführt, und das radiale und tangentiale Verzeichnung aufweisen kann. Das Objektiv kann ein Tiltobjektiv sein, d.h. die optische Achse kann schräg zum Sensor der Kamera stehen (dies wird manchmal auch als Scheimpflug-Objektiv bezeichnet). Da auch hyperzentrische Objektive eine perspektivische Projektion durchführen, sind Kameras mit hyperzentrischen Objektiven ebenfalls Lochkameras. Die Modelle für normale (d.h. nicht verschwenkbare) perspektivische oder bildseitig telezentrische Objektive sind identisch. Im Gegensatz hierzu unterscheiden sich die Modelle für perspektivische und bildseitig telezentrische Tiltobjektive erheblich, wie unten beschrieben.

Telezentrische Kamera mit Flächensensor: Die Kombination einer Kamera mit Flächensensor mit einem Objektiv, das auf der Objektseite eine telezentrische Projektion (also eine Parallelprojektion) durchführt, und das radiale und tangentiale Verzeichnung aufweisen kann. Das Objektiv kann ein Tiltobjektiv sein. Die Modelle für normale (d.h. nicht verschwenkbare) beidseitig oder objektseitig telezentrische Objektive sind identisch. Im Gegensatz hierzu unterscheiden sich die Modelle für beidseitig und objektseitig telezentrische Tiltobjektive erheblich, wie unten beschrieben.

Lochkamera mit Zeilensensor: Die Kombination einer Kamera mit Zeilensensor mit einem zentralperspektivisch abbildenden Objektiv, das radiale Verzeichnung aufweisen kann. Tiltobjektive werden derzeit nicht für Zeilenkameras unterstützt.

Telezentrische Kamera mit Zeilensensor: Die Kombination einer Kamera mit Zeilensensor mit einem telezentrisch abbildenden Objektiv, das radiale Verzeichnung aufweisen kann. Tiltobjektive werden derzeit nicht für Zeilenkameras unterstützt.

Um einen 3D-Punkt $\mathbf{p}^w = (\mathbf{x}^w, \mathbf{y}^w, \mathbf{z}^w)^T$ in Weltkoordinaten in einen 2D-Punkt $\mathbf{q}^i = (\mathbf{r}, \mathbf{c})^T$ in Pixelkoordinaten umzuwandeln, ist eine Kette von Transformationen notwendig:

$$\mathbf{p}^w \rightarrow \mathbf{p}^c \rightarrow \mathbf{q}^c \rightarrow \tilde{\mathbf{q}}^c \left[\rightarrow \mathbf{q}^t\right] \rightarrow \mathbf{q}^i$$

$\mathbf{p}^w$	3D Punkt in Weltkoordinaten
$\mathbf{p}^c$	3D Punkt, transformiert in das Kamerakoordinatensystem
$\mathbf{q}^c$	Punkt, projiziert in die Bildebene (in metrischen Koordinaten)
$\tilde{\mathbf{q}}^c$	Punkt unter Berücksichtigung der Linsenverzeichnung
$\mathbf{q}^t$	Bei Verwendung eines Tiltobjektivs liegt $\tilde{\mathbf{q}}^c$ nur auf einer virtuellen Bildebene eines Systems ohne Tiltobjektiv. In diesem Fall wird $\tilde{\mathbf{q}}^c$ auf den Punkt $\mathbf{q}^t$ in der gekippten Bildebene projiziert.
$\mathbf{q}^i$	Bildpunkt in Pixelkoordinaten

Im Folgenden werden die einzelnen Schritte genauer erläutert. Für eine noch detailliertere Ausführung und Modellskizzen verweisen wir auf das Kapitel „Basics“, Abschnitt „Camera Model and Parameters“ im „`Solution Guide III-C 3D Vision`“.

Transformationsschritt 1: $\mathbf{p}^w \rightarrow \mathbf{p}^c$ Der Punkt $\mathbf{p}^w$ wird vom Welt- in das Kamerakoordinatensystem transformiert (Punkte in Form von homogenen Vektoren, vgl. `affine_trans_point_3d`):

$$\begin{pmatrix} \mathbf{p}^c \\ 1 \end{pmatrix} = \begin{pmatrix} x^c \\ y^c \\ z^c \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0\,0\,0 & 1 \end{bmatrix} \cdot \begin{pmatrix} \mathbf{p}^w \\ 1 \end{pmatrix}$$

mit $\mathbf{R}$ und $\mathbf{T}$ als Rotations- und Translationsmatritzen (siehe Kapitel „Basics“, Abschnitt „3D Transformations and Poses“ im „`Solution Guide III-C 3D Vision`“ für detaillierte Informationen).

Transformationsschritt 2: $\mathbf{p}^c \rightarrow \mathbf{q}^c$ Falls als Kameramodell das einer **Lochkamera mit Flächensensor** verwendet wird, wird die Projektion von $\mathbf{p}^c = (\mathbf{x}^c, \mathbf{y}^c, \mathbf{z}^c)^T$ in die Bildebene mit der folgenden Gleichung beschrieben:

$$\mathbf{q}^c = \begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{\mathbf{z}^c} \begin{pmatrix} \mathbf{x}^c \\ \mathbf{y}^c \end{pmatrix}$$

wobei $f =$ `Focus` ist. Für Kameras mit hyperzentrischen Objektiven gilt:

$$\mathbf{q}^c = \begin{pmatrix} u \\ v \end{pmatrix} = \frac{-f}{\mathbf{z}^c} \begin{pmatrix} \mathbf{x}^c \\ \mathbf{y}^c \end{pmatrix}$$

Falls eine **telezentrische Kamera mit Flächensensor** verwendet wird, wird die Projektion mit der folgenden Gleichung beschrieben:

$$\mathbf{q}^c = \begin{pmatrix} u \\ v \end{pmatrix} = m \begin{pmatrix} \mathbf{x}^c \\ \mathbf{y}^c \end{pmatrix}$$

wobei $m =$ `Magnification` ist.

Transformationsschritt 3: $\mathbf{q}^c \rightarrow \tilde{\mathbf{q}}^c$ Für alle Kameraarten kann die Verzeichnung entweder mit dem Divisionsmodell oder mit dem Polynommodell modelliert werden.

Das **Divisionsmodell** verwendet den Parameter `Kappa`, um die radiale Verzeichnung zu modellieren.

Die folgenden Gleichungen werden verwendet, um mit Hilfe des Divisionsmodells die in der Bildebene gegebenen verzeichneten Koordinaten in Koordinaten ohne Verzeichnungen zu transformieren:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{1 + \kappa(\tilde{u}^2 + \tilde{v}^2)} \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix}$$

Diese Gleichungen lassen sich analytisch invertieren. Dies führt zu den folgenden Gleichungen, die mit Hilfe des Divisionsmodells verzeichnungsfreie Koordinaten zu Koordinaten mit Verzeichnung transformieren:

$$\tilde{\mathbf{q}}^c = \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} = \frac{2}{1 + \sqrt{1 - 4\kappa(u^2 + v^2)}} \begin{pmatrix} u \\ v \end{pmatrix}$$

Das **Polynommodell** verwendet drei Parameter (K_1, K_2, K_3), um die radiale Verzeichnung zu modellieren und zwei Parameter (P_1, P_2), um die tangentiale Verzeichnung zu modellieren.

Die folgenden Gleichungen werden verwendet, um mit Hilfe des Polynommodells die in der Bildebene gegebenen verzeichneten Koordinaten in Koordinaten ohne Verzeichnungen zu transformieren:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \tilde{u} + \tilde{u}(K_1 r^2 + K_2 r^4 + K_3 r^6) + P_1(r^2 + 2\tilde{u}^2) + 2P_2\tilde{u}\tilde{v} \\ \tilde{v} + \tilde{v}(K_1 r^2 + K_2 r^4 + K_3 r^6) + 2P_1\tilde{u}\tilde{v} + P_2(r^2 + 2\tilde{v}^2) \end{pmatrix}$$

mit $r = \sqrt{\tilde{u}^2 + \tilde{v}^2}$

Diese Gleichungen sind nicht analytisch invertierbar. Daher müssen die verzeichneten Koordinaten numerisch aus den Koordinaten ohne Verzeichnungen bestimmt werden.

Zusätzlicher Transformationsschritt für Tiltobjektive: $\tilde{\mathbf{q}}^c \rightarrow \mathbf{q}^t$ Falls es sich beim Objektiv um ein **Tiltobjektiv** handelt, wird die Verkippung des Objektivs relativ zur Bildebene durch die zwei Parameter rot und $tilt$ beschrieben.

Bei diesem Schritt muss, wie unten beschrieben, zwischen verschiedenen Arten von Tiltobjektiven unterschieden werden. Siehe Kapitel „Basics“, Abschnitt „Camera Model and Parameters“ im „`Solution Guide III-C 3D Vision`“ für einen Überblick über die verschiedenen Arten von Tiltobjektiven.

Für **zentralperspektivisch abbildende Tiltobjektive** und **objektseitig telezentrische Tiltobjektive** (die eine perspektivische Projektion auf der Bildseite des Objektivs durchführen), wird die Projektion des Punktes $\tilde{\mathbf{q}}^c = (\tilde{u}, \tilde{v})^T$ in den Punkt $\mathbf{q}^t = (\hat{u}, \hat{v})^T$, der in der verkippten Bildebene liegt, durch eine projektive 2D-Transformation, d.h. durch eine homogene 3×3-Matrix $\mathbf{H}$, beschrieben (siehe `projective_trans_point_2d`):

$$\begin{pmatrix} \mathbf{q}^t \\ q_w^t \end{pmatrix} = \mathbf{H} \cdot \begin{pmatrix} \tilde{\mathbf{q}}^c \\ 1 \end{pmatrix}$$

wobei q_w^t die zusätzliche Koordinate aus der projektiven Transformation eines homogenen Punktes ist.

$$\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} = \begin{pmatrix} q_{11}q_{33} - q_{13}q_{31} & q_{21}q_{33} - q_{23}q_{31} & 0 \\ q_{12}q_{33} - q_{13}q_{32} & q_{22}q_{33} - q_{23}q_{32} & 0 \\ q_{13}/d & q_{23}/d & q_{33} \end{pmatrix}$$

wobei $d =$ `ImagePlaneDist` und

$$\mathbf{Q} = \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{pmatrix} = \begin{pmatrix} (\cos\rho)^2(1-\cos\tau) + \cos\tau & \cos\rho\sin\rho(1-\cos\tau) & \sin\rho\sin\tau \\ \cos\rho\sin\rho(1-\cos\tau) & (\sin\rho)^2(1-\cos\tau) + \cos\tau & -\cos\rho\sin\tau \\ -\sin\rho\sin\tau & \cos\rho\sin\tau & \cos\tau \end{pmatrix}$$

mit $\rho = rot$ und $\tau = tilt$.

Für **bildseitig telezentrische Tiltobjektive** und **beidseitig telezentrische Tiltobjektive** (die eine Parallelprojektion auf der Bildseite des Objektivs durchführen) wird die Projektion auf die verkippte Bildebene durch eine lineare 2D-Transformation, d.h. eine 2×2-Matrix, beschrieben:

$$\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} = \frac{1}{q_{11}q_{22} - q_{12}q_{21}} \begin{pmatrix} q_{22} & -q_{12} \\ -q_{21} & q_{11} \end{pmatrix}$$

wobei $\mathbf{Q}$ definiert ist wie bei zentralperspektivisch abbildenden Objektiven.

Transformationsschritt 4: $\mathbf{q}^t \rightarrow \mathbf{q}^i$ / $\tilde{\mathbf{q}}^c \rightarrow \mathbf{q}^i$ Schließlich wird die Transformation des Punktes $\tilde{\mathbf{q}}^c = (\tilde{u}, \tilde{v})^T$ (bzw. $\mathbf{q}^t$ falls ein Tiltobjektiv verwendet wurde) vom Koordinatensystem der Bildebene ins Pixelkoordinatensystem folgendermaßen beschrieben:

$$\mathbf{q}^i = \begin{pmatrix} r \\ c \end{pmatrix} = \begin{pmatrix} \frac{\tilde{v}}{S_y} + C_y \\ \frac{\tilde{u}}{S_x} + C_x \end{pmatrix}$$

Für **Kameras mit Zeilensensor** muss auch die Relativbewegung zwischen Kamera und Objekt modelliert werden. In HALCON werden hierzu die folgenden Annahmen getroffen:

1. Die Kamera bewegt sich mit konstanter Geschwindigkeit entlang einer Geraden.
2. Die Orientierung der Kamera ist konstant.
3. Die Relativbewegung ist für alle Bilder gleich.

Diese Bewegung wird durch den Bewegungsvektor $V = (V_x, V_y, V_z)^T$ beschrieben. Dieser Vektor muss in der Einheit [Meter/Bildzeile] im Kamerakoordinatensystem gegeben sein. Er beschreibt die Bewegung der Kamera in Bezug auf ein unbewegtes Objekt. Dies ist äquivalent zu der Annahme einer unbewegten Kamera und einem Objekt, das sich entlang $-V$ bewegt.

Das Kamerakoordinatensystem ist für Kameras mit Zeilensensor folgendermaßen definiert. Der Ursprung liegt im Projektionszentrum (für Lochkameras) bzw. im Zentrum der Verzeichnungen (für telezentrische Kameras). Die z-Achse ist mit der optischen Achse identisch und ist so orientiert, dass alle Punkte, die von der Kamera aus sichtbar sind, positive z-Koordinaten haben. Die y-Achse ist senkrecht zur Sensorzeile und zur z-Achse. Sie ist so orientiert, dass der Bewegungsvektor eine positive y-Komponente hat. Die x-Achse ist senkrecht zur y- und z-Achse. Sie ist so orientiert, dass x-, y- und z-Achse ein rechtshändiges Koordinatensystem bilden.

Da sich die Kamera während der Bildaufnahme über das Objekt bewegt, bewegt sich auch das Kamerakoordinatensystem relativ zum Objekt. Das bedeutet, dass jede Bildzeile von einer anderen Position aus aufgenommen wird, dass es also prinzipiell für jede Bildzeile eine eigene Pose gibt. Zur Vereinfachung beziehen sich in HALCON alle Transformationen zwischen Weltkoordinaten und Kamerakoordinaten auf die Pose der ersten Bildzeile. Die Bewegung V wird bei der Projektion des Punktes $\mathbf{p}^c$ in das Bild berücksichtigt. Infolgedessen wird vom Operator `find_calib_object` auch nur die Pose der ersten Bildzeile bestimmt (und von `calibrate_cameras` in den Kalibrierergebnissen gespeichert).

Die Transformation von Welt- in Kamerakoordinaten ($\mathbf{p}^w \rightarrow \mathbf{p}^c$) ist für **Zeilenkameras** identisch. Daher wird Transformationsschritt 1, wie oben beschrieben auch hier angewandt.

Für **Lochkameras mit Zeilensensor** ist die Abbildung des Punktes $\mathbf{p}^c$, der im Kamerakoordinatensystem gegeben ist, auf ein (Sub-)Pixel (r, c) im Bild folgendermaßen definiert:

Mit

$$\mathbf{p}^c = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

muss das folgende Gleichungssystem nach m, $\tilde{u}$ und t aufgelöst werden:

$$\begin{aligned} m \cdot u(\tilde{u}, p_v) &= x - t \cdot V_x \\ m \cdot v(\tilde{u}, p_v) &= y - t \cdot V_y \\ m \cdot \text{Focus} &= z - t \cdot V_z, \end{aligned}$$

wobei $u(\tilde{u}, \tilde{v})$ und $v(\tilde{u}, \tilde{v})$ die Entzerrungsfunktionen sind, die oben für Flächenkameras beschrieben wurden, und $p_v = -S_y \cdot C_y$ ist.

Für **telezentrische Kameras mit Zeilensensor** muss das folgende Gleichungssystem nach $\tilde{u}$ und t aufgelöst werden:

$$\begin{aligned} u(\tilde{u}, p_v)/\text{Magnification} &= x - t \cdot V_x \\ v(\tilde{u}, p_v)/\text{Magnification} &= y - t \cdot V_y, \end{aligned}$$

wobei $u(\tilde{u}, \tilde{v})$, $v(\tilde{u}, \tilde{v})$ und p_v wie oben definiert sind.

Die obigen Formeln beinhalten bereits die Kompensation der Verzeichnungen.

Zum Schluss wird der Punkt ins Pixelkoordinatensystem transformiert:

$$\mathbf{q}^i = \begin{pmatrix} r \\ c \end{pmatrix} = \begin{pmatrix} t \\ \frac{\tilde{u}}{S_x} + C_x \end{pmatrix}.$$

Weitere Beschränkungen in Bezug auf spezifische Kameratypen

Für **Lochkameras** gilt: Falls die Kalibrierplatten in allen Bildern parallel zueinander sind (insbesondere, falls sie alle in derselben Ebene liegen), ist es unmöglich, den `Focus` gemeinsam mit allen sechs externen Parametern zu bestimmen. So ist es z.B. unmöglich, in diesem Fall `Focus` und den Abstand der Kalibrierplatte zur Kamera zu bestimmen. Um alle Kameraparameter eindeutig bestimmen zu können, muss die Kalibrierplatte in unterschiedlichen Orientierungen in den Bildern aufgenommen werden. Insbesondere sollte die Kalibrierplatte um die x- und y-Achsen des Kamerakoordinatensystems verkippt werden, d.h. sie sollte relativ zur Bildebene verkippt werden.

Bei **telezentrischen Objektiven** hat der Abstand der Kalibrierplatte zur Kamera keine Auswirkung auf das Bild der Kalibrierplatte und kann nicht bestimmt werden. Daher wird die z-Komponente der Ergebnis-Pose in den Kalibrierergebnissen auf 1 m gesetzt. Weiterhin kann, wie zuvor beschrieben, für Zeilenkameras mit telezentrischen Objektiven V_z nicht bestimmt werden und wird daher auf seinem Startwert belassen, außer es handelt sich um einen Aufbau mit mehreren Kameras, die einen gemeinsamen Bewegungsvektor besitzen, wo V_z bestimmt werden kann.

Für **Tiltobjektive** kann die Verkippung umso präziser bestimmt werden, je größer die Verzeichnungen des Objektivs sind. Für Objektive mit geringer Verzeichnung kann die Verkippung nicht robust bestimmt werden. Daher können die optimierten Verkippungsparameter möglicherweise signifikant von den nominalen Verkippungsparametern des Aufbaus abweichen. In so einem Fall prüfen Sie bitte `Error`. Falls `Error` klein ist, beschreibt der optimierte Parametersatz die Abbildungsgeometrie innerhalb des kalibrierten Volumens konsistent und kann für genaue Messungen verwendet werden.

Für **perspektivische Tiltobjektive** und **objektseitig telezentrische Tiltobjektive** kann der Abstand der Bildebene nur eindeutig bestimmt werden, falls der Tilt nicht 0 Grad ist. Je kleiner der Tilt ist, desto ungenauer kann der Abstand der Bildebene bestimmt werden. Daher kann der optimierte Abstand der Bildebene möglicherweise signifikant vom nominalen Abstand der Bildebene des Aufbaus abweichen. In so einem Fall prüfen Sie bitte `Error`. Falls `Error` klein ist, beschreibt der optimierte Parametersatz die Abbildungsgeometrie innerhalb des kalibrierten Volumens konsistent und kann für genaue Messungen verwendet werden.

Für **perspektivische Tiltobjektive** und **objektseitig telezentrische Tiltobjektive**, die um die horizontale oder vertikale Achse verschwenkt sind, d.h. für die der Rotationswinkel 0, 90, 180 oder 270 Grad beträgt, können der Tilt-Winkel $tilt$, Skalierungsfaktor S_x, die Brennweite f (für perspektivische Tiltobjektive) bzw. der Vergrößerungsmaßstab m (für objektseitig telezentrische Tiltobjektive) und der Abstand der verkippten Bildebene vom Zentrum der perspektivischen Projektion d nicht eindeutig bestimmt werden. In diesem Fall sollte S_x durch folgenden Aufruf von der Optimierung ausgeschlossen werden:

```
set_calib_data (CalibDataID, 'camera', 'general',
                'excluded_settings', 'sx').
```

Weiterhin ist zu beachten, dass es für Tiltobjektive nur möglich ist, $tilt$ und rot gleichzeitig zu bestimmen. Dies beruht auf einer Implementationsentscheidung, die dazu führt, dass die Optimierung numerisch stabiler wird. Folglich können $tilt$ und rot von der Optimierung nur gemeinsam und mit folgendem Aufruf ausgeschlossen werden:

```
set_calib_data (CalibDataID, 'camera', 'general',
                'excluded_settings', 'tilt').
```

Lochkameras mit Tiltobjektiven mit großer Brennweite haben näherungsweise ein telezentrisches Abbildungsverhalten. Auch in diesem Fall sind, wie bereits beschrieben, S_x und die Verkippungsparameter $tilt$ und rot korreliert. Sie können gleichzeitig nur unpräzise bestimmt werden. In diesem Fall ist es ratsam, S_x von der Optimierung auszuschließen.

Bei telezentrischen Objektiven gibt es immer zwei mögliche Lagen einer Kalibrierplatte pro Bild. Daher ist es unmöglich, zu entscheiden, welche der beiden Lagen tatsächlich vorhanden ist. Diese Mehrdeutigkeit wirkt sich auch auf die Schwenkparameter $tilt$ und rot von telezentrischen Tiltobjektiven aus. Folglich kann die Kamerakalibrierung, abhängig von den Startparametern für $tilt$ und rot alternative Werte statt der nominalen zurückliefern.

In so einem Fall prüfen Sie bitte Error. Falls Error klein ist, beschreibt der optimierte Parametersatz die Abbildungsgeometrie innerhalb des kalibrierten Volumens konsistent und kann für genaue Messungen verwendet werden.

Bei **Zeilenkameras mit dem polynomiellen Verzeichnungsmodell** (sowohl für Lochkameras als auch für Kameras mit telezentrischen Objektiven) sind die Parameter P_1 and P_2 hochgradig korreliert mit anderen Parametern des Kameramodells. Daher können sie typischerweise nicht verlässlich bestimmt werden und sollten von der Kalibrierung durch den folgenden Aufruf ausgeschlossen werden:

```
set_calib_data (CalibDataID, 'camera', 'general',
                'excluded_settings', 'poly_tan_2').
```

14.1 Binokular

binocular_calibration (: : NX, NY, NZ, NRow1, NCol1, NRow2, NCol2, StartCamParam1, StartCamParam2, NStartPose1, NStartPose2, EstimateParams : CamParam1, CamParam2, NFinalPose1, NFinalPose2, RelPose, Errors)

Bestimmung aller Kameraparameter eines binokularen Stereoaufbaus.

Im Allgemeinen bedeutet binokulare Kalibrierung die exakte Bestimmung der Parameter, welche die 3D Rekonstruktion eines 3D Punktes von den korrespondierenden Abbildungen dieses Punktes in einem binokularen Stereo System modellieren. Diese Rekonstruktion wird durch die internen Parameter CamParam1 von Kamera 1 und CamParam2 von Kamera 2 bestimmt, welche das zugrunde liegende Kameramodell beschreiben, und von den externen Parametern RelPose, welche die relative Lage des Kamerasystems 2 im Kamerasystem 1 beschreiben.

Die bekannten 3D Modellpunkte (mit den Koordinaten NX, NY, NZ) werden in die Bildebene der beiden Kameras (Kamera 1 und Kamera 2) projiziert. Die Summe der quadrierten Abstände zwischen diesen Projektionen und den korrespondierenden Bildpunkten (mit den Koordinaten NRow1, NCol1 für Kamera 1 und NRow2, NCol2 für Kamera 2) wird daraufhin minimiert. Es muss darauf geachtet werden, dass alle Modellpunkte in beiden Bildern sichtbar sein müssen. Das zugrunde liegende Kameramodell wird im Kapitel Kalibrierung erläutert. Das Kameramodell für perspektivische und telezentrische Flächenkameras sowie für telezentrische Zeilenkameras wird durch 9 bis 16 Parameter (für jede Kamera separat) beschrieben (siehe Kalibrierung). Die initialen Werte dieser internen Parameter werden in StartCamParam1 für Kamera 1 und StartCamParam2 für Kamera 2 übergeben. Als Näherungswert kann man diese den Datenblättern der Kameras entnehmen. Zusätzlich werden die initialen Näherungswerte NStartPose1 und NStartPose2 für die einzelnen 3D-Lagen des 3D Kalibriermodells bezüglich des jeweiligen Kamerakoordinatensystems (*ccs*) von Kamera 1 und Kamera 2 benötigt. Diese 3D-Lagen werden in der Form ${}^{ccs}\mathbf{P}_{wcs}$ erwartet. Dabei steht *wcs* für das Weltkoordinatensystem, siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". Die 3D-Lagen kann man durch den Operator find_marks_and_pose erhalten. Da der Kalibrieralgorithmus gleichzeitig Korrespondenzen zwischen gemessenen Bild- und bekannten Modellpunkten aus mehreren Bildpaaren behandeln kann, müssen die 3D-Lagen (NStartPose1, NStartPose2) und die gemessenen Punkte (NRow1, NCol1, NRow2, NCol2) in einer zu den Bildern korrespondierenden Reihenfolge übergeben werden.

Der Eingabeparameter EstimateParams legt die zu berechnenden Kameraparameter fest. Üblicherweise wird dieser Parameter auf *'all'* gesetzt, d.h. alle externen (Translationen und Rotationen) und alle internen Kameraparameter werden bestimmt. Falls z.B. die internen Parameter schon bestimmt worden sind (z.B. durch einen früheren Aufrufen von binocular_calibration), ist es oftmals interessant, lediglich die 3D-Lage der Kamerasysteme zueinander zu bestimmen (RelPose). In diesem Fall kann in EstimateParams der Wert *'pose_rel'* übergeben werden. Auch die internen Kameraparameter lassen sich mit den Werten *'cam_param1'* bzw. *'cam_param2'* zusammenfassen. Andernfalls enthält EstimateParams ein Tupel von Stringwerten, welche die zu schätzenden Kameraparameter festlegen. Wenn das Polynommodell für die Modellierung der Verzeichnungen verwendet wird, ist zu beachten, dass nur die Werte *'k1_i'*, *'k2_i'* und *'k3_i'* einzeln in EstimateParams gesetzt werden können. *'p1'* und *'p2'* kann nur in der Gruppe *'poly_tan_2_i'* gesetzt werden (wobei *'i'* für den Index der Kamera steht). *'poly_i'* bezeichnet die Gruppe *'k1_i'*, *'k2_i'*, *'k3_i'* und *'poly_tan_2_i'*.

In der folgenden Liste sind alle möglichen Parameter für EstimateParams zu finden:

Mögliche Parameter für EstimateParams	*Ermittelte Parameter*
'all' (default)	Alle internen Kameraparameter, sowie die relative Pose der Kameras und die Posen der Kalibrierobjekte.
'pose'	Relative Pose zwischen beiden Kameras, sowie die Posen der Kalibrierobjekte.
'pose_rel'	Relative Pose zwischen beiden Kameras.
'alpha_rel', 'beta_rel', 'gamma_rel', 'transx_rel', 'transy_rel', 'transz_rel'	Rotationswinkel und Translation der relativen Pose zwischen beiden Kameras..
'pose_caltabs'	Posen der Kalibrierobjekte.
'alpha_caltabs', 'beta_caltabs', 'gamma_caltabs', 'transx_caltabs', 'transy_caltabs', 'transz_caltabs'	Rotationswinkel und Translation der Posen der Kalibrierobjekte.
'cam_param1', 'cam_param2'	Alle internen Kameraparameter von Kamera 1 bzw. Kamera 2.
'focus1', 'magnification1', 'kappa1', 'poly_1', 'k1_1', 'k2_1', 'k3_1', 'poly_tan_2_1', 'image_plane_dist1', 'tilt1', 'cx1', 'cy1', 'sx1', 'sy1', 'focus2', 'magnification2', 'kappa2', 'poly_2', 'k1_2', 'k2_2', 'k3_2', 'poly_tan_2_2', 'image_plane_dist2', 'tilt2', 'cx2', 'cy2', 'sx2', 'sy2'	Einzelne interne Kameraparameter von Kamera 1 bzw. Kamera 2.
'common_motion_vector'	Legt fest, ob zwei Zeilenkameras einen gemeinsamen Bewegungsvektor besitzen. Dies ist der Fall, wenn die beiden Kameras starr montiert sind und das Objekt linear vor den Kameras bewegt wird oder falls die zwei starr montierten Kameras mit demselben Linearmechanismus bewegt werden. Dies wird als Voreinstellung angenommen. Deshalb ist es nur notwendig, '~common_motion_vector' zu setzen, falls sich die Kameras unabhängig voneinander in unterschiedliche Richtungen bewegen.

Es ist auch möglich, Parameter, die nicht geschätzt werden sollen, über ein vorgesetztes '~' Zeichen im String auszuschließen. Die Werte *['pose_rel', '~transx_rel']* beispielsweise haben denselben Effekt wie *['alpha_rel','beta_rel','gamma_rel','transy_rel','transz_rel']*. *['all','~focus1']* dagegen schätzt alle internen und externen Parameter bis auf die Brennweite von Kamera1. Das '~' Präfix kann mit Ausnahme von *'all'* jedem Parameterwert vorangestellt werden.

Die geschätzten Kameraparameter werden in CamParam1 für Kamera 1 und CamParam2 für Kamera 2 zurückgegeben.

Die externen Parameter werden analog zu camera_calibration zurück gegeben, die Posen der 3D Transformation des Kalibriermodells zum jeweiligen Kamerakoordinatensystems *ccs* in NFinalPose1 und NFinalPose2 zurückgegeben. Das heißt, die Posen sind in der Form ${}^{ccs}\mathbf{P}_{wcs}$ mit *wcs* als Weltkoordinatensystem des 3D Kalibriermodells, siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". Die relative Pose ${}^{ccs1}\mathbf{P}_{ccs2}$, RelPose, legt die Transformation von Punkten von *ccs2* nach *ccs1* fest. Damit hängen die finalen Transformationen gemäß der folgenden Gleichung zusammen (unter Vernachlässigung der Effekte durch die Ausgleichsrechnung der Mehrbild-Kalibrierung):

HomMat3D_NFinalPose2 = INV(HomMat3D_RelPose) * HomMat3D_NFinalPose1,

wobei HomMat3D_* die homogene Transformationsmatrix der entsprechenden Lage beschreibt und INV() eine homogene Matrix invertiert.

Die errechneten mittleren Fehler für jede Kamera, welche in `Errors` zurückgegeben werden, vermitteln einen Eindruck von der Genauigkeit der Kalibrierung. Er beschreibt einen mittleren euklidischen Abstand der mit den ermittelten Parametern ins Bild projezierten Mittelpunkte der Modellmarken von ihren Abbildungen.

Bei der Verwendung von Kameras mit telezentrischen Objektiven müssen zusätzliche Voraussetzungen für den Kalibrieraufbau gelten. Diese können im Kapitel Kalibrierung nachgelesen werden.

Achtung

Stereosysteme, die sowohl Kameras mit hyperzentrischen Objektiven als auch Kameras ohne hyperzentrische Objektive enthalten, werden nicht unterstützt. Weiterhin werden Stereosysteme, die sowohl Flächenkameras als auch Zeilenkameras enthalten, nicht unterstützt.

Parameter

- ▷ **NX** (input_control) . . . . . number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen X-Koordinaten der Kalibriermarken (in Meter).
- ▷ **NY** (input_control) . . . . . number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Y-Koordinaten der Kalibriermarken (in Meter).
  Parameteranzahl: NY == NX
- ▷ **NZ** (input_control) . . . . . number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Z-Koordinaten der Kalibriermarken (in Meter).
  Parameteranzahl: NZ == NX
- ▷ **NRow1** (input_control) . . . . . number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Zeilen-Koordinaten der extrahierten Kalibriermarken von Kamera 1 (in Pixel).
- ▷ **NCol1** (input_control) . . . . . number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Spalten-Koordinaten der extrahierten Kalibriermarken von Kamera 1 (in Pixel).
  Parameteranzahl: NCol1 == NRow1
- ▷ **NRow2** (input_control) . . . . . number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Zeilen-Koordinaten der extrahierten Kalibriermarken von Kamera 2 (in Pixel).
  Parameteranzahl: NRow2 == NRow1
- ▷ **NCol2** (input_control) . . . . . number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Spalten-Koordinaten der extrahierten Kalibriermarken von Kamera 2 (in Pixel).
  Parameteranzahl: NCol2 == NRow1
- ▷ **StartCamParam1** (input_control) . . . . . campar ⇝ *real* / integer / string
  Startwerte für die internen Parameter der Kamera 1.
- ▷ **StartCamParam2** (input_control) . . . . . campar ⇝ *real* / integer / string
  Startwerte für die internen Parameter der Kamera 2.
- ▷ **NStartPose1** (input_control) . . . . . pose(-array) ⇝ *real* / integer
  Geordnetes Tupel mit allen Startwerten der externen Parameter von Kamera 1.
  Parameteranzahl: NStartPose1 == 7 * NRow1 / NX
- ▷ **NStartPose2** (input_control) . . . . . pose(-array) ⇝ *real* / integer
  Geordnetes Tupel mit allen Startwerten der externen Parameter von Kamera 2.
  Parameteranzahl: NStartPose2 == 7 * NRow1 / NX
- ▷ **EstimateParams** (input_control) . . . . . string-array ⇝ *string*
  Zu schätzenden Kameraparameter.
  Defaultwert: 'all'
  Werteliste: `EstimateParams` ∈ {'all', 'pose', 'pose_caltabs', 'pose_rel', 'cam_param1', 'cam_param2', 'alpha_rel', 'beta_rel', 'gamma_rel', 'transx_rel', 'transy_rel', 'transz_rel', 'alpha_caltabs', 'beta_caltabs', 'gamma_caltabs', 'transx_caltabs', 'transy_caltabs', 'transz_caltabs', 'focus1', 'magnification1', 'kappa1', 'poly_1', 'k1_1', 'k2_1', 'k3_1', 'poly_tan_2_1', 'image_plane_dist1', 'tilt1', 'cx1', 'cy1', 'sx1', 'sy1', 'focus2', 'magnification2', 'kappa2', 'poly_2', 'k1_2', 'k2_2', 'k3_2', 'poly_tan_2_2', 'image_plane_dist2', 'tilt2', 'cx2', 'cy2', 'sx2', 'sy2', 'common_motion_vector'}
- ▷ **CamParam1** (output_control) . . . . . campar ⇝ *real* / integer / string
  Interne Parameter der Kamera 1.
- ▷ **CamParam2** (output_control) . . . . . campar ⇝ *real* / integer / string
  Interne Parameter der Kamera 2.

▷ **NFinalPose1** (output_control) ..... pose(-array) ⇝ *real* / integer
Geordnetes Tupel mit allen externen Parametern von Kamera 1.
Parameteranzahl: NFinalPose1 == 7 * NRow1 / NX

▷ **NFinalPose2** (output_control) ..... pose(-array) ⇝ *real* / integer
Geordnetes Tupel mit allen externen Parametern von Kamera 2.
Parameteranzahl: NFinalPose2 == 7 * NRow1 / NX

▷ **RelPose** (output_control) ..... pose ⇝ *real* / integer
Lage von Kamera 2 bezüglich Kamera 1.

▷ **Errors** (output_control) ..... real(-array) ⇝ *real*
Durchschnittlicher Fehler in Pixel.

Beispiel

```
* Open image source.
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1, \
                   'default', 'images_l.seq', 'default', 0, -1, AcqHandle1)
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1, \
                   'default', 'images_r.seq', 'default', 1, -1, AcqHandle2)

* Initialize the start parameters.
caltab_points ('caltab_30mm.descr', X, Y, Z)
StartCamParam1 := ['area_scan_division', 0.0125, 0, 7.4e-6, 7.4e-6, \
                   Width/2.0, Height/2.0, Width, Height]
StartCamParam2 := StartCamParam1
Rows1 := []
Cols1 := []
StartPoses1 := []
Rows2 := []
Cols2 := []
StartPoses2 := []

* Find calibration marks and startposes.
for i := 0 to 11 by 1
  grab_image_async (Image1, AcqHandle1, -1)
  grab_image_async (Image2, AcqHandle2, -1)
  find_caltab (Image1, CalPlate1, 'caltab_30mm.descr', 3, 120, 5)
  find_caltab (Image2, CalPlate2, 'caltab_30mm.descr', 3, 120, 5)
  find_marks_and_pose (Image1, CalPlate1, 'caltab_30mm.descr', \
                       StartCamParam1, 128, 10, 20, 0.7, 5, 100, \
                       RCoord1, CCoord1, StartPose1)
  Rows1 := [Rows1,RCoord1]
  Cols1 := [Cols1,CCoord1]
  StartPoses1 := [StartPoses1,StartPose1]
  find_marks_and_pose (Image2, CalPlate2, 'caltab_30mm.descr', \
                       StartCamParam2, 128, 10, 20, 0.7, 5, 100, \
                       RCoord2, CCoord2, StartPose2)
  Rows2 := [Rows2,RCoord2]
  Cols2 := [Cols2,CCoord2]
  StartPoses2 := [StartPoses2,StartPose2]
endfor

* Calibrate the stereo rig.
binocular_calibration (X, Y, Z, Rows1, Cols1, Rows2, Cols2, StartCamParam1, \
                       StartCamParam2, StartPoses1, StartPoses2, 'all', \
                       CamParam1, CamParam2, NFinalPose1, NFinalPose2, \
                       RelPose, Errors)
* Archive the results.
write_cam_par (CamParam1, 'cam_left-125.dat')
write_cam_par (CamParam2, 'cam_right-125.dat')
```

```
write_pose (RelPose, 'rel_pose.dat')

* Rectify the stereo images.
gen_binocular_rectification_map (Map1, Map2, CamParam1, CamParam2, \
                                 RelPose, 1, 'viewing_direction', 'bilinear', \
                                 CamParamRect1, CamParamRect2, \
                                 CamPoseRect1, CamPoseRect2, \
                                 RelPoseRect)
map_image (Image1, Map1, ImageMapped1)
map_image (Image2, Map2, ImageMapped2)
```

Ergebnis

Sind die Parameterwerte korrekt und konnten die gesuchten Parameter durch das Bündelausgleichsverfahren bestimmt werden, dann liefert `binocular_calibration` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_marks_and_pose`, `caltab_points`, `read_cam_par`

Nachfolger

`write_pose`, `write_cam_par`, `pose_to_hom_mat3d`, `disp_caltab`, `gen_binocular_rectification_map`

Siehe auch

`find_caltab`, `sim_caltab`, `read_cam_par`, `create_pose`, `convert_pose_type`, `read_pose`, `hom_mat3d_to_pose`, `create_caltab`, `binocular_disparity`, `binocular_distance`

Modul

3D Metrology

14.2 Hand-Auge

`calibrate_hand_eye` (: : CalibDataID : Errors)

Durchführen einer Hand-Auge-Kalibrierung.

Der Operator `calibrate_hand_eye` bestimmt die 3D-Lage eines Roboters („Hand“) relativ zu einer Kamera oder 3D-Sensor („Auge“) basierend auf einem Kalibrierdatenmodell `CalibDataID`. Mit den ermittelten 3D-Posen können die Posen des Objektes im Kamerakoordinatensystem in das Roboterbasiskoordinatensystem transformiert werden. Der Roboter kann dann z.B. ein inspiziertes Teil greifen. Es gibt zwei mögliche Konfigurationen von Roboter-Kamera-Systemen (Hand-Auge-Systemen): Die Kamera kann auf dem Roboter montiert sein oder stationär sein und den Roboter beobachten. Beachten Sie, dass der Begriff „Roboter“ für ein beliebiges System steht, das Objekte bewegt. Daher können mit `calibrate_hand_eye` viele verschiedene Systeme kalibriert werden, von Schwenk-Neige-Köpfen zu Vielfachs-Manipulatoren.

Eine Hand-Auge-Kalibrierung kann über eine geschlossene Verkettung von insgesamt vier euklidischen Transformationen beschrieben werden. In dieser Kette sind zwei nicht aufeinander folgende Transformationen bekannt. Diese stammen entweder aus der Robotersteuerung oder werden aus Kameradaten, z.B. beobachtete Posen eines Kalibrierkörpers, bestimmt. Die beiden anderen Transformationen sind unbekannt aber konstant und werden durch die Hand-Auge-Kalibrierung bestimmt.

Eine Hand-Auge-Kalibrierung läuft ähnlich ab wie die Kalibrierung der externen Kameraparameter (siehe Kalibrierung): Verschiedene Posen eines Kalibrierkörpers im Kamerakoordinatensystem sowie die korrespondierenden Posen des Werkzeugs im Roboterbasiskoordinatensystem werden bestimmt und im Kalibrierdatenmodell `CalibDataID` gesetzt.

Im Gegensatz zur Kamerakalibrierung wird der Kalibrierkörper nicht manuell bewegt. Dies ist die Aufgabe des Roboters. Zwei Hand-Auge Kalibrierszenarien werden unterschieden. Der Roboter bewegt entweder die Kamera (bei auf dem Roboter montierter Kamera) oder den Kalibrierkörper (bei stationärer Kamera). Die Bewegungen des Roboters werden als bekannt vorausgesetzt. Sie werden als Eingabe für die Kalibrierung benutzt und mittels `set_calib_data` im Kalibrierdatenmodell `CalibDataID` gesetzt.

Durch die Hand-Auge-Kalibrierung werden zwei Posen bestimmt: Im Szenario „bewegte Kamera“ wird die 3D-Pose des Endeffektors im Kamerakoordinatensystem (*'tool_in_cam_pose'*) und die 3D-Pose des Kalibrierkörpers in Roboterbasiskoordinaten (*'obj_in_base_pose'*) bestimmt. Im Szenario „stationäre Kamera“ wird die 3D-Pose der Roboterbasis im Kamerakoordinatensystem (*'base_in_cam_pose'*) sowie die 3D-Pose des Kalibrierkörpers im Tool-Koordinatensystem (*'obj_in_tool_pose'*) bestimmt. Ihr Posentyp ist identisch mit dem Typ der Eingabeposen.

Im Folgenden werden die zwei Hand-Auge-Konfigurationen detaillierter beschrieben, gefolgt von genereller Information über die benötigten Daten und die Vorbereitung des Kalibrierdatenmodells.

Bewegte Kamera (auf dem Roboter montiert)

In dieser Konfiguration ist der Kalibrierkörper stationär, und die Kamera wird vom Roboter an verschiedene Positionen bewegt. Die grundlegende Idee hinter der Hand-Auge-Kalibrierung ist, dass die aus einem Kalibrierbild extrahierte Information, d.h. die 3D-Lage des Kalibrierkörpers relativ zur Kamera (d.h. die externen Kameraparameter), als Kette von 3D-Lagen bzw. homogenen Transformationsmatrizen aufgefasst werden kann, vom Kalibrierkörper über die Roboter-Basis zu dessen Tool (Endeffektor) und schließlich zur Kamera:

Bewegte Kamera:

$$^{camera}\mathbf{H}_{cal} = {}^{camera}\mathbf{H}_{tool} \cdot ({}^{base}\mathbf{H}_{tool})^{-1} \cdot {}^{base}\mathbf{H}_{cal}$$

'obj_in_cam_pose' 'tool_in_cam_pose' 'tool_in_base_pose' 'obj_in_base_pose'

Aus den gegebenen Posen des Kalibrierkörpers (*'obj_in_cam_pose'*) und den Posen des Endeffektors in Roboterbasiskoordinaten (*'tool_in_base_pose'*) bestimmt der Operator `calibrate_hand_eye` die zwei unbekannten Transformationen an den Enden der Kette, d.h. die 3D-Lage des Roboter-Tools in Kamerakoordinaten ($^{camera}\mathbf{H}_{tool}$, *'tool_in_cam_pose'*) und die 3D-Lage des Kalibrierkörpers in Roboter-Basiskoordinaten ($^{base}\mathbf{H}_{cal}$, *'obj_in_base_pose'*). Diese Posen sind konstant.

Im Gegensatz dazu ist die Transformation in der Mitte der Kette, $^{tool}\mathbf{H}_{base}$, bekannt, aber unterschiedlich für jede Beobachtung des Kalibrierkörpers, da sie die 3D-Lage des Endeffektors (Tool) bezüglich des Roboterbasiskoordinatensystems beschreibt. In der gegebenen Gleichung werden die inversen Transformationsmatrizen benutzt (3D-Lage der Roboter-Basis in Roboter-Toolkoordinaten). Die Invertierung wird intern vorgenommen.

Es ist zu beachten, dass die Z-Translation von *'obj_in_base_pose'* im Rahmen der Kalibrierung von SCARA-Robotern nicht eindeutig bestimmt werden kann. Um diese Mehrdeutigkeit aufzulösen, wird die Z-Translation von *'obj_in_base_pose'* intern auf 0.0 gesetzt und *'tool_in_cam_pose'* entsprechend berechnet. Es ist notwendig, die tatsächliche Translation in Z nach der Kalibrierung zu bestimmen indem der Roboter an eine Pose bekannter Höhe im Kamerakoordinatensystem gefahren wird. Hierfür empfiehlt sich das folgende Vorgehen: Der Kalibrierkörper wird an einer beliebigen Stelle platziert. Der Roboter wird dann so bewegt, dass die Kamera den Kalibrierkörper sieht. Nun wird ein Bild des Kalibrierkörpers aufgenommen und die aktuelle Roboterpose abgefragt (*ToolInBasePose1*). Anhand des Bildes kann die Pose des Kalibrierkörpers im Kamerakoordinatensystem ermittelt werden (*ObjInCamPose1*). Anschließend wird der Greifpunkt des Roboters von Hand zum Ursprung des Kalibrierkörpers bewegt und erneut die Roboterpose abgefragt (*ToolInBasePose2*). Diese drei Posen können zusammen mit dem Ergebnis der Kalibrierung (*ToolInCamPose*) folgendermaßen verwendet werden, um die Mehrdeutigkeit in Z aufzulösen:

```
pose_invert (ToolInCamPose, CamInToolPose)
pose_compose (CamInToolPose, ObjInCamPose1, ObjInToolPose1)
pose_invert (ToolInBasePose1, BaseInToolPose1)
pose_compose (BaseInToolPose1, ToolInBasePose2, Tool2InTool1Pose)
ZCorrection := ObjInToolPose1[2]-Tool2InTool1Pose[2]
set_origin_pose (ToolInCamPose, 0, 0, ZCorrection, ToolInCamPoseFinal)
```

Stationäre Kamera

In dieser Konfiguration greift der Roboter den Kalibrierkörper und bewegt ihn vor der Kamera. Auch in dieser Konfiguration entspricht die aus einem Kalibrierbild extrahierte Information, d.h. die 3D-Lage des Kalibrierkörpers in

Kamerakoordinaten (d.h. die externen Kameraparameter) einer Kette von 3D-Lagen bzw. homogenen Transformationsmatrizen, diesmal vom Kalibrierkörper über das Roboter-Tool zu dessen Basis und schließlich zur Kamera:

Stationäre Kamera:

$$^{camera}\mathbf{H}_{cal} = {}^{camera}\mathbf{H}_{base} \cdot {}^{base}\mathbf{H}_{tool} \cdot {}^{tool}\mathbf{H}_{cal}$$

'obj_in_cam_pose' 'base_in_cam_pose' 'tool_in_base_pose' 'obj_in_tool_pose'

Analog zur Konfiguration mit bewegter Kamera bestimmt der Operator `calibrate_hand_eye` die zwei Transformationen an den Enden der Kette, hier nun die 3D-Lage der Roboter-Basis in Kamerakoordinaten ($^{camera}\mathbf{H}_{base}$, *'base_in_cam_pose'*) und die 3D-Lage des Kalibrierkörpers in Roboter-Toolkoordinaten ($^{tool}\mathbf{H}_{cal}$, *'obj_in_tool_pose'*).

Die Transformation in der Mitte der Kette, $^{base}\mathbf{H}_{tool}$, beschreibt die 3D-Lage des den Kalibrierkörper bewegenden Roboters, d.h. die 3D-Lage des Roboter-Tools in Roboter-Basiskoordinaten. Die Transformation $^{camera}\mathbf{H}_{cal}$ beschreibt die Pose des Kalibrierkörpers relativ zum Kamerakoordinatensystem.

Es ist zu beachten, dass die Z-Translation von *'obj_in_tool_pose'* im Rahmen der Kalibrierung von SCARA-Robotern nicht eindeutig bestimmt werden kann. Um diese Mehrdeutigkeit aufzulösen, wird die Z-Translation von *'obj_in_tool_pose'* intern auf 0.0 gesetzt und *'base_in_cam_pose'* entsprechend berechnet. Es ist notwendig, die tatsächliche Translation in Z nach der Kalibrierung zu bestimmen indem der Roboter an eine Pose bekannter Höhe im Kamerakoordinatensystem gefahren wird. Hierfür empfiehlt sich das folgende Vorgehen: Ein (nicht am Roboter befestigter) Kalibrierkörper wird an einer beliebigen Stelle so platziert, dass er von der Kamera gesehen wird. Anhand eines aufgenommenen Bildes muss dann die Pose des Kalibrierkörpers im Kamerakoordinatensystem ermittelt werden (*ObjInCamPose*). Anschließend wird der Greifpunkt des Roboters von Hand zum Ursprung des Kalibrierkörpers bewegt und die Roboterpose abgefragt (*ToolInBasePose*). Die beiden Posen können zusammen mit dem Ergebnis der Kalibrierung (*BaseInCamPose*) folgendermaßen verwendet werden, um die Mehrdeutigkeit in Z aufzulösen:

```
pose_invert (BaseInCamPose, CamInBasePose)
pose_compose (CamInBasePose, ObjInCamPose, ObjInBasePose)
ZCorrection := ObjInBasePose[2]-ToolInBasePose[2]
set_origin_pose (BaseInCamPose, 0, 0, ZCorrection, BaseInCamPoseFinal)
```

Vorbereitung der Eingabekalibrierdaten

Bevor der Operator `calibrate_hand_eye` aufgerufen wird, müssen die benötigten Daten im Kalibrierdatenmodell gesammelt werden. Dabei müssen diese Schritte befolgt werden:

1. **Erstellen des Kalibrierdatenmodells** mit dem Operator `create_calib_data`, wobei die Anzahl der Kameras und die Anzahl der Kalibrierkörper im Modell festgelegt werden. Abhängig vom Szenario muss für `CalibSetup` der Wert *'hand_eye_moving_camera'*, *'hand_eye_stationary_camera'*, *'hand_eye_scara_moving_camera'* oder *'hand_eye_scara_stationary_camera'* gesetzt werden. Diese vier Szenarien unterscheiden zum einen, ob die Kamera oder der Kalibrierkörper vom Roboter bewegt wird und zum anderen, ob ein anthropomorpher oder ein SCARA-Roboter kalibriert wird. Der Arm eines anthropomorphen Roboters hat 3 Drehgelenke, die typischerweise 6 Freiheitsgrade abdecken (3 Translationen und 3 Rotationen). SCARA-Roboter besitzen dagegen nur zwei parallele Drehgelenke und ein paralleles Schubgelenk, die nur 4 Freiheitsgrade (3 Translationen und 1 Rotation) abdecken. Grob gesagt ist ein anthropomorpher Roboter im Gegensatz zum SCARA-Roboter in der Lage, seinen Endeffektor zu verkippen.
2. **Festlegen der Optimierungsmethode** mit dem Operator `set_calib_data`. Für den Parameter `DataName`=*'optimization_method'* sind zwei Optionen für `DataValue` möglich, `DataValue`=*'nonlinear'* und `DataValue`=*'linear'* (siehe Abschnitt 'Die eigentliche Kalibrierung durchführen').
3. **Setzen der Posen des Kalibrierkörpers**
   (a) Jede beobachtete 3D-Pose des Kalibrierkörpers kann direkt mittels des Operators `set_calib_data_observ_pose` gesetzt werden. Somit ist es in der Hand-Auge-Kalibrierung möglich generische 3D-Sensoren zu benutzen, die ein Kalibrierobjekt beobachten.

(b) Andererseits kann die Pose des Kalibrierkörpers mittels Kamerabildern bestimmt werden. Dafür muss der Kalibrierkörper im Kalibrierdatenmodell `CalibDataID` mit dem Operator `set_calib_data_calib_object` gesetzt werden. Initiale Kameraparameter werden mit dem Operator `set_calib_data_cam_param` gesetzt. Wenn eine Standard-Kalibrierplatte verwendet wird, bestimmt der Operator `find_calib_object` die Pose des Kalibrierkörpers relativ zur Kamera und speichert sie im Kalibrierdatenmodell `CalibDataID`. Für anthropomorphe (d.h. nicht-SCARA) Roboter kalibriert der Operator `calibrate_hand_eye` in diesem Fall die Kamera bevor die eigentliche Hand-Auge-Kalibrierung durchgeführt wird. Falls die gegebenen Kameraparameter schon kalibriert sind, kann die Kamerakalibrierung wie folgt ausgeschaltet werden.

```
set_calib_data(CalibDataID,'camera',0,'general',
               'excluded_settings','params').
```

Im Gegensatz dazu geht `calibrate_hand_eye` bei SCARA-Robotern immer davon aus, dass die gegebenen Kameraparameter bereits kalibriert sind. Daher wird in diesem Fall vor der Hand-Auge-Kalibrierung keine automatische interne Kalibrierung der Kamera durchgeführt. Der Grund hierfür ist, dass die Kamerakalibrierung nur mit unterschiedlicher Verkippung der Kalibrierplatte zuverlässig ist. Bei der Hand-Auge-Kalibrierung ist die Kalibrierplatte oft annähernd parallel zur Bildebene, was bei einem SCARA-Roboter dazu führt, dass alle Kameraposen annähernd parallel sind. Daher muss die Kamera zuvor mit einem separaten Satz an Kalibrierbildern kalibriert worden sein.

4. **Spezifizieren der Posen des Tools** im Roboterbasiskoordinatensystem. Für jede Pose des Kalibrierkörpers im Kamerakoordinatensystem muss die korrespondierende Pose des Tools im Roboterbasiskoordinatensystem wie folgt gesetzt werden.

```
set_calib_data(CalibDataID,'tool', PoseNumber, 'tool_in_base_pose',
               ToolInBasePose)
```

Die eigentliche Kalibrierung durchführen

Der Operator `calibrate_hand_eye` kann die Kalibrierung auf zwei verschiedene Arten durchführen. In beiden Fällen werden alle gegebenen Posen des Kalibrierobjekts in Kamerakoordinaten sowie die zugehörigen Posen des Tools in Roboterbasiskoordinaten für die Kalibrierung benutzt. Mittels des Operators `set_calib_data` wird spezifiziert welche Methode benutzt wird.

Soll die Kalibrierung mittels eines linearen Algorithmus durchgeführt werden, so muss die Parameterkombination `DataName`=*'optimization_method'* und `DataValue`=*'linear'* gewählt werden. Dieser Algorithmus ist schnell jedoch in vielen praktischen Fällen nicht genau genug.

Wird die Parameterkombination `DataName`=*'optimization_method'* und `DataValue`=*'nonlinear'* gewählt, so wird ein nicht linearer Algorithmus benutzt, welcher das genaueste Ergebnis liefert und daher auch empfohlen ist.

Ergebnisse der Kalibrierung überprüfen

Nach einer erfolgreichen Kalibrierung wird der Lagefehler über die komplette Transformationskette in `Errors` zurückgegeben. Genau genommen wird ein Tupel mit vier Elementen zurückgegeben, welches zuerst den mittleren, quadratischen Fehler der Translation, dann den mittleren, quadratischen Fehler der Rotation, an dritter Stelle den maximalen Fehler der Translation und schließlich an vierter Stelle den maximalen Fehler der Rotation angibt. Diese Fehler geben einen allgemeinen Hinweis darauf, ob die Optimierung erfolgreich war.

Die Fehler werden in den Einheiten zurückgegeben, in denen die Eingabeposen gegeben waren. Das bedeutet, dass die Translationsfehler typischerweise in Metern zurückgegeben werden. Die Rotationsfehler werden immer in Grad zurückgegeben.

Kalibrierergebnisse abfragen

Die Posen, die durch den Operator `calibrate_hand_eye` berechnet werden, können mit `get_calib_data` abgefragt werden. Für das Szenario „bewegte Kamera“ können die 3D-Pose des Tools in Kamerakoordinaten (*'tool_in_cam_pose'*) und die 3D-Pose des Kalibrierkörpers im Roboterbasiskoordinatensystem (*'obj_in_base_pose'*) abgefragt werden. Für das Szenario „stationäre Kamera“ können die 3D-Pose der Roboterbasis im Kamerakoordinatensystem (*'base_in_cam_pose'*) und die 3D-Pose des Kalibrierkörpers im Koordinatensystem des Tools (*'obj_in_tool_pose'*) abgefragt werden.

Abfragen der Eingabedaten

Wurden die 3D-Posen des Kalibrierobjektes mit `find_calib_object` bestimmt, dann werden diese bei anthropomorphen (d.h. nicht-SCARA) Robotern dazu verwendet, die Kamera vor der eigentlichen Hand-Auge-Kalibrierung zu kalibrieren, und werden somit ebenfalls kalibriert. Die kalibrierten Posen können mit `get_calib_data` und dem Parameter `ItemType`=*'calib_obj_pose'* abgefragt werden. Die beobachteten 3D-Posen des Kalibrierobjektes im Kamerakoordinatensystem können mit `get_calib_data_observ_pose`

abgefragt werden. Die zugehörigen 3D-Posen des Tools im Roboterbasiskoordinatensystem können mit get_calib_data abgefragt werden.

Ein Set geeigneter Posen des Kalibrierkörpers aufnehmen

Ist eine Standard-Kalibrierplatte vorhanden, sollte folgendermaßen vorgegangen werden:

- Die Position des Kalibrierkörpers (bewegte Kamera: relativ zur Roboter-Basis; stationäre Kamera: relativ zum Roboter-Tool) und die Position der Kamera (bewegte Kamera: relativ zum Roboter-Tool; stationäre Kamera: relativ zur Roboter-Basis) bleiben während der Aufnahmen unverändert.
- Die theoretisch minimale Anzahl von Posen des Kalibrierobjektes ist drei. Aber, es wird empfohlen mindestens 10 Posen aufzunehmen, bei denen die Position der Kamera bzw. die Position der Roboterhand möglichst stark variiert.

  Für anthropomorphe (d.h. nicht-SCARA) Roboter ist ein ausreichendes Ausmaß an Rotation zwischen den Bildern notwendig und sollte mindestens 30 Grad besser aber 60 Grad betragen. Die Rotationen zwischen den Posen des Kalibrierobjektes müssen mindestens zwei verschiedene Rotationsachsen aufweisen. Sehr verschiedene Orientierungen führen zu präziseren Ergebnissen in der Hand-Auge-Kalibrierung. SCARA-Roboter weisen nur eine Rotationsachse auf. Die Rotation zwischen den Bildern sollte ebenfalls groß sein.
- Wird eine Kamera benutzt, so müssen die internen Kameraparameter während und nach der Kalibrierung konstant sein. Beachten Sie, dass Veränderungen der Bildgröße, der Brennweite, der Blende oder des Fokus eine Veränderung der internen Kameraparameter bedeuten.
- Die Kamera darf zwischen den Aufnahmen nicht verändert werden, d.h. weder Brennweite und Blende, noch Fokussierung darf verändert werden, denn für alle Aufnahmen gelten die gleichen internen Kameraparameter. Achten Sie darauf, dass bei einer Änderung des Abstands der Kamera vom Kalibrierkörper durch die Roboterbewegung die Fokussierung (Schärfe im Bild) noch ausreicht. Sorgen Sie daher bei den Aufnahmen für eine gute Ausleuchtung der Kalibrierplatte, denn die Tiefenschärfe nimmt mit kleiner Blende zu.

Lagen des Roboter-Tools aufnehmen

Wir empfehlen, die 3D-Lagen in einem separaten Programm zu erzeugen und mittels write_pose in Dateien abzuspeichern. Im Kalibrierprogramm können sie dann eingelesen werden und im Kalibrierdatenmodell gesetzt werden.

Die 3D-Lage eines Roboters kann über seine kartesische Schnittstelle typischerweise in einer Notation abgefragt werden, die den Darstellungstypen mit den Codes 0 oder 2 entsprechen (OrderOfRotation = *'gba'* oder *'abg'*, siehe create_pose). In diesem Fall können die abgefragten Werte direkt als Eingabe für create_pose verwendet werden.

Falls die kartesische Schnittstelle des Roboters die Orientierung anders beschreibt, z.B. in der Darstellung ZYZ ($\mathbf{R}_z(\varphi 1) \cdot \mathbf{R}_y(\varphi 2) \cdot \mathbf{R}_z(\varphi 3)$), kann die entsprechende homogene Transformationsmatrix schrittweise mit den Operatoren hom_mat3d_rotate und hom_mat3d_translate aufgebaut und dann mit hom_mat3d_to_pose in eine 3D-Lage gewandelt werden. Der folgende Beispiel-Code erzeugt eine 3D-Lage aus der oben beschriebenen ZYZ-Darstellung:

```
hom_mat3d_identity (HomMat3DIdent)
hom_mat3d_rotate (HomMat3DIdent, φ3, 'z', 0, 0, 0, HomMat3DRotZ)
hom_mat3d_rotate (HomMat3DRotZ, φ2, 'y', 0, 0, 0, HomMat3DRotYZ)
hom_mat3d_rotate (HomMat3DRotYZ, φ1, 'z', 0, 0, 0, HomMat3DRotZYZ)
hom_mat3d_translate (HomMat3DRotZYZ, Tx, Ty, Tz, base_H_tool)
hom_mat3d_to_pose (base_H_tool, RobPose)
```

Bitte beachten Sie, dass die Hand-Auge-Kalibrierung nur dann erfolgreich sein kann, wenn die Posen des Tools im Roboterbasiskoordinatensystem mit hoher Genauigkeit angegeben werden!

Parameter

▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **Errors** (output_control) . . . . . number-array ⇝ *real*
Durchschnittlicher verbleibender Fehler nach der Optimierung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_calib_data`, `set_calib_data_cam_param`, `set_calib_data_calib_object`, `set_calib_data_observ_pose`, `find_calib_object`, `set_calib_data`, `remove_calib_data`, `remove_calib_data_observ`

Nachfolger

`get_calib_data`

Literatur

K. Daniilidis: „Hand-Eye Calibration Using Dual Quaternions“; International Journal of Robotics Research, Vol. 18, No. 3, pp. 286-298; 1999.

M. Ulrich, C. Steger: „Hand-Eye Calibration of SCARA Robots Using Dual Quaternions“; Pattern Recognition and Image Analysis, Vol. 26, No. 1, pp. 231-239; January 2016.

Modul

Calibration

get_calib_data_observ_pose (: : CalibDataID, CameraIdx, CalibObjIdx, CalibObjPoseIdx : ObjInCameraPose)

Abfragen der beobachteten Posen eines Kalibrierobjektes aus einem Kalibrierdatenmodell.

Der Operator `get_calib_data_observ_pose` liest die beobachteten Posen des Kalibrierobjektes relativ zum Kamerakoordinatensystem aus einem Kalibrierdatenmodell `CalibDataID`. Die beobachteten Posen müssen vorher mittels des Operators `set_calib_data_observ_pose` gesetzt werden (siehe `set_calib_data_observ_pose` für eine detaillierte Beschreibung der Argumente). Falls das Kalibrierobjekt mit einer Kamera beobachtet wurde, wurde seine Pose durch den Operator `find_calib_object` im Kalibrierdatenmodell `CalibDataID` gespeichert.

Parameter

- ▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
  Handle des Kalibrierdatenmodells.
- ▷ **CameraIdx** (input_control) . . . . . number ⇝ *integer*
  Index der beobachtenden Kamera.
  Defaultwert: 0
- ▷ **CalibObjIdx** (input_control) . . . . . number ⇝ *integer*
  Index des beobachteten Kalibrierobjekts.
  Defaultwert: 0
- ▷ **CalibObjPoseIdx** (input_control) . . . . . number ⇝ *integer*
  Index der Pose des beobachteten Kalibrierobjekts.
  Defaultwert: 0
- ▷ **ObjInCameraPose** (output_control) . . . . . pose ⇝ *real* / integer
  Gespeicherte beobachtete Poses des Kalibrierobjekts relativ zur beobachtenden Kamera.
  Parameteranzahl: 7

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Calibration

hand_eye_calibration (: : X, Y, Z, Row, Col, NumPoints, RobotPoses, CameraParam, Method, QualityType : CameraPose, CalibrationPose, Quality)

Durchführen einer Hand-Auge-Kalibrierung.

Der Operator hand_eye_calibration bestimmt die 3D-Lage eines Roboters („Hand“) relativ zu einer Kamera („Auge“). Mit dieser Information können die Ergebnisse der Bildverarbeitung in das Koordinatensystem des Roboters transformiert werden, der dann z.B. ein inspiziertes Teil greift. Bitte beachten Sie, dass der Operator hand_eye_calibration keine 3D-Sensoren unterstützt. Eine Hand-Auge-Kalibrierung mit 3D-Sensoren kann nur mit dem Operator calibrate_hand_eye vorgenommen werden. Jener Operator stellt des Weiteren eine benutzerfreundlichere Schnittstelle für die Hand-Auge-Kalibrierung bereit als der hier vorliegende Operator hand_eye_calibration, da die jeweiligen Bezugssysteme klar ausgezeichnet sind.

Es gibt zwei mögliche Konfigurationen von Roboter-Kamera-Systemen (Hand-Auge-Systemen): Die Kamera kann auf dem Roboter montiert sein oder stationär sein und den Roboter beobachten. Beachten Sie, dass der Begriff „Roboter“ für ein beliebiges System steht, das Objekte bewegt. Daher können mit hand_eye_calibration viele verschiedene Systeme kalibriert werden, von Schwenk-Neige-Köpfen zu Vielfachs-Manipulatoren.

Eine Hand-Auge-Kalibrierung kann über eine geschlossene Verkettung von insgesamt vier euklidischen Transformationen beschrieben werden. In dieser Kette sind zwei nicht aufeinander folgende Transformation bekannt, welche von der Robotersteuerung und der Kalibrierung der Marker stammen. Die beiden anderen Transformation sind unbekannt aber konstant und werden durch die Hand-Auge-Kalibrierung bestimmt.

Eine Hand-Auge-Kalibrierung läuft ähnlich ab wie die Kalibrierung der externen Kameraparameter (siehe camera_calibration): Zuerst werden ein Satz Bilder eines Kalibrierkörpers aufgenommen, in diesen werden dann zu bekannten Kalibrierpunkten korrespondierende Bildpunkte gesucht und dem Operator hand_eye_calibration in den Parametern X, Y, Z, Row, Col und NumPoints übergeben. Falls die Standard-Kalibrierplatte verwendet wird, können die Korrespondenzen sehr einfach mit den Operatoren find_caltab und find_marks_and_pose bestimmt werden. Es ist zu beachten, dass die internen Parameter der verwendeten Kamera vorher zu kalibrieren sind.

Im Gegensatz zur Kamerakalibrierung wird der Kalibrierkörper nicht manuell bewegt. Dies ist die Aufgabe des Roboters, der entweder die Kamera bewegt (bei auf dem Roboter montierter Kamera) oder den Kalibrierkörper (bei stationärer Kamera). Die Bewegungen des Roboters werden als bekannt vorausgesetzt und daher ebenfalls als Eingabe für die Kalibrierung benutzt (Parameter RobotPoses).

Im Folgenden werden die zwei Hand-Auge-Konfigurationen detaillierter beschrieben, gefolgt von genereller Information über den Prozess der Hand-Auge-Kalibrierung.

Bewegte Kamera (auf dem Roboter montiert)

In dieser Konfiguration ist der Kalibrierkörper stationär, und die Kamera wird vom Roboter an verschiedene Positionen bewegt. Die grundlegende Idee hinter der Hand-Auge-Kalibrierung ist, dass die aus einem Kalibrierbild extrahierte Information, d.h. die 3D-Lage des Kalibrierkörpers relativ zur Kamera (d.h. die externen Kameraparameter), als Kette von 3D-Lagen bzw. homogenen Transformationsmatrizen aufgefasst werden kann, vom Kalibrierkörper über die Roboter-Basis zu dessen Tool (Endeffektor) und schließlich zur Kamera:

Bewegte Kamera:

$$ {}^{cam}\mathbf{H}_{cal} = {}^{cam}\mathbf{H}_{tool} \cdot {}^{tool}\mathbf{H}_{base} \cdot {}^{base}\mathbf{H}_{cal} $$

CameraPose RobotPoses CalibrationPose

Aus den Kalibrierbildern bestimmt der Operator hand_eye_calibration die zwei Transformationen an den Enden der Kette, d.h. die 3D-Lage des Roboter-Tools in Kamerakoordinaten (${}^{cam}\mathbf{H}_{tool}$,CameraPose) und die 3D-Lage des Kalibrierkörpers in Roboter-Basiskoordinaten (${}^{base}\mathbf{H}_{cal}$,CalibrationPose).

Im Gegensatz dazu ist die Transformation in der Mitte der Kette, ${}^{tool}\mathbf{H}_{base}$, bekannt, aber unterschiedlich für jedes Kalibrierbild, da sie die 3D-Lage des die Kamera bewegenden Roboters beschreibt, genauer gesagt deren

Inverse (3D-Lage der Roboter-Basis in Roboter-Toolkoordinaten). Die (inversen) 3D-Lagen des Roboters in den Kalibrierbildern müssen im Parameter RobotPoses übergeben werden.

Es ist zu beachten, dass die Z-Translation von CalibrationPose im Rahmen der Kalibrierung von SCARA-Robotern nicht eindeutig bestimmt werden kann. Um diese Mehrdeutigkeit aufzulösen, wird die Z-Translation von CalibrationPose intern auf 0.0 gesetzt und CameraPose entsprechend berechnet. Es ist notwendig, die tatsächliche Translation in Z nach der Kalibrierung zu bestimmen (siehe calibrate_hand_eye).

Stationäre Kamera

In dieser Konfiguration greift der Roboter den Kalibrierkörper und bewegt ihn vor der Kamera. Auch in dieser Konfiguration entspricht die aus einem Kalibrierbild extrahierte Information, d.h. die 3D-Lage des Kalibrierkörpers in Kamerakoordinaten (d.h. die externen Kameraparameter) einer Kette von 3D-Lagen bzw. homogenen Transformationsmatrizen, diesmal vom Kalibrierkörper über das Roboter-Tool zu dessen Basis und schließlich zur Kamera:

Stationäre Kamera:

$$ {}^{cam}\mathbf{H}_{cal} = {}^{cam}\mathbf{H}_{base} \cdot {}^{base}\mathbf{H}_{tool} \cdot {}^{tool}\mathbf{H}_{cal} $$

CameraPose RobotPoses CalibrationPose

Analog zur Konfiguration mit bewegter Kamera bestimmt der Operator hand_eye_calibration die zwei Transformationen an den Enden der Kette, hier nun die 3D-Lage der Roboter-Basis in Kamerakoordinaten (${}^{cam}\mathbf{H}_{base}$,CameraPose) und die 3D-Lage des Kalibrierkörpers in Roboter-Toolkoordinaten (${}^{tool}\mathbf{H}_{cal}$,CalibrationPose).

Die Transformation in der Mitte der Kette, ${}^{base}\mathbf{H}_{tool}$, beschreibt die 3D-Lage des den Kalibrierkörper bewegenden Roboters, d.h. die 3D-Lage des Roboter-Tools in Roboter-Basiskoordinaten. Die 3D-Lagen des Roboters in den Kalibrierbildern müssen im Parameter RobotPoses übergeben werden.

Es ist zu beachten, dass die Z-Translation von CalibrationPose im Rahmen der Kalibrierung von SCARA-Robotern nicht eindeutig bestimmt werden kann. Um diese Mehrdeutigkeit aufzulösen, wird die Z-Translation von CalibrationPose intern auf 0.0 gesetzt und CameraPose entsprechend berechnet. Es ist notwendig, die tatsächliche Translation in Z nach der Kalibrierung zu bestimmen (siehe calibrate_hand_eye).

Zusätzliche Information über den Kalibrierprozess

Die folgenden Abschnitte befassen sich jeweils mit einzelnen Fragen, die sich bei der Verwendung von hand_eye_calibration stellen und sollen daher sowohl dem Verständnis als auch als Leitfaden für die eigene Anwendung dienen.

Wie erhalte ich die 3D-Kalibrierpunkte? Als Grundlage für die Hand-Auge-Kalibrierung dienen 3D-Kalibrierpunkte im Weltkoordinatensystem (X, Y, Y) und ihre zugehörigen Abbildungen im Bildkoordinatensystem (Row, Col). Für eine erfolgreiche Hand-Auge-Kalibrierung müssen möglichst viele Bilder der 3D-Kalibrierpunkte aus möglichst unterschiedlichen Roboterpositionen aufgenommen werden.

Prinzipiell können beliebige bekannte Punkte im Weltkoordinatensystem verwendet werden. Man muss lediglich die Projektionen im Bild kennen. Idealerweise verwendet man jedoch eine Standard-Kalibrierplatte, wie sie mit gen_caltab erzeugt werden kann. Dann können mit find_caltab und find_marks_and_pose die Positionen der Kalibrierplatte und ihrer Marken extrahiert werden; der Operator caltab_points liefert die 3D-Koordinaten der Kalibriermarken (siehe auch die Beschreibung von camera_calibration).

Mit dem Parameter NumPoints wird angegeben, wie viele 3D-Kalibrierpunkte für jede Lage des Roboters und damit für jedes Kalibrierbild verwendet werden. Damit ist die Zuordnung der in X, Y, Y linearisiert angegebenen 3D-Kalibrierpunkte und ihrer Projektionen (Row, Col) zu der entsprechenden 3D-Lage des Roboters (RobotPoses) möglich. Beachten Sie, dass im Gegensatz zu camera_calibration die 3D-Koordinaten der Kalibrierpunkte für jedes Kalibrierbild angegeben werden müssen und nicht nur einmal!

Wie nehme ich eine Menge von geeigneten Bildern auf? Ist eine Standard-Kalibrierplatte vorhanden, sollte folgendermaßen vorgegangen werden:

- Die Position des Kalibrierkörpers (bewegte Kamera: relativ zur Roboter-Basis; stationäre Kamera: relativ zum Roboter-Tool) und die Position der Kamera (bewegte Kamera: relativ zum Roboter-Tool; stationäre Kamera: relativ zur Roboter-Basis) bleiben während der Aufnahmen unverändert.

- Die internen Kameraparameter sind konstant und müssen zuvor bestimmt sein und in CameraParam übergeben werden (vgl. camera_calibration). Beachten Sie, dass Veränderungen der Bildgröße, der Brennweite, der Blende oder des Fokus eine Veränderung der internen Kameraparameter bedeuten.
- Die theoretisch minimale Anzahl der aufzunehmenden Bilder ist drei. Aber, es wird empfohlen mindestens 10 Bilder aufzunehmen, bei denen die Position der Kamera bzw. die Position der Roboterhand möglichst stark variiert.
  Für konventionelle (d.h. nicht-SCARA) Roboter ist ein ausreichendes Ausmaß an Rotation zwischen den Bildern notwendig und sollte mindestens 30 Grad besser aber 60 Grad betragen. Die Rotationen zwischen den Posen des Kalibrierobjektes müssen mindestens zwei verschiedene Rotationsachsen aufweisen. Sehr verschiedene Orientierungen führen zu präziseren Ergebnissen in der Hand-Auge-Kalibrierung. SCARA-Roboter weisen nur eine Rotationsachse auf. Die Rotation zwischen den Bildern sollte ebenfalls groß sein.
- Die Kalibrierplatte muss jeweils komplett (inkl. Rand!) im Bild sichtbar sein.
- Es sollten möglichst keine Reflexionen oder ähnliche Störungen auf der Kalibrierplatte zu sehen sein.
- Werden individuelle Marker anstelle der Kalibrierplatte verwendet, so ist die minimale Anzahl der Marker in jedem Bild vier.
- Die Kalibrierplatte sollte jeweils mindestens etwa ein Viertel des gesamten Bildes ausfüllen, damit die Marken robust detektiert werden können.
- Die Kamera darf zwischen den Aufnahmen nicht verändert werden, d.h. weder Brennweite und Blende, noch Fokussierung darf verändert werden, denn für alle Aufnahmen gelten die gleichen internen Kameraparameter. Achten Sie darauf, dass bei einer Änderung des Abstands der Kamera vom Kalibrierkörper durch die Roboterbewegung die Fokussierung (Schärfe im Bild) noch ausreicht. Sorgen Sie daher bei den Aufnahmen für eine gute Ausleuchtung der Kalibrierplatte, denn die Tiefenschärfe nimmt mit kleiner Blende zu.

Wie erhalte ich die Lagen der Bewegungseinheit? Im Parameter RobotPoses müssen die 3D-Lagen des Roboters in den Kalibrierbildern (bewegte Kamera: 3D-Lage der Roboter-Basis in Roboter-Toolkoordinaten; stationäre Kamera: 3D-Lage des Roboter-Tools in Roboter-Basiskoordinaten) übergeben werden. Wir empfehlen, die 3D-Lagen in einem separaten Programm zu erzeugen und mittels write_pose in Dateien abzuspeichern. Im Kalibrierprogramm können sie dann eingelesen und in einem Tupel aufgesammelt werden, wie das Beispiel weiter unten zeigt. Außerdem empfehlen wir, in den Dateien unabhängig von der verwendeten Hand-Auge-Koordination die 3D-Lage des Roboter-Tools in Roboter-Basiskoordinaten zu speichern. Beim Einsatz einer bewegten Kamera erfolgt die Invertierung der eingelesenen 3D-Lagen dann vor dem Einfügen in das Tupel. Auch dies ist im Beispielprogramm beschrieben.

Die 3D-Lage eines Roboters kann über seine kartesische Schnittstelle typischerweise in einer Notation abgefragt werden, die den Darstellungstypen mit den Codes 0 oder 2 entsprechen (OrderOfRotation = *'gba'* oder *'abg'*, siehe create_pose). In diesem Fall können die abgefragten Werte direkt als Eingabe für create_pose verwendet werden.

Falls die kartesische Schnittstelle des Roboters die Orientierung anders beschreibt, z.B. in der Darstellung ZYZ ($\mathbf{R}_z(\varphi 1) \cdot \mathbf{R}_y(\varphi 2) \cdot \mathbf{R}_z(\varphi 3)$), kann die entsprechende homogene Transformationsmatrix schrittweise mit den Operatoren hom_mat3d_rotate und hom_mat3d_translate aufgebaut und dann mit hom_mat3d_to_pose in eine 3D-Lage gewandelt werden. Der folgende Beispiel-Code erzeugt eine 3D-Lage aus der oben beschriebenen ZYZ-Darstellung:

```
hom_mat3d_identity (HomMat3DIdent)
hom_mat3d_rotate (HomMat3DIdent, φ3, 'z', 0, 0, 0, HomMat3DRotZ)
hom_mat3d_rotate (HomMat3DRotZ, φ2, 'y', 0, 0, 0, HomMat3DRotYZ)
hom_mat3d_rotate (HomMat3DRotYZ, φ1, 'z', 0, 0, 0, HomMat3DRotZYZ)
hom_mat3d_translate (HomMat3DRotZYZ, Tx, Ty, Tz, base_H_tool)
hom_mat3d_to_pose (base_H_tool, RobPose)
```

Bitte beachten Sie, dass die Hand-Auge-Kalibrierung nur dann erfolgreich sein kann, wenn die Roboterpositionen RobotPoses mit hoher Genauigkeit angegeben werden!

Wie sieht die Reihenfolge der einzelnen Parameter aus? Die Länge des Tupels NumPoints entspricht der Anzahl der verschiedenen Positionen des Roboters und damit der Anzahl der Kalibrierbilder. Mit NumPoints wird angegeben, wie viele Kalibrierpunkte pro Bild verwendet werden. Wird die Standard-Kalibrierplatte verwendet, so sind dies 49 Punkte pro Bild. Wurden 15 Aufnahmen gemacht so ist NumPoints ein Tupel der Länge 15, wobei alle Einträge die Zahl 49 enthalten.

Die Anzahl der Kalibrierbilder muss auch bei den Tupeln mit den Koordinaten der 3D-Kalibriermarkenpunkte bzw. der extrahierten 2D-Markenpunkte berücksichtigt werden. Bei einer Anzahl von 15 Kalibrierbildern mit

jeweils 49 Markenpunkte müssen daher die Tupel `Row`, `Col`, `X`, `Y` und `Z` $15 \cdot 49 = 735$ Werte enthalten. Die Reihenfolge der Werte erfolgt dabei bildweise. d.h. die ersten 49 Werte gehören zu den 49 Marken im ersten Bild. Die Reihenfolge der 3D-Markenpunkte und der 2D-Markenpunkte muss in jedem Bild gleich sein.

Die Länge des Tupels `RobotPoses` ist ebenfalls abhängig von der Anzahl der Kalibrierbilder. Werden beispielsweise 15 Aufnahmen und damit 15 3D-Lagen verwendet, so ergibt sich die Länge des Tupels `RobotPoses` zu 15*7=105 (15 mal je 7 Lageparameter). Die ersten 7 Werte entsprechen daher der 3D-Lage des Roboters bei der ersten Aufnahme, die nächsten 7 denen bei der zweiten Aufnahme usw.

Algorithmen und Ausgabeparameter Der Parameter `Method` bestimmt den zugrunde liegenden Algorithmus der Hand-Auge-Kalibrierung: Mit *'linear'* wird ein linearer Algorithmus ausgewählt, welcher schnell aber in praktischen Fällen nicht besonders genau ist. Mit *'nonlinear'* wird ein nicht linearer Algorithmus gewählt, welcher das genaueste Ergebnis liefert und daher auch empfohlen ist.

Für die Kalibrierung von SCARA-Robotern muss für `Method` *'scara_linear'* oder *'scara_nonlinear'* gewählt werden. Während der Arm eines anthropomorphen Roboters 3 Drehgelenke hat, die typischerweise 6 Freiheitsgrade abdecken (3 Translationen und 3 Rotationen) besitzen SCARA-Roboter dagegen nur zwei parallele Drehgelenke und ein paralleles Schubgelenk, die nur 4 Freiheitsgrade (3 Translationen und 1 Rotation) abdecken. Grob gesagt ist ein anthropomorpher Roboter im Gegensatz zum SCARA-Roboter in der Lage, seinen Endeffektor zu verkippen.

Der Parameter `QualityType` schaltet zwischen unterschiedlichen Möglichkeiten die Qualität der Ergebnisse im Ausgabewert `Quality` zu beurteilen. Mit *'error_pose'* wird der Lagefehler über die komplette Transformationskette gewählt. Genaugenommen wird ein Tupel mit vier Elementen zurückgegeben, welches zuerst den mittleren, quadratischen Fehler der Translation, dann den mittleren, quadratischen Fehler der Rotation, an dritter Stelle den maximalen Fehler der Translation und schließlich an vierter Stelle den maximalen Fehler der Rotation angibt. Mit *'standard_deviation'* wird ein 12-elementiges Tupel mit den Standardabweichungen der beiden berechneten Lagen ausgegeben: Die ersten sechs Elemente beziehen sich auf die Kameralage und die zweiten sechs auf die Lage der Kalibriermarker. Bei der Wahl von *'covariance'* wird die volle 12x12 Kovarianzmatrix beider Lagen zurückgegeben. Wie die Poseparameter selbst, werden auch die Standardabweichungen und die Kovarianzen in den Einheiten [m] und [°] angegeben. Es ist zu beachten, dass bei der Wahl von *'linear'* oder *'scara_linear'* als Parameter `Method` nur die Qualitätsbewertung für den Lagefehler (*'error_pose'*) möglich ist.

Parameter

- ▷ **X** (input_control) ..... number-array ⇝ *real* / integer
  Linearisierte Liste, die alle x-Koordinaten der Kalibrierpunkte enthält (in der Bildreihenfolge).
- ▷ **Y** (input_control) ..... number-array ⇝ *real* / integer
  Linearisierte Liste, die alle y-Koordinaten der Kalibrierpunkte enthält (in der Bildreihenfolge).
- ▷ **Z** (input_control) ..... number-array ⇝ *real* / integer
  Linearisierte Liste, die alle z-Koordinaten der Kalibrierpunkte enthält (in der Bildreihenfolge).
- ▷ **Row** (input_control) ..... number-array ⇝ *real* / integer
  Linearisierte Liste, die alle Zeilenkoordinaten der Kalibrierpunkte enthält (in der Bildreihenfolge).
- ▷ **Col** (input_control) ..... number-array ⇝ *real* / integer
  Linearisierte Liste, die alle Spaltenkoordinaten der Kalibrierpunkte enthält (in der Bildreihenfolge).
- ▷ **NumPoints** (input_control) ..... integer-array ⇝ *integer*
  Für jedes Bild: Anzahl der Kalibrierpunkte.
- ▷ **RobotPoses** (input_control) ..... pose-array ⇝ *real* / integer
  Zu jedem Bild die bekannten Werte für die 3D-Lage des Roboters (bewegte Kamera: Roboter-Basis in Roboter-Toolkoordinaten; stationäre Kamera: Roboter-Tool in Roboter-Basiskoordinaten).
- ▷ **CameraParam** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **Method** (input_control) ..... string ⇝ *string*
  Methode der Hand-Auge-Kalibrierung.
  Defaultwert: 'nonlinear'
  Werteliste: `Method` ∈ {'linear', 'nonlinear', 'scara_linear', 'scara_nonlinear'}
- ▷ **QualityType** (input_control) ..... string(-array) ⇝ *string*
  Typ der Qualitätsbewertung.
  Defaultwert: 'error_pose'
  Werteliste: `QualityType` ∈ {'error_pose', 'standard_deviation', 'covariance'}

▷ **CameraPose** (output_control) . . . . . pose ⇝ *real* / integer
Berechnete relative Kamera Lage: 3D-Lage des Roboter-Tools (bewegte Kamera) bzw. der Roboter-Basis (stationäre Kamera) in Kamerakoordinaten.

▷ **CalibrationPose** (output_control) . . . . . pose ⇝ *real* / integer
Berechnete 3D-Lage der Kalibriermarker in Roboter-Basiskoordinaten (bewegte Kamera) bzw. Roboter-Toolkoordinaten (stationäre Kamera).

▷ **Quality** (output_control) . . . . . real(-array) ⇝ *real*
Qualitätsbewertung des Ergebnisses.

Beispiel

```
* Note that, in order to use this code snippet, you must provide
* the camera parameters, the calibration plate description file,
* the calibration images, and the robot poses.
read_cam_par('campar.dat', CameraParam)
CalDescr := 'caltab.descr'
caltab_points(CalDescr, X, Y, Z)
* Process all calibration images.
for i := 0 to NumImages-1 by 1
  read_image(Image, 'calib_'+i$'02d')
  * Find marks on the calibration plate in every image.
  find_caltab(Image, CalPlate, CalDescr, 3, 150, 5)
  find_marks_and_pose(Image, CalPlate, CalDescr, CameraParam, 128, 10, 18, \
                      0.9, 15, 100, RCoordTmp, CCoordTmp, StartPose)
  * Accumulate 2D and 3D coordinates of the marks.
  RCoord := [RCoord, RCoordTmp]
  CCoord := [CCoord, CCoordTmp]
  XCoord := [XCoord, X]
  YCoord := [YCoord, Y]
  ZCoord := [ZCoord, Z]
  NumMarker := [NumMarker, |RCoordTmp|]
  * Read pose of the robot tool in robot base coordinates.
  read_pose('robpose_'+i$'02d'+'.dat', RobPose)
  * Moving camera? Invert pose.
  if (IsMovingCameraConfig == 'true')
    pose_to_hom_mat3d(RobPose, base_H_tool)
    hom_mat3d_invert(base_H_tool, tool_H_base)
    hom_mat3d_to_pose(tool_H_base, RobPose)
  endif
  * Accumulate robot poses.
  RobotPoses := [RobotPoses, RobPose]
endfor
*
* Perform hand-eye calibration.
*
hand_eye_calibration(XCoord, YCoord, ZCoord, RCoord, CCoord, NumMarker, \
                     RobotPoses, CameraParam, 'nonlinear', 'error_pose', \
                     CameraPose, CalibrationPose, Error)
```

Ergebnis

`hand_eye_calibration` liefert den Wert H_MSG_TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

find_marks_and_pose, camera_calibration, calibrate_cameras

Nachfolger

write_pose, convert_pose_type, pose_to_hom_mat3d, disp_caltab, sim_caltab

Alternativen

calibrate_hand_eye

Siehe auch

find_caltab, find_marks_and_pose, disp_caltab, sim_caltab, write_cam_par, read_cam_par, create_pose, convert_pose_type, write_pose, read_pose, pose_to_hom_mat3d, hom_mat3d_to_pose, caltab_points, gen_caltab, calibrate_hand_eye

Literatur

K. Daniilidis: „Hand-Eye Calibration Using Dual Quaternions“; International Journal of Robotics Research, Vol. 18, No. 3, pp. 286-298; 1999.

M. Ulrich, C. Steger: „Hand-Eye Calibration of SCARA Robots Using Dual Quaternions“; Pattern Recognition and Image Analysis, Vol. 26, No. 1, pp. 231-239; January 2016.

Modul

Calibration

set_calib_data_observ_pose (: : CalibDataID, CameraIdx, CalibObjIdx, CalibObjPoseIdx, ObjInCameraPose :)

Setzen von beobachteten Posen eines Kalibrierobjekts in einem Kalibrierdatenmodell.

Für ein Kalibrierdatenmodell des Types CalibSetup=*'hand_eye_moving_cam'*, *'hand_eye_stationary_cam'*, *'hand_eye_scara_moving_cam'* oder *'hand_eye_scara_stationary_cam'* ohne Kalibrierobjekt (siehe create_calib_data) basiert die Hand-Auge-Kalibrierung auf Beobachtungen der Pose eines beliebigen Objekts in einem Kamerakoordinatensystem. Im Folgenden wird dieses Objekt als Kalibrierobjekt bezeichnet. Zusätzlich müssen die korrespondierenden Posen des Roboter-Tools im Roboterbasiskoordinatensystem bekannt sein. Mittels set_calib_data_observ_pose, wird die beobachtete Pose des Kalibrierobjektes im Kalibrierdatenmodell CalibDataID gespeichert. Eine Beobachtung der Pose besteht aus den folgenden Daten:

CameraIdx: Index der beobachtenden Kamera

CalibObjIdx: Index des beobachteten Kalibrierobjektes

CalibObjPoseIdx: Index der beobachteten Pose des Kalibrierobjekts. Der Index kann beliebig gewählt werden ohne einer strengen Ordnung zu folgen. Falls ein Index gewählt wird, der für das Kalibrierobjekt CalibObjIdx bereits existiert, so wird die entsprechende Pose durch die neue Pose ersetzt.

ObjInCameraPose: Pose des beobachteten Kalibrierobjekts relative zur beobachtenden Kamera.

Da das Modell für Kalibrierdaten CalibDataID einen allgemeinen Sensor ohne Kalibrierobjekt verwendet (d.h. das Modell wurde mit create_calib_data mit NumCameras=0 und NumCalibObjects=0 erzeugt), müssen sowohl CameraIdx als auch CalibObjIdx auf 0 gesetzt werden. Falls das Modell ein Kalibrierobjekt verwendet (d.h. NumCameras=1 und NumCalibObjects=1), muss stattdessen find_calib_object oder set_calib_data_observ_points verwendet werden.

Die beobachteten Posen können mittels des Operators get_calib_data_observ_pose abgefragt werden, wobei die gleichen Werte für die Argumente CameraIdx, CalibObjIdx und CalibObjPoseIdx zu setzen sind.

Parameter

▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **CameraIdx** (input_control) . . . . . number ⇝ *integer*
Index der beobachtenden Kamera.
Defaultwert: 0
Wertevorschläge: CameraIdx ∈ {0, 1, 2}

▷ **CalibObjIdx** (input_control) . . . . . number ⇝ *integer*
Index des beobachteten Kalibrierobjekts.
Defaultwert: 0
Wertevorschläge: CalibObjIdx ∈ {0, 1, 2}

▷ **CalibObjPoseIdx** (input_control) . . . . . number ⇝ *integer*
Index der Pose des beobachteten Kalibrierobjekts.
Defaultwert: 0
Wertevorschläge: CalibObjPoseIdx ∈ {0, 1, 2}
Restriktion: CalibObjPoseIdx >= 0

▷ **ObjInCameraPose** (input_control) . . . . . pose ⇝ *real* / integer
Gespeicherte beobachtete Poses des Kalibrierobjekts relativ zur beobachtenden Kamera.
Parameteranzahl: 7

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

find_marks_and_pose, set_calib_data_cam_param, set_calib_data_calib_object

Nachfolger

set_calib_data, calibrate_cameras

Alternativen

find_calib_object

Modul

Calibration

14.3 Inverse Projektion

get_line_of_sight (: : Row, Column, CameraParam : PX, PY, PZ, QX, QY, QZ)

Berechnen des zu einem Bildpunkt gehörenden Sichtstrahls.

get_line_of_sight berechnet für ein Pixel (Row, Column) im Bild den zugehörigen Sichtstrahl. Der Sichtstrahl ist eine Gerade im Kamerakoordinatensystem, die durch zwei auf ihr liegende Punkte (PX,PY,PZ) und (QX,QY,QZ) beschrieben wird. Die Kamera wird durch die internen Kameraparameter CameraParam beschrieben (siehe Kalibrierung). Falls eine Lochkamera verwendet wird, liegt der zweite Punkt in der Brennpunktebene, d.h. bei Flächenkameras entspricht der Ausgabeparameter QZ der Brennweite der Kamera, wohingegen QZ bei Zeilenkameras zusätzlich noch von der Relativbewegung zwischen Kamera und Objekt abhängt. Die Geradengleichung des Sichtstrahls ist gegeben durch

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} PX \\ PY \\ PZ \end{pmatrix} + l \cdot \begin{pmatrix} QX - PX \\ QY - PY \\ QZ - PZ \end{pmatrix}$$

Der Vorteil der Darstellung der Geraden durch zwei Punkte ist, dass es hiermit einfacher wird, die Gerade im Raum zu transformieren. Es muss lediglich der Operator affine_trans_point_3d auf die beiden Punkte angewandt werden.

Parameter

- ▷ **Row** (input_control) . . . . . real-array ⇝ *real*
  Zeilenkoordinate des Pixels.
- ▷ **Column** (input_control) . . . . . real-array ⇝ *real*
  Spaltenkoordinate des Pixels.
- ▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **PX** (output_control) . . . . . real-array ⇝ *real*
  X-Koordinate des ersten Punktes auf dem Sichtstrahl im Kamerakoordinatensystem.
- ▷ **PY** (output_control) . . . . . real-array ⇝ *real*
  Y-Koordinate des ersten Punktes auf dem Sichtstrahl im Kamerakoordinatensystem.
- ▷ **PZ** (output_control) . . . . . real-array ⇝ *real*
  Z-Koordinate des ersten Punktes auf dem Sichtstrahl im Kamerakoordinatensystem.
- ▷ **QX** (output_control) . . . . . real-array ⇝ *real*
  X-Koordinate des zweiten Punktes auf dem Sichtstrahl im Kamerakoordinatensystem.
- ▷ **QY** (output_control) . . . . . real-array ⇝ *real*
  Y-Koordinate des zweiten Punktes auf dem Sichtstrahl im Kamerakoordinatensystem.
- ▷ **QZ** (output_control) . . . . . real-array ⇝ *real*
  Z-Koordinate des zweiten Punktes auf dem Sichtstrahl im Kamerakoordinatensystem.

Beispiel

```
* Set internal camera parameters.
* Note the, typically, these values are the result of a prior
* calibration.
gen_cam_par_area_scan_division (0.01, 30, 4.65e-006, 4.65e-006, \
                                640, 480, 1280, 960, CameraParam)
* Inverse projection.
get_line_of_sight([50, 100], [100, 200], CameraParam, PX, PY, PZ, QX, QY, QZ)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_line_of_sight` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_cam_par`, `camera_calibration`

Nachfolger

`affine_trans_point_3d`

Siehe auch

`camera_calibration`, `disp_caltab`, `read_cam_par`, `project_3d_point`, `affine_trans_point_3d`

Modul

Calibration

14.4 Kalibrierkörper

caltab_points (: : CalPlateDescr : X, Y, Z)

Lesen der Markenmittelpunkte aus einer Kalibrierkörperbeschreibungsdatei.

caltab_points liest die Markenmittelpunkte aus der Kalibrierkörperbeschreibungsdatei CalPlateDescr (siehe gen_caltab für Kalibrierplatten mit rechteckiger Markenanordnung und create_caltab für Kalibrierplatten mit hexagonaler Markenanordnung) und liefert ihre Koordinaten in X, Y und Z zurück. Die Markenmittelpunkte liegen als 3D-Koordinaten im Kalibrierkörperkoordinatensystem vor und stellen damit das 3D-Modell des Kalibrierkörpers dar. Das Kalibrierkörperkoordinatensystem ist für Kalibrierplatten mit rechteckiger Markenanordnung im Mittelpunkt der Kalibrierkörperoberfläche platziert. Für Kalibrierplatten mit hexagonaler Markenanordnung befindet sich der Ursprung in der Mitte der zentralen Marke des ersten Suchmusters. Die z-Achse zeigt in den Kalibrierkörper hinein, die x-Achse nach rechts und die y-Achse nach unten.

Die von caltab_points ausgegebenen Markenmittelpunkte dienen typischerweise als Eingabe-Parameter für den Operator camera_calibration. Dieser Operator projiziert die Modellpunkte in das Bild, minimiert die Distanz zwischen den projizierten Modellpunkten und den im Bild beobachteten 2D-Koordinaten (vgl. find_marks_and_pose) und ermittelt so die genauen Werte für die gesuchten internen und externen Kameraparameter.

Parameter

▷ **CalPlateDescr** (input_control) ..... filename.read ⇝ *string*
Dateiname der Kalibrierkörperbeschreibungsdatei.
Defaultwert: 'calplate_320mm.cpd'
Werteliste: CalPlateDescr ∈ {'calplate_5mm.cpd', 'calplate_10mm.cpd', 'calplate_20mm.cpd', 'calplate_40mm.cpd', 'calplate_80mm.cpd', 'calplate_160mm.cpd', 'calplate_320mm.cpd', 'calplate_640mm.cpd', 'calplate_1200mm.cpd', 'calplate_20mm_dark_on_light.cpd', 'calplate_40mm_dark_on_light.cpd', 'calplate_80mm_dark_on_light.cpd', 'caltab_650um.descr', 'caltab_2500um.descr', 'caltab_6mm.descr', 'caltab_10mm.descr', 'caltab_30mm.descr', 'caltab_100mm.descr', 'caltab_200mm.descr', 'caltab_800mm.descr', 'caltab_small.descr', 'caltab_big.descr'}
Dateiendung: .cpd, .descr

▷ **X** (output_control) ..... real-array ⇝ *real*
X-Koordinaten der Markenmittelpunkte im Koordinatensystem des Kalibrierkörpers.

▷ **Y** (output_control) ..... real-array ⇝ *real*
Y-Koordinaten der Markenmittelpunkte im Koordinatensystem des Kalibrierkörpers.

▷ **Z** (output_control) ..... real-array ⇝ *real*
Z-Koordinaten der Markenmittelpunkte im Koordinatensystem des Kalibrierkörpers.

Beispiel

```
* Read calibration image.
read_image(Image, 'calib/calib-3d-coord-03')
CalTabDescr := 'caltab_100mm.descr'
* Find calibration pattern.
find_caltab(Image, CalPlate1, CalTabDescr, 3, 112, 5)
* Find calibration marks and start poses.
StartCamPar := ['area_scan_division', 0.008, 0.0, 0.000011, 0.000011, \
                384, 288, 768, 576]
find_marks_and_pose(Image,CalPlate1,CalTabDescr, StartCamPar, \
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1, \
                    StartPose1)
* Read 3D positions of calibration marks.
caltab_points(CalTabDescr, NX, NY, NZ)
*  Calibrate camera.
camera_calibration(NX, NY, NZ, RCoord1, CCoord1, StartCamPar, \
                   StartPose1, 'all', CameraParam, FinalPose, Errors)
*  Visualize calibration result.
dev_display(Image)
disp_caltab(WindowHandle, CalTabDescr, CameraParam, FinalPose, 1.0)
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei CalPlateDescr erfolgreich gelesen werden, dann liefert caltab_points den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`camera_calibration`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`, `project_3d_point`, `get_line_of_sight`, `gen_caltab`

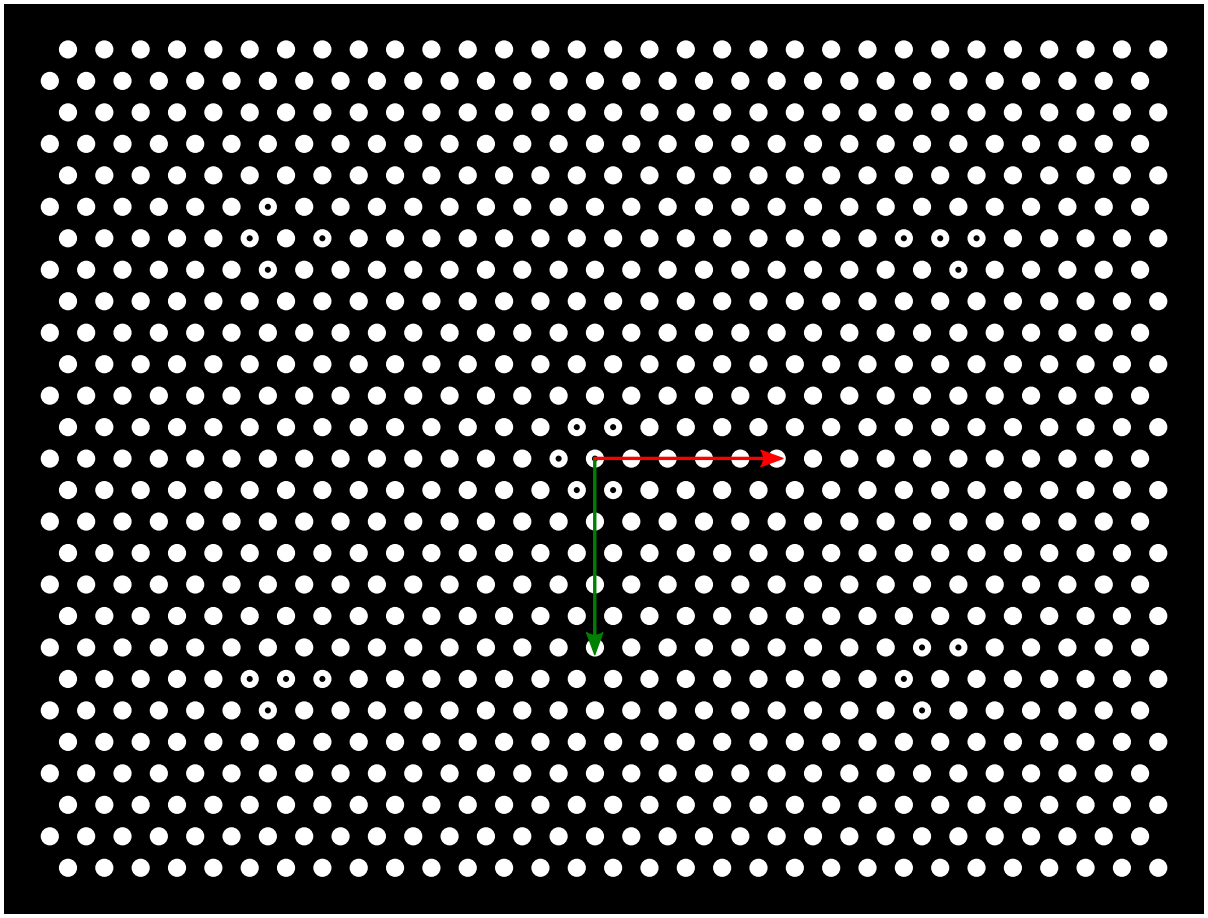
Modul

Foundation

create_caltab (: : NumRows, MarksPerRow, Diameter, FinderRow, FinderColumn, Polarity, CalPlateDescr, CalPlatePSFile :)

Erzeugen der Kalibrierkörperbeschreibungsdatei und der passenden PostScript-Datei für eine Kalibrierplatte mit hexagonaler Markenanordnung.

`create_caltab` erzeugt die Kalibrierkörperbeschreibungsdatei eines Standardkalibrierkörpers für HALCON mit hexagonaler Markenanordnung. Dieser Kalibrierkörper besteht aus `NumRows` mal `MarksPerRow` kreisförmigen Marken. Diese Marken sind in einem hexagonalen Muster angeordnet, so dass jede Marke (ausgenommen die am Rand liegenden) sechs Nachbarn mit gleichem Abstand hat.



Ein Standard-HALCON-Kalibrierkörper mit hexagonaler Markenanordnung und sein Koordinatensystem.

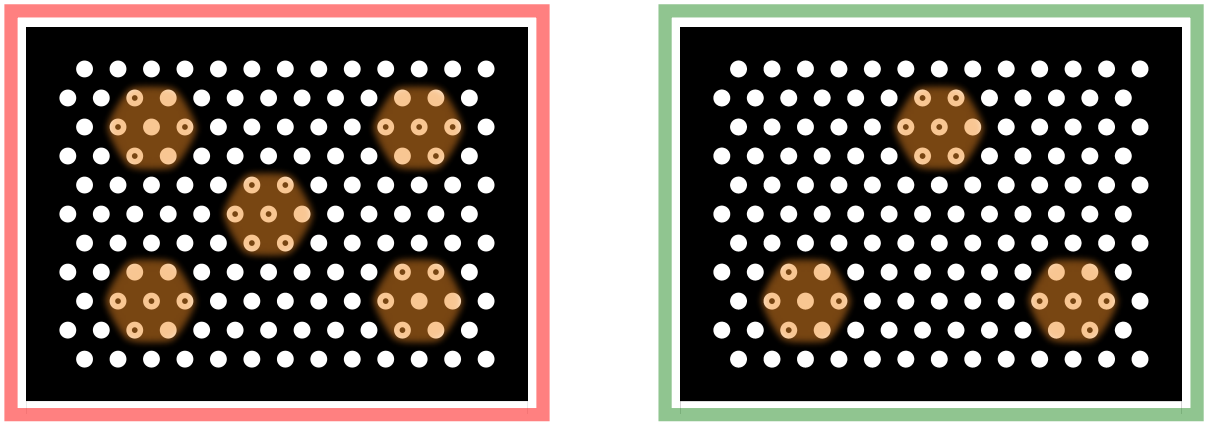
Der Durchmesser der Marken wird durch den Parameter `Diameter` in Metern festgelegt. Der Abstand zwischen den Zentren der horizontal benachbarten Marken ist $2 \cdot Diameter$. Der Abstand zwischen benachbarten Zeilen von Kalibriermarken ist $2 \cdot Diameter \cdot \sqrt{0.75}$. Die Breite und die Höhe der generierten Kalibrierplatte lassen sich mit den folgenden Formeln ermitteln:

$\texttt{Width} = (2 \left\lfloor \frac{\texttt{MarksPerRow}-1}{2} \right\rfloor + 3) \cdot 2 \cdot \texttt{Diameter}$

$\texttt{Height} = (2 \left\lfloor \frac{\texttt{NumRows}-1}{2} \right\rfloor \cdot \sqrt{3} + 5) \cdot \texttt{Diameter}$

Das Muster des Kalibrierkörpers enthält ein bis fünf Suchmuster. Ein Suchmuster besteht aus einem speziellen Hexagon von Marken (also einer Marke und deren sechs Nachbarmarken), das vier oder sechs Marken mit einem Loch enthält. Jedes dieser bis zu fünf Suchmuster ist eindeutig und kann dazu verwendet werden die Orientierung des Kalibrierkörpers und die Lage des Suchmusters auf dem Kalibrierkörper zu bestimmen. Mindestens eines der Suchmuster muss vollständig sichtbar sein, damit der Kalibrierkörper von `find_calib_object` gefunden werden kann. Die Position der zentralen Marken der Suchmuster werden in `FinderRow` und `FinderColumn` angegeben. Die Länge dieser Tupel bestimmt somit die Anzahl der Suchmuster auf dem Kalibrierkörper. Zwei Suchmuster dürfen sich nicht überlappen. Es wird empfohlen, einen gewissen Abstand zwischen den Suchmustern einzuhalten damit die Marken mit Loch eindeutig einem Suchmuster zugeordnet werden können. Als Faustregel gilt: Wenn zu wenig Marken vorhanden sind, um die Suchmuster an möglichst eindeutigen Stellen zu platzieren, sollte die Anzahl der Suchmuster reduziert werden, damit sie gleichmäßiger verteilt werden können. Ein solches Beispiel ist weiter unten abgebildet, wobei zu beachten ist, dass das Finden der Suchmuster auch vom verwendeten Kamerasetup abhängt.

Der Ursprung des Koordinatensystems des Kalibrierkörpers befindet sich in der Mitte der zentralen Marke des ersten Suchmusters.



Die Suchmuster einer Kalibrierplatte sollten nicht zu nahe zueinander platziert werden (links). Falls nicht genügend Marken auf der Kalibrierplatte sind, um die Suchmuster genügend weit voneinander zu platzieren, sollte die Anzahl an Suchmustern entsprechend reduziert werden (rechts).

Abhängig von `Polarity` sind die Marken entweder hell auf dunklem Hintergrund (*'light_on_dark'*) oder dunkel auf hellem Hintergrund (*'dark_on_light'*).

Die Datei `CalPlateDescr` enthält die Kalibrierkörperbeschreibung, die bei allen HALCON-Operationen die diesen Kalibrierkörber verwenden, angegeben werden muss (zum Beispiel in `set_calib_data_calib_object` oder `sim_caltab`). Die Dateiendung für eine Beschreibungsdatei eines Kalibrierkörpers mit hexagonaler Markenanordnung ist 'cpd'.

Eine Kalibrierkörperbeschreibung enthält Informationen über:

- die Anzahl der Zeilen und Spalten des Kalibrierkörpers
- die Anzahl der Marken pro Zeile und Spalte
- die Verschiebung des Koordinatensystems zur Plattenoberfläche in z-Richtung
- den Rand des Kalibrierkörpers
- die Polarität der Marken
- die Anzahl und Position der Suchmuster
- die x,y-Koordinaten und den Radius von jeder Kalibriermarke

Eine Datei, die mit `create_caltab` erzeugt wurde, sieht z.B. folgendermaßen aus (Kommentare sind durch ein '#' am Anfang der Zeile markiert):

```
# Plate Description Version 3
# HALCON Version 20.11 --  Wed Dec 16 11:02:00 2020
# Description of the standard calibration plate
# used for the camera calibration in HALCON
# (generated by create\_caltab)
#
#

# 27 rows x 31 columns
# Width, height of calibration plate [meter]: 0.170323, 0.129118
# Distance between mark centers [meter]: 0.0051613

# Number of marks in y-dimension (rows)
r 27

# Number of marks in x-dimension (columns)
c 31

# offset of coordinate system in z-dimension [meter] (optional):
z 0

# rim of the calibration plate (min x, max y, max x, min y) [meter]:
o -0.083871125 0.0645592449151841 0.086451775 -0.0645592449151841

# polarity of the marks (light or dark):
p light

# number of finder pattern marks:
f 5

# position of the finder patterns (central mark):  x y [index]
15 13
6 6
24 6
6 20
24 20

# calibration marks:  x y radius [meter]

# calibration marks at y = -0.0581076 m
-0.07483885 -0.0581076199151841 0.001290325
-0.06967755 -0.0581076199151841 0.001290325
-0.06451625 -0.0581076199151841 0.001290325
-0.05935495 -0.0581076199151841 0.001290325
-0.05419365 -0.0581076199151841 0.001290325
-0.04903235 -0.0581076199151841 0.001290325
-0.04387105 -0.0581076199151841 0.001290325
-0.03870975 -0.0581076199151841 0.001290325
-0.03354845 -0.0581076199151841 0.001290325
-0.02838715 -0.0581076199151841 0.001290325
-0.02322585 -0.0581076199151841 0.001290325
-0.01806455 -0.0581076199151841 0.001290325
-0.01290325 -0.0581076199151841 0.001290325
-0.00774195 -0.0581076199151841 0.001290325
-0.00258065 -0.0581076199151841 0.001290325
0.00258065 -0.0581076199151841 0.001290325
0.00774195 -0.0581076199151841 0.001290325
0.01290325 -0.0581076199151841 0.001290325
0.01806455 -0.0581076199151841 0.001290325
```

```
0.02322585 -0.0581076199151841 0.001290325
0.02838715 -0.0581076199151841 0.001290325
0.03354845 -0.0581076199151841 0.001290325
0.03870975 -0.0581076199151841 0.001290325
0.04387105 -0.0581076199151841 0.001290325
0.04903235 -0.0581076199151841 0.001290325
0.05419365 -0.0581076199151841 0.001290325
0.05935495 -0.0581076199151841 0.001290325
0.06451625 -0.0581076199151841 0.001290325
0.06967755 -0.0581076199151841 0.001290325
0.07483885 -0.0581076199151841 0.001290325
0.08000015 -0.0581076199151841 0.001290325

# calibration marks at y = -0.0536378 m
-0.0774195 -0.0536378029986315 0.001290325
-0.0722582 -0.0536378029986315 0.001290325
-0.0670969 -0.0536378029986315 0.001290325
-0.0619356 -0.0536378029986315 0.001290325
-0.0567743 -0.0536378029986315 0.001290325
-0.051613 -0.0536378029986315 0.001290325
-0.0464517 -0.0536378029986315 0.001290325
-0.0412904 -0.0536378029986315 0.001290325
-0.0361291 -0.0536378029986315 0.001290325
-0.0309678 -0.0536378029986315 0.001290325
-0.0258065 -0.0536378029986315 0.001290325
-0.0206452 -0.0536378029986315 0.001290325
-0.0154839 -0.0536378029986315 0.001290325
-0.0103226 -0.0536378029986315 0.001290325
-0.0051613 -0.0536378029986315 0.001290325
0 -0.0536378029986315 0.001290325
0.0051613 -0.0536378029986315 0.001290325
0.0103226 -0.0536378029986315 0.001290325
0.0154839 -0.0536378029986315 0.001290325
0.0206452 -0.0536378029986315 0.001290325
0.0258065 -0.0536378029986315 0.001290325
0.0309678 -0.0536378029986315 0.001290325
0.0361291 -0.0536378029986315 0.001290325
0.0412904 -0.0536378029986315 0.001290325
0.0464517 -0.0536378029986315 0.001290325
0.051613 -0.0536378029986315 0.001290325
0.0567743 -0.0536378029986315 0.001290325
0.0619356 -0.0536378029986315 0.001290325
0.0670969 -0.0536378029986315 0.001290325
0.0722582 -0.0536378029986315 0.001290325
0.0774195 -0.0536378029986315 0.001290325

# calibration marks at y = -0.049168 m
...

# calibration marks at y = -0.0446982 m
...

# calibration marks at y = -0.0402284 m
...

# calibration marks at y = -0.0357585 m
...

# calibration marks at y = -0.0312887 m
```

```
...

# calibration marks at y = -0.0268189 m
...

# calibration marks at y = -0.0223491 m
...

# calibration marks at y = -0.0178793 m
...

# calibration marks at y = -0.0134095 m
...

# calibration marks at y = -0.00893963 m
...

# calibration marks at y = -0.00446982 m
...

# calibration marks at y = 0 m
...

# calibration marks at y = 0.00446982 m
...

# calibration marks at y = 0.00893963 m
...

# calibration marks at y = 0.0134095 m
...

# calibration marks at y = 0.0178793 m
...

# calibration marks at y = 0.0223491 m
...

# calibration marks at y = 0.0268189 m
...

# calibration marks at y = 0.0312887 m
...

# calibration marks at y = 0.0357585 m
...

# calibration marks at y = 0.0402284 m
...

# calibration marks at y = 0.0446982 m
...

# calibration marks at y = 0.049168 m
...

# calibration marks at y = 0.0536378 m
...
```

```
# calibration marks at y = 0.0581076 m
...
```

Es ist zu beachten, dass hier für eine bessere Übersichtlichkeit nur die Koordinaten und Radien der Marken aus den ersten beiden Zeilen vollständig aufgeführt sind. Die entsprechenden Koordinaten und Radien der Marken in den anderen Zeilen wurden daher ausgelassen.

Die Datei `CalPlatePSFile` enthält die korrespondierende PostScript-Beschreibung des Kalibrierkörpers, die zum Ausdrucken des Kalibrierkörpers verwendet werden kann.

Achtung

Je nach Genauigkeit des verwendeten Ausgabegeräts (z.B. Laserdrucker) stimmt der ausgedruckte Kalibrierkörper nicht genau mit der Kalibrierkörperbeschreibungsdatei `CalPlateDescr` überein. Es müssen daher ggf. die Koordinaten der Kalibrierkörpermarken in der Kalibrierkörperbeschreibungsdatei an die realen Maße angepasst werden!

Parameter

- ▷ **`NumRows`** (input_control) ..... integer ⇝ *integer*
  Number of rows.
  Defaultwert: 27
  Empfohlene Schrittweite: 1
  Restriktion: NumRows > 2
- ▷ **`MarksPerRow`** (input_control) ..... integer ⇝ *integer*
  Number of marks per line.
  Defaultwert: 31
  Empfohlene Schrittweite: 1
  Restriktion: MarksPerRow > 2
- ▷ **`Diameter`** (input_control) ..... real ⇝ *real*
  Durchmesser der Marken.
  Defaultwert: 0.00258065
  Wertevorschläge: `Diameter` ∈ {0.00258065, 0.1, 0.0125, 0.00375, 0.00125}
- ▷ **`FinderRow`** (input_control) ..... integer(-array) ⇝ *integer*
  Zeilenindizes der Suchmuster.
  Defaultwert: [13,6,6,20,20]
- ▷ **`FinderColumn`** (input_control) ..... integer(-array) ⇝ *integer*
  Spaltenindizes der Suchmuster.
  Defaultwert: [15,6,24,6,24]
- ▷ **`Polarity`** (input_control) ..... string ⇝ *string*
  Polarität der Marken
  Defaultwert: 'light_on_dark'
  Wertevorschläge: `Polarity` ∈ {'light_on_dark', 'dark_on_light'}
- ▷ **`CalPlateDescr`** (input_control) ..... filename.write ⇝ *string*
  Dateiname der Kalibrierplattenbeschreibung.
  Defaultwert: 'calplate.cpd'
  Werteliste: `CalPlateDescr` ∈ {'calplate.cpd'}
  Dateiendung: `.cpd`
- ▷ **`CalPlatePSFile`** (input_control) ..... filename.write ⇝ *string*
  Dateiname der PostScript Datei.
  Defaultwert: 'calplate.ps'
  Dateiendung: `.ps`

Beispiel

```
* Parameters to create the descriptor for the 160mm wide calibration
* plate.
create_caltab (27, 31, 0.00258065, [13, 6, 6, 20, 20], [15, 6, 24, 6, 24], \
               'light_on_dark', 'calplate.cpd', 'caltab.ps')
```

Ergebnis

Sind die Parameterwerte korrekt und konnten beide Dateien erfolgreich geschrieben werden, dann liefert `create_caltab` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`read_cam_par`, `caltab_points`

Alternativen

`gen_caltab`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`

Modul

Foundation

disp_caltab (: : WindowHandle, CalPlateDescr, CameraParam, CalPlatePose, ScaleFac :)

Projizieren und Zeichnen des 3D-Kalibrierkörpermodells ins Bild.

`disp_caltab` visualisiert im durch `WindowHandle` angegebenen Fenster die Marken und die Verbindungslinien zwischen den Marken des durch `CalPlateDescr` beschriebenen Kalibrierkörpers. Zusätzlich werden die x- und y-Achse des Koordinatensystems der Platte auf der Plattenoberfläche eingezeichnet. Dazu wird das 3D-Modell des Kalibrierkörpers mittels der internen (`CameraParam`) und externen Kameraparameter (`CalPlatePose` auf die Bildebene projiziert. Die 3D-Lage `CalPlatePose` wird in der Form ${}^{ccs}\mathbf{P}_{wcs}$ erwartet. Dabei steht *ccs* für das Kamerakoordinatensystem und *wcs* für das Weltkoordinatensystem (siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision"). Dies bedeutet, die 3D-Lage entspricht der Lage des Kalibrierkörpers in Kamerakoordinaten. Das verwendete Kameramodell wird in Kalibrierung beschrieben.

Typischerweise wird `disp_caltab` verwendet, um das Ergebnis der Kamerakalibrierung (vgl. Kalibrierung oder `camera_calibration`) durch Überblendung auf das originale Bild zu verifizieren. Die aktuelle Strichstärke kann mit `set_line_width`, die aktuelle Zeichenfarbe mit `set_color` gesetzt werden. Ferner kann der Beschriftungsstiel der Koordinatenachsen mit `set_font` verändert werden.

Der Parameter `ScaleFac` beeinflusst die Anzahl der Stützstellen, mit der die ellipsenförmigen Konturen der Kalibrierkörpermarken approximiert werden. Die Erhöhung der Stützstellenanzahl ist sinnvoll, wenn der Bildbereich im Ausgabefenster vergrößert dargestellt wird (vgl. `set_part`).

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster, in dem die Kalibrierplatte visualisiert werden soll.

▷ **CalPlateDescr** (input_control) . . . . . filename.read ⇝ *string*
Dateiname der Kalibrierkörperbeschreibungsdatei.
Defaultwert: 'calplate_320.cpd'
Werteliste: `CalPlateDescr` ∈ {'calplate_5mm.cpd', 'calplate_10mm.cpd', 'calplate_20mm.cpd', 'calplate_40mm.cpd', 'calplate_80mm.cpd', 'calplate_160mm.cpd', 'calplate_320mm.cpd', 'calplate_640mm.cpd', 'calplate_1200mm.cpd', 'calplate_20mm_dark_on_light.cpd', 'calplate_40mm_dark_on_light.cpd', 'calplate_80mm_dark_on_light.cpd', 'caltab_650um.descr', 'caltab_2500um.descr', 'caltab_6mm.descr', 'caltab_10mm.descr', 'caltab_30mm.descr', 'caltab_100mm.descr', 'caltab_200mm.descr', 'caltab_800mm.descr', 'caltab_small.descr', 'caltab_big.descr'}
Dateiendung: `.cpd`, `.descr`

▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
Interne Kameraparameter.

▷ **CalPlatePose** (input_control) . . . . . pose ⇝ *real* / integer
externe Kameraparameter (3D-Lage des Kalibrierkörpers in Kamerakoordinaten).
Parameteranzahl: 7

▷ **ScaleFac** (input_control) ..... real ⇝ *real*
Skalierungsfaktor für Darstellung.
Defaultwert: 1.0
Wertevorschläge: ScaleFac ∈ {0.5, 1.0, 2.0, 3.0}
Empfohlene Schrittweite: 0.05
Restriktion: 0.0 < ScaleFac

Beispiel

```
* Read image of calibration plate.
read_image (Image, 'calib/calib_single_camera_01')
get_image_size (Image, Width, Height)
* Create and setup the calibration model.
create_calib_data ('calibration_object', 1, 1, CalibDataID)
CalPlateDescr := 'calplate_80mm.cpd'
set_calib_data_calib_object (CalibDataID, 0, CalPlateDescr)
CamParam := ['area_scan_division', 0.008, -1500, 3.7e-6, 3.7e-6, \
             640, 470, 1292, 964]
set_calib_data_cam_param (CalibDataID, 0, [], CamParam)
* Localize calibration plate in the image.
find_calib_object (Image, CalibDataID, 0, 0, 0, [], [])
get_calib_data_observ_pose (CalibDataID, 0, 0, 0, StartPose)
* Display calibration plate.
disp_caltab (WindowHandle, CalPlateDescr, CamParam, StartPose, 1)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert disp_caltab den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

camera_calibration, read_cam_par, read_pose

Siehe auch

find_marks_and_pose, camera_calibration, sim_caltab, write_cam_par, read_cam_par, create_pose, write_pose, read_pose, project_3d_point, get_line_of_sight

Modul

Foundation

find_calib_object (Image : : CalibDataID, CameraIdx, CalibObjIdx, CalibObjPoseIdx, GenParamName, GenParamValue :)

Finden der HALCON-Kalibrierplatte im Bild, Extrahieren der Kalibriermarken sowie Speichern der extrahierten Daten im Kalibrierdatenmodell.

find_calib_object sucht im Bild Image nach einer HALCON-Kalibrierplatte, welche der Beschreibung des Kalibrierobjekts CalibObjIdx im Kalibrierdatenmodell CalibDataID entspricht. Wird die Kalibrierplatte gefunden, extrahiert find_calib_object die Mittelpunkte und die Konturen der Kalibriermarken und schätzt die 3D-Pose der Kalibrierplatte relativ zur beobachtenden Kamera CameraIdx. Diese Daten werden im Kalibrierdatenmodell für die Ansicht CalibObjPoseIdx des Kalibrierkörpers gespeichert. Um ein erfolgreiches Auffinden zu ermöglichen, muss mindestens ein Suchmuster vollständig im Bild sichtbar sein. Bei Kalibrierplatten mit hexagonaler Markenanordnung ist dies ein Hexagon von Marken, das vier oder sechs Marken mit einem Loch enthält, während es bei Kalibrierplatten mit rechteckiger Markenanordnung der Rahmen mit Dreieck in einer Ecke ist.

Vorbereitung der Eingabedaten

Bevor der Operator `find_calib_object` aufgerufen werden kann, muss zunächst ein Kalibrierdatenmodell definiert werden. Dabei müssen die folgenden Schritte durchgeführt werden:

1. **Erstellen des Kalibrierdatenmodells** mit dem Operator `create_calib_data`, wobei die Anzahl der Kameras und die Anzahl der Kalibrierkörper im Modell festgelegt werden.
2. **Definieren des Kameratyps und der internen Kameraparameter** für alle Kameras mit dem Operator `set_calib_data_cam_param`. Dabei ist zu beachten, dass nur Kameras vom gleichen Typ in einem Aufbau kalibriert werden können.
3. **Definieren der Kalibrierkörperbeschreibung** für alle Kalibrierkörper mit dem Operator `set_calib_data_calib_object`. Dabei ist zu beachten, dass für einen erfolgreichen Aufruf von `find_calib_object` eine gültige Beschreibungsdatei der Kalibrierplatte benötigt wird. Diese Beschreibungsdatei muss zuvor über `set_calib_data_calib_object` gesetzt werden. Die Verwendung eines benutzerdefinierten Kalibrierkörpers kann dementsprechend nur über den Operator `set_calib_data_observ_points` erfolgen.

Sammeln von Beobachtungen

`find_calib_object` sammelt Beobachtungen und speichert diese im Kalibrierdatenmodell (siehe `set_calib_data_observ_points` für eine Beschreibung der Kalibrierdatenbeobachtungen). Darüber hinaus speichert er zusätzliche Informationen für die Beobachtungen, welche dem Modell nicht mit `set_calib_data_observ_points` zugefügt werden können und abhängig von der verwendeten Kalibrierplatte sind. Während für Kalibrierplatten mit rechteckiger Markenanordnung (siehe `gen_caltab`) der Rand der Kalibrierplatte den Beobachtungen hinzugefügt wird, wird bei Kalibrierplatten mit hexagonalem Muster (siehe `create_caltab`) eines der Suchmuster gespeichert. Zusätzlich werden unabhängig von der verwendeten Kalibrierplatte die Konturen der einzelnen Marken dem Kalibrierdatenmodell zugeführt.

Setzen zusätzlicher Parameter

Bei **Kalibrierplatten mit hexagonaler Markenanordnung** kann mit `GenParamName` und `GenParamValue` folgender, zusätzlicher Parameter gesetzt werden:

'sigma': Glättungsfaktor für die Extraktion der Markenkonturen. Je größer der Wert für *'sigma'* ist, umso größer ist die Filterbreite und damit das Einzugsgebiet des Filters (siehe auch `edges_sub_pix` für den Einfluss der Filterbreite bei Canny-Filterung).

`Wertevorschläge`: *0.5*, *0.7*, *0.9*, *1.0* `(default)`, *1.2*, *1.5*

Für **Kalibrierplatten mit rechteckiger Markenanordnung** entspricht `find_calib_object` der Operatorfolge `find_caltab`, `find_marks_and_pose` und `set_calib_data_observ_points`. Für diese Kalibrierplatten können mit `GenParamName` und `GenParamValue` die folgenden Parameter gesetzt werden:

'alpha': Glättungsfaktor für die Extraktion der Markenkonturen. Je größer der Wert für *'alpha'* ist, umso kleiner ist die Filterbreite und damit das Einzugsgebiet des Filters (siehe auch `edges_sub_pix` für den Einfluss der Filterbreite bei Lanser2-Filterung).

`Wertevorschläge`: *0.5*, *0.7*, *0.9* `(default)`, *1.0*, *1.2*, *1.5*

'gap_tolerance': Toleranzfaktor für den Abstand zwischen einzelnen Marken. Falls die Marken dichter aneinander erscheinen als erwartet, kann *'gap_tolerance'* < *1.0* gesetzt werden, um zu vermeiden, dass Störungsmuster außerhalb der Kalibrierplatte mit der Kalibrierplatte assoziiert werden. Dies ist z.B. dann sinnvoll, wenn der Hintergrund Muster aufweist, die den Kalibriermarken ähneln. Wenn die Abstände zwischen einzelnen Marken sehr stark variieren, z.B., wenn die Platte stark perspektivisch verzerrt im Bild erscheint, kann man *'gap_tolerance'* > *1.0* setzen und so auch die Gruppierung über größere Distanzen ermöglichen. (siehe auch `find_caltab`)

`Wertevorschläge`: *0.75*, *0.9*, *1.0* `(default)`, *1.1*, *1.2*, *1.5*

'max_diam_marks': Maximal erwarteter Durchmesser der Marken (intern benötigt von `find_marks_and_pose`). Standardmäßig wird dieser Wert durch einen vorhergehenden internen Aufruf von `find_caltab` geschätzt. Wenn allerdings die Schätzung fehlerhaft ist, der interne Aufruf von `find_caltab` scheitert oder übersprungen wird (siehe *'skip_find_caltab'*), kann eine separate Anpassung dieses Parameterwerts notwendig sein.

`Wertevorschläge`: *50.0*, *100.0*, *150.0*, *200.0*, *300.0*

'skip_find_caltab': Überspringen des internen Aufrufs von `find_caltab`. Wenn *'skip_find_caltab'*=*'true'*, wird nur die Bildregion des Bildes `Image` zum Begrenzen der Markensuche in `find_marks_and_pose` berücksichtigt. Über diesen Mechanismus kann ein benutzerdefiniertes Suchen der Kalibrierkörperregion nach Bedarf implementiert werden. Der Suchbereich ist dann mit dem Operator `reduce_domain` vor dem Aufruf von `find_calib_object` einzuschränken.

`Werteliste`: *'false'* (`default`), *'true'*

Wir empfehlen, bei Nutzung einer HALCON-Kalibrierplatte als Kalibrierkörper möglichst immer den Operator `find_calib_object` zu verwenden, da dieser im Gegensatz zu `set_calib_data_observ_points` auch die Konturen der Kalibriermarken im Kalibrierdatenmodell speichert und dadurch eine genauere Kalibrierung mit dem Operator `calibrate_cameras` ermöglicht.

Nach einem erfolgreichen Aufruf von `find_calib_object` können die extrahierten Punkte und Konturen mit dem Operator `get_calib_data_observ_points` bzw. `get_calib_data_observ_contours` abgefragt werden.

Parameter

▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
Eingabebild.

▷ **`CalibDataID`** (input_control) ..... calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **`CameraIdx`** (input_control) ..... number ⇝ *integer*
Index der beobachtenden Kamera.
Defaultwert: 0
Wertevorschläge: `CameraIdx` ∈ {0, 1, 2}

▷ **`CalibObjIdx`** (input_control) ..... number ⇝ *integer*
Index des beobachteten Kalibrierkörpers.
Defaultwert: 0
Wertevorschläge: `CalibObjIdx` ∈ {0, 1, 2}

▷ **`CalibObjPoseIdx`** (input_control) ..... number ⇝ *integer*
Index der beobachteten Pose.
Defaultwert: 0
Wertevorschläge: `CalibObjPoseIdx` ∈ {0, 1, 2}
Restriktion: CalibObjPoseIdx >= 0

▷ **`GenParamName`** (input_control) ..... attribute.name-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'gap_tolerance', 'alpha', 'sigma', 'max_diam_marks', 'skip_find_caltab'}

▷ **`GenParamValue`** (input_control) ..... attribute.value-array ⇝ *string* / real / integer
Werte der generischen Parameter.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {0.5, 0.9, 1.0, 1.2, 1.5, 2.0, 'true', 'false'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`read_image`, `find_marks_and_pose`, `set_calib_data_cam_param`, `set_calib_data_calib_object`

Nachfolger

set_calib_data, calibrate_cameras

Alternativen

find_caltab, find_marks_and_pose, set_calib_data_observ_points

Modul

Calibration

find_caltab (Image : CalPlate : CalPlateDescr, SizeGauss, MarkThresh, MinDiamMarks :)

Segmentieren der Region einer Standard-Kalibrierplatte mit rechteckiger Markenanordnung im Bild.

find_caltab bestimmt die Region einer Standard-Kalibrierplatte mit runden Marken im übergebenen Bild Image. Die gesuchte Region muss einer Standard-Kalibrierplatte mit rechteckiger Markenanordnung entsprechen, welche in der Datei CalPlateDescr spezifiziert ist. Die segmentierte Region wird in CalPlate zurückgeliefert. Der Operator bietet zwei Algorithmen an. Wenn man jeweils einen passenden Integerwert im Parameter SizeGauss, MarkThresh und MinDiamMarks übergibt, wird der *Standardalgorithmus* aufgerufen. Wenn man eine Liste mit Parameternamen und -werten in SizeGauss bzw. MarkThresh übergibt (oder auch nur zwei leere Tupel) wird der *erweiterte Algorithmus* aufgerufen. In diesem Fall wird der Parameter MinDiamMarks ignoriert.

Standardalgorithmus

Zunächst wird das Bild geglättet (vgl. gauss_image). Die Größe der verwendeten Filtermaske wird dabei durch SizeGauss eingestellt. Anschließend wird ein Schwellenwertoperator (vgl. threshold) mit minimalem Grauwert MarkThresh angewandt. Unter den extrahierten zusammenhängenden Regionen wird nun eine möglichst konvexe Region mit annähernd korrekter Anzahl von Löchern (entsprechend den dunklen Marken) gesucht. Um Rauschen zu unterdrücken, werden kleine Löcher mit geringerem Durchmesser als die erwartete Markengröße MinDiamMarks zuvor eliminiert. Die Anzahl der Kalibrierkörpermarken wird der Beschreibungsdatei CalPlateDescr des Kalibrierkörpers entnommen. Die komplette Erklärung dieser Datei ist bei der Beschreibung von gen_caltab zu finden.

Erweiterter Algorithmus

Zunächst wird eine Bildpyramide von Image erstellt. Beginnend mit der höchsten Pyramidenebene werden runde Regionen anhand eines dynamischen Schwellenwerts segmentiert. Anschließend werden sie abhängig von ihrem gegenseitigen Abstand gruppiert. Falls die Anzahl der Kandidaten für die Kalibrierkörpermarken in einer Gruppe die Soll-Anzahl genau erreicht, wird die Suche unterbrochen. Der umschließende helle Bereich wird segmentiert und als Treffer in CalPlate zurückgegeben.

Durch die Bildpyramide ist die Suche unabhängig von der Bild- und Markengröße. Außerdem ist der Algorithmus durch die Regionensegmentierung mithilfe eines dynamischen Schwellenwerts robust gegenüber einer schlechten oder ungleichmäßigen Beleuchtung. Deswegen sind in der Regel, anders als bei dem Standardalgorithmus, keine Parameter nötig. Sind dennoch Anpassungen erwünscht, können ausgewählte Parameter in SizeGauss und ihre Werte in MarkThresh gesetzt werden. Derzeit ist der folgende Parameter verfügbar:

'gap_tolerance': Toleranzfaktor für den Abstand zwischen einzelnen Marken. Falls die Marken dichter aneinander erscheinen als erwartet, kann man *'gap_tolerance'* < *1.0* setzen, um zu vermeiden, dass Störungsmuster außerhalb der Kalibrierplatte mit der Kalibrierplatte assoziiert werden. Dies ist z.B. dann sinnvoll, wenn die Platte gekippt aufgenommen wird und der Hintergrund Muster aufweist, die den Kalibriermarken ähneln. Wenn die Abstände zwischen einzelnen Marken sehr stark variieren, z.B., wenn die Platte stark perspektivisch verzerrt im Bild erscheint, kann man *'gap_tolerance'* > *1.0* setzen und so auch die Gruppierung über größere Distanzen ermöglichen.

Wertevorschläge: *0.75*, *0.9*, *1.0* (default), *1.1*, *1.2*, *1.5*

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object* : byte / uint2
Eingabebild.

▷ **CalPlate** (output_object) ..... region ⇝ *object*
Ausgaberegion.

▷ **CalPlateDescr** (input_control) ..... filename.read ⇝ *string*
Dateiname der Kalibrierkörperbeschreibungsdatei.
Defaultwert: 'caltab_100.descr'
Werteliste: `CalPlateDescr` ∈ {'caltab_650um.descr', 'caltab_2500um.descr', 'caltab_6mm.descr', 'caltab_10mm.descr', 'caltab_30mm.descr', 'caltab_100mm.descr', 'caltab_200mm.descr', 'caltab_800mm.descr', 'caltab_small.descr', 'caltab_big.descr'}
Dateiendung: `.descr`

▷ **SizeGauss** (input_control) ..... integer(-array) ⇝ *integer* / string
Filtergröße.
Defaultwert: 3
Werteliste: `SizeGauss` ∈ {0, 3, 5, 7, 9, 11, 'gap_tolerance'}

▷ **MarkThresh** (input_control) ..... integer(-array) ⇝ *integer* / real
Schwellenwert zur Markenextraktion.
Defaultwert: 112
Wertevorschläge: `MarkThresh` ∈ {48, 64, 80, 96, 112, 128, 144, 160, 0.5, 0.9, 1.0, 1.1, 1.5}

▷ **MinDiamMarks** (input_control) ..... integer ⇝ *integer*
Erwarteter Mindestdurchmesser der Kalibrierkörpermarken.
Defaultwert: 5
Wertevorschläge: `MinDiamMarks` ∈ {3, 5, 9, 15, 30, 50, 70}

Beispiel

```
* Read calibration image.
read_image(Image, 'calib/calib_distorted_01')
* Find calibration pattern.
find_caltab(Image, CalPlate, 'caltab_100mm.descr', 3, 112, 5)
```

Ergebnis

Sind die Parameterwerte korrekt und findet `find_caltab` eine passende Bildregion, dann liefert `find_caltab` den Wert TRUE. Das Verhalten bei leerer Eingabe (kein Eingabebild vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` und das bei leerer Ergebnisregion mit `set_system(::'store_empty_region',<'2 (H_MSG_TRUE)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`read_image`

Nachfolger

`find_marks_and_pose`

Siehe auch

`find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`, `caltab_points`, `gen_caltab`

Modul

Foundation

find_marks_and_pose (Image, CalPlateRegion : : CalPlateDescr, StartCamParam, StartThresh, DeltaThresh, MinThresh, Alpha, MinContLength, MaxDiamMarks : RCoord, CCoord, StartPose)

Extrahieren der rechteckig angeordneten 2D-Kalibriermarken aus dem Bild und Bestimmen der Startwerte für die externen Kameraparameter.

find_marks_and_pose dient zur Berechnung der Eingabedaten für eine nachfolgende Kalibrierung mit einer Kalibrierplatte mit rechteckiger Markenanordnung (vgl. Kalibrierung oder camera_calibration): Zum einen werden die 2D-Mittelpunkte [RCoord,CCoord] der Kalibrierkörpermarken innerhalb der Region CalPlateRegion im Bild Image extrahiert und geordnet. Außerdem wird eine grobe Schätzung für die externen Kameraparameter (StartPose) berechnet, d.h. für die 3D-Lage (= Position und Orientierung, Pose) des Kalibrierkörpers im Kamerakoordinatensystem (siehe create_pose für Information zu 3D-Lagen).

Innerhalb der Region CalPlateRegion, die beispielsweise von dem Operator find_caltab detektiert wurde, im Eingabebild Image wird ein Kantenoperator angewendet, vgl. Operator edges_image, Modus 'lanser2'. Der Filterparameter für diese Kantendetektion ist über Parameter Alpha einstellbar. Um eine stärkere Glättung zu erreichen, muss ein kleinerer Wert für Alpha verwendet werden. Im Kantenbild werden geschlossene Konturen gesucht, deren Anzahl gerade derjenigen aus der Kalibrierkörperbeschreibungsdatei CalPlateDescr entspricht und die ein ellipsenförmiges Aussehen besitzen. Konturen, die kürzer als MinContLength sind, werden dabei verworfen, ebenso Konturen, die eine Bildregion mit einem Durchmesser größer als MaxDiamMarks umschließen (also z.B. die Umrandung des Kalibrierkörpers selbst). Zur Kontursuche im Kantenbild wird ein Schwellenwertoperator angewandt, der die hellen Bereiche im Amplitudenbild des Kantenoperators segmentiert. Zunächst wird der Schwellenwert auf den Wert StartThresh gesetzt. Schlägt die Suche nach den Konturen bzw. die nachfolgende Schätzung der Kameralage fehl, wird der Schwellenwert sukzessive um DeltaThresh bis minimal auf den Wert MinThresh erniedrigt.

Jede der gefundenen Konturen wird subpixelgenau verfeinert (vgl. edges_sub_pix) und anschließend vermöge eines Ausgleichsverfahrens durch eine Ellipse approximiert. Die Mittelpunkte dieser Ellipsen stellen eine gute Näherung der gesuchten 2D-Bildkoordinaten [RCoord,CCoord] der Kalibrierkörpermarkenmittelpunkte dar. Die Reihenfolge der einzelnen Werte innerhalb dieser beiden Tupel erfolgt zeilenweise von links oben nach rechts unten im Bild und muss unbedingt mit der Reihenfolge der 3D-Koordinaten der Marken in der Kalibrierkörperbeschreibungsdatei CalPlateDescr übereinstimmen, da so die Korrespondenzen zwischen den extrahierten Bild- und den bekannten Modellmarken (gegeben durch caltab_points) festgelegt werden! Wenn in einer Ecke der Platte eine dreieckige Marke durch die Beschreibungsdatei CalPlateDescr definiert ist (siehe gen_caltab), wird zusätzlich die Orientierung der Platte ermittelt. Die Bild-Modell-Zuordnung der Punkte beginnt dann mit der Eckmarke des Modells im negativen Quadranten (links oben) und wird zeilenweise von links oben nach rechts unten fortgesetzt.

Aus den Ellipsenparametern der einzelnen Marken wird abschließend noch eine grobe Schätzung für die externen Kameraparameter berechnet. Dazu werden die aufgestellten Korrespondenzen zwischen den extrahierten Bild- und den bekannten Modellmarken verwendet, die sich durch die Reihenfolge der Markenpunkte in [RCoord,CCoord] bzw. in der Datei CalPlateDescr ergeben. Die Schätzung StartPose beschreibt die Lage (Pose) des Kalibrierkörpers im Kamerakoordinatensystem, wie sie als Startwert für die Kamerakalibrierung durch den Operator camera_calibration benötigt wird.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **CalPlateRegion** (input_object) ..... region ⇝ *object*
  Kalibrierkörperregion.
- ▷ **CalPlateDescr** (input_control) ..... filename.read ⇝ *string*
  Dateiname der Kalibrierkörperbeschreibung.
  Defaultwert: 'caltab_100.descr'
  Werteliste: CalPlateDescr ∈ {'caltab_650um.descr', 'caltab_2500um.descr', 'caltab_6mm.descr', 'caltab_10mm.descr', 'caltab_30mm.descr', 'caltab_100mm.descr', 'caltab_200mm.descr', 'caltab_800mm.descr', 'caltab_small.descr', 'caltab_big.descr'}
  Dateiendung: .descr
- ▷ **StartCamParam** (input_control) ..... campar ⇝ *real* / integer / string
  Startwerte für die internen Kameraparameter.
- ▷ **StartThresh** (input_control) ..... number ⇝ *integer*
  Startschwellenwert zur Konturdetektion.
  Defaultwert: 128
  Wertevorschläge: StartThresh ∈ {80, 96, 112, 128, 144, 160}
  Restriktion: StartThresh > 0

▷ **DeltaThresh** (input_control) . . . . . number ⇝ *integer*
Schleifenwert zur Verkleinerung von StartThresh.
Defaultwert: 10
Wertevorschläge: DeltaThresh ∈ {6, 8, 10, 12, 14, 16, 18, 20, 22}
Restriktion: DeltaThresh > 0

▷ **MinThresh** (input_control) . . . . . number ⇝ *integer*
Minimaler Schwellenwert zur Konturdetektion.
Defaultwert: 18
Wertevorschläge: MinThresh ∈ {8, 10, 12, 14, 16, 18, 20, 22}
Restriktion: MinThresh > 0

▷ **Alpha** (input_control) . . . . . real ⇝ *real*
Filterparameter zur Konturdetektion, vgl. edges_image.
Defaultwert: 0.9
Wertevorschläge: Alpha ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1}
Typischer Wertebereich: 0.2 ≤ Alpha ≤ 2.0
Restriktion: Alpha > 0.0

▷ **MinContLength** (input_control) . . . . . real ⇝ *real*
Minimale Länge der Markenkonturen.
Defaultwert: 15.0
Wertevorschläge: MinContLength ∈ {10.0, 15.0, 20.0, 30.0, 40.0, 100.0}
Restriktion: MinContLength > 0.0

▷ **MaxDiamMarks** (input_control) . . . . . real ⇝ *real*
Maximal erwarteter Markendurchmesser.
Defaultwert: 100.0
Wertevorschläge: MaxDiamMarks ∈ {50.0, 100.0, 150.0, 200.0, 300.0}
Restriktion: MaxDiamMarks > 0.0

▷ **RCoord** (output_control) . . . . . real-array ⇝ *real*
Tupel aller detektierten Zeilen-Koordinaten (in Pixel).

▷ **CCoord** (output_control) . . . . . real-array ⇝ *real*
Tupel aller detektierten Spalten-Koordinaten (in Pixel).

▷ **StartPose** (output_control) . . . . . pose ⇝ *real* / integer
Schätzung für die externen Kameraparameter.
Parameteranzahl: 7

Beispiel

```
* Read calibration image.
read_image(Image, 'calib/calib_distorted_01')
* Find calibration pattern.
find_caltab(Image,CalPlate, 'caltab_100mm.descr', 3, 112, 5)
* Find calibration marks and start pose.
find_marks_and_pose(Image, CalPlate, 'caltab_100mm.descr' , \
                    ['area_scan_division', 0.008, 0.0, \
                     0.000011, 0.000011, 384, 288, 640, 512], \
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord, CCoord, StartPose)
```

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Schätzung für die externen Kameraparameter bestimmt werden, dann liefert find_marks_and_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

find_caltab

Nachfolger

camera_calibration

Siehe auch

find_caltab, camera_calibration, disp_caltab, sim_caltab, read_cam_par, read_pose, create_pose, pose_to_hom_mat3d, caltab_points, gen_caltab, edges_sub_pix, edges_image

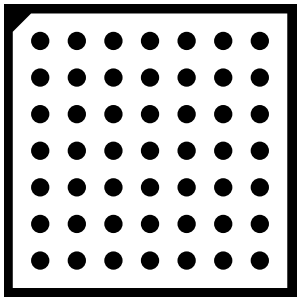
Modul

Foundation

gen_caltab (: : XNum, YNum, MarkDist, DiameterRatio, CalPlateDescr, CalPlatePSFile :)

Erzeugen der Kalibrierkörperbeschreibungsdatei und der passenden PostScript-Datei für eine Kalibrierplatte mit rechteckiger Markenanordnung.

gen_caltab erzeugt die Beschreibung eines planaren Standard-Kalibrierkörpers für HALCON mit rechteckiger Markenanordnung. Der Kalibrierkörper hat XNum mal YNum schwarze, runde Marken auf weißem Grund, die von einem schwarzen Rahmen umrandet sind.



Ein Standard-HALCON-Kalibrierkörper mit rechteckiger Markenanordnung

Die Marken sind in einem rechteckigen Gitter mit YNum und XNum äquidistanten Zeilen und Spalten angeordnet. Den Abstand zwischen diesen Zeilen und Spalten legt der Parameter MarkDist in Metern fest. Der Durchmesser der kreisförmigen Marken kann durch den Parameter DiameterRatio beeinflusst werden und ist über die Gleichung $\texttt{Diameter} = \texttt{MarkDist} \cdot \texttt{DiameterRatio}$ definiert. Wird nun beispielsweise ein Markenabstand von 0.01 m verwendet und das Verhältnis von Markendurchmesser zu Markenabstand auf 0.5 gesetzt, so umfasst der äußere schwarze Rand 8 cm und der Radius einer einzelnen Marke beträgt 2.5 mm. Das Kalibrierkörperkoordinatensystem ist im Mittelpunkt der Kalibrierkörperoberfläche platziert, die z-Achse zeigt in den Kalibrierkörper hinein, die x-Achse nach rechts und die y-Achse nach unten.

Der schwarze Rahmen der Kalibrierplatte enthält in der linken oberen Ecke eine dreieckige schwarze Orientierungsmarkierung, mit welcher die Lage der Kalibrierplatte eindeutig bestimmt werden kann. Die Breite und die Höhe der generierten Kalibrierplatte lassen sich mit den folgenden Formeln ermitteln:

$\texttt{Width} = \texttt{MarkDist} \cdot (\texttt{XNum} + 1)$

$\texttt{Height} = \texttt{MarkDist} \cdot (\texttt{YNum} + 1)$

Die Datei CalPlateDescr enthält die eigentliche Kalibrierkörperbeschreibung, d.h. die Anzahl der Zeilen und Spalten des Kalibrierkörpers, die Geometrie des Rahmens (vgl. find_caltab) und der dreieckigen Orientierungsmarke, eine Verschiebung des Koordinatensystems zur Plattenoberfläche in z-Richtung und die Koordinaten und den Radius aller Kalibrierkörpermarken im Kalibrierkörperkoordinatensystem. Die Definition der Orientierungsmarke t und des Offsets z ist optional und kann auskommentiert werden. Die Dateiendung für die Kalibrierkörperbeschreibung ist in HALCON 'descr'. Eine solche Datei hat z.B. folgendes Aussehen (Kommentare werden durch ein '#' am Zeilenanfang markiert):

```
# Plate Description Version 2
# HALCON Version 7.1 --  Fri Jun 24 16:41:00 2005
```

```
# Description of the standard calibration plate
# used for the camera calibration in HALCON
# (generated by gen\_caltab)
#
#
# 7 rows x 7 columns
# Width, height of the black frame [meter]: 0.1, 0.1
# Distance between mark centers [meter]: 0.0125

# Number of marks in y-dimension (rows)
r 7

# Number of marks in x-dimension (columns)
c 7

#   offset of coordinate system in z-dimension [meter] (optional):
z 0

# Rectangular border (rim and black frame) of calibration plate
#   rim of the calibration plate (min x, max y, max x, min y) [meter]:
o -0.05125 0.05125 0.05125 -0.05125
#   outer border of the black frame (min x, max y, max x, min y) [meter]:
i -0.05 0.05 0.05 -0.05
#   triangular corner mark given by two corner points (x,y, x,y) [meter]
#   (optional):
t -0.05 -0.0375 -0.0375 -0.05

#   width of the black frame [meter]:
w 0.003125

# calibration marks:  x y radius [meter]

# calibration marks at y = -0.0375 m
-0.0375 -0.0375 0.003125
-0.025 -0.0375 0.003125
-0.0125 -0.0375 0.003125
-3.46945e-018 -0.0375 0.003125
0.0125 -0.0375 0.003125
0.025 -0.0375 0.003125
0.0375 -0.0375 0.003125

# calibration marks at y = -0.025 m
-0.0375 -0.025 0.003125
-0.025 -0.025 0.003125
-0.0125 -0.025 0.003125
-3.46945e-018 -0.025 0.003125
0.0125 -0.025 0.003125
0.025 -0.025 0.003125
0.0375 -0.025 0.003125

# calibration marks at y = -0.0125 m
-0.0375 -0.0125 0.003125
-0.025 -0.0125 0.003125
-0.0125 -0.0125 0.003125
-3.46945e-018 -0.0125 0.003125
0.0125 -0.0125 0.003125
0.025 -0.0125 0.003125
0.0375 -0.0125 0.003125
```

```
# calibration marks at y = -3.46945e-018 m
-0.0375 -3.46945e-018 0.003125
-0.025 -3.46945e-018 0.003125
-0.0125 -3.46945e-018 0.003125
-3.46945e-018 -3.46945e-018 0.003125
0.0125 -3.46945e-018 0.003125
0.025 -3.46945e-018 0.003125
0.0375 -3.46945e-018 0.003125

# calibration marks at y = 0.0125 m
-0.0375 0.0125 0.003125
-0.025 0.0125 0.003125
-0.0125 0.0125 0.003125
-3.46945e-018 0.0125 0.003125
0.0125 0.0125 0.003125
0.025 0.0125 0.003125
0.0375 0.0125 0.003125

# calibration marks at y = 0.025 m
-0.0375 0.025 0.003125
-0.025 0.025 0.003125
-0.0125 0.025 0.003125
-3.46945e-018 0.025 0.003125
0.0125 0.025 0.003125
0.025 0.025 0.003125
0.0375 0.025 0.003125

# calibration marks at y = 0.0375 m
-0.0375 0.0375 0.003125
-0.025 0.0375 0.003125
-0.0125 0.0375 0.003125
-3.46945e-018 0.0375 0.003125
0.0125 0.0375 0.003125
0.025 0.0375 0.003125
0.0375 0.0375 0.003125
```

Die Datei `CalPlatePSFile` enthält die korrespondierende PostScript-Beschreibung des Kalibrierkörpers.

Achtung

Je nach der erzielten Genauigkeit des verwendeten Ausgabegeräts (z.B. Laserdrucker) stimmt der ausgedruckte Kalibrierkörper nicht genau mit der Kalibrierkörperbeschreibungsdatei `CalPlateDescr` überein. Es müssen daher ggf. die Koordinaten der Kalibrierkörpermarken in der Kalibrierkörperbeschreibungsdatei an die realen Maße angepasst werden!

Parameter

▷ **XNum** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Marken in X-Richtung.
Defaultwert: 7
Wertevorschläge: XNum ∈ {5, 7, 9}
Empfohlene Schrittweite: 1
Restriktion: XNum > 1

▷ **YNum** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Marken Y-Richtung.
Defaultwert: 7
Wertevorschläge: YNum ∈ {5, 7, 9}
Empfohlene Schrittweite: 1
Restriktion: YNum > 1

▷ **MarkDist** (input_control) ..... real ⇝ *real*
Abstand der Marken in Meter.
Defaultwert: 0.0125
Wertevorschläge: MarkDist ∈ {0.1, 0.0125, 0.00375, 0.00125}
Restriktion: 0.0 < MarkDist

▷ **DiameterRatio** (input_control) ..... real ⇝ *real*
Verhältnis des Markendurchmessers zum Markenabstand.
Defaultwert: 0.5
Wertevorschläge: DiameterRatio ∈ {0.5, 0.55, 0.6, 0.65}
Restriktion: 0.0 < DiameterRatio < 1.0

▷ **CalPlateDescr** (input_control) ..... filename.write ⇝ *string*
Dateiname der Kalibrierkörperbeschreibungsdatei.
Defaultwert: 'caltab.descr'
Werteliste: CalPlateDescr ∈ {'caltab.descr'}
Dateiendung: .descr

▷ **CalPlatePSFile** (input_control) ..... filename.write ⇝ *string*
Dateiname der PostScript-Datei.
Defaultwert: 'caltab.ps'
Dateiendung: .ps

Beispiel

```
* Create calibration plate with width = 80 cm.
gen_caltab( 7, 7, 0.1, 0.5, 'caltab.descr', 'caltab.ps')
```

Ergebnis

Sind die Parameterwerte korrekt und konnten beide Dateien erfolgreich geschrieben werden, dann liefert gen_caltab den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

read_cam_par, caltab_points

Alternativen

create_caltab

Siehe auch

find_caltab, find_marks_and_pose, camera_calibration, disp_caltab, sim_caltab

Modul

Foundation

sim_caltab (: SimImage : CalPlateDescr, CameraParam, CalPlatePose, GrayBackground, GrayPlate, GrayMarks, ScaleFac :)

Simulieren eines Bildes mit Kalibrierkörper.

sim_caltab dient der Simulation eines Kalibrierbildes. Dazu wird die Kalibrierkörperbeschreibung aus der Datei CalPlateDescr eingelesen und die darin enthaltende 3D-Beschreibung mit Hilfe der angegebenen Kameraparameter, d.h., den internen Kameraparametern CameraParam und externen Kameraparametern CalPlatePose, auf die Bildebene projiziert (vgl. project_3d_point). Die 3D-Lage CalPlatePose wird in der Form ${}^{ccs}\mathbf{P}_{wcs}$ erwartet. Dabei steht *ccs* für das Kamerakoordinatensystem und *wcs* für das Weltkoordinatensystem (siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision").

In dem simulierten Bild ist nur der Kalibrierkörper abgebildet. Der Bildhintergrund wird mit dem Grauwert GrayBackground, der Hintergrund des Kalibrierkörpers mit GrayPlate und die Kalibrierkörpermarken werden mit GrayMarks belegt. Der Parameter ScaleFac beeinflusst die Anzahl der Stützstellen, mit der die ellipsenförmigen Konturen der Kalibrierkörpermarken approximiert werden (vgl. disp_caltab). Die Erhöhung der Stützstellenanzahl bewirkt daher eine genauere Bestimmung der Markenbegrenzung, erhöht aber gleichzeitig die Rechenzeit. Für jedes Pixel des simulierten Bildes, das eine solche subpixelgenaue Markenbegrenzung berührt, wird der Grauwert linear zwischen GrayMarks und GrayPlate anhand des Verhältnisses Innerhalb/Außerhalb gesetzt.

Mit dem Operator sim_caltab lassen sich synthetische Kalibrierbilder (mit bekannten Kameraparametern!) erzeugen, mit denen man die Güte des Kalibrieralgorithmus (vgl. Kalibrierung) testen kann.

Parameter

- ▷ **SimImage** (output_object) ..... image ⇝ *object :* byte
  Simuliertes Kalibrierbild.
- ▷ **CalPlateDescr** (input_control) ..... filename.read ⇝ *string*
  Dateiname der Kalibrierkörperbeschreibungsdatei.
  Defaultwert: 'calplate_320mm.cpd'
  Werteliste: CalPlateDescr ∈ {'calplate_5mm.cpd', 'calplate_10mm.cpd', 'calplate_20mm.cpd', 'calplate_40mm.cpd', 'calplate_80mm.cpd', 'calplate_160mm.cpd', 'calplate_320mm.cpd', 'calplate_640mm.cpd', 'calplate_1200mm.cpd', 'calplate_20mm_dark_on_light.cpd', 'calplate_40mm_dark_on_light.cpd', 'calplate_80mm_dark_on_light.cpd', 'caltab_650um.descr', 'caltab_2500um.descr', 'caltab_6mm.descr', 'caltab_10mm.descr', 'caltab_30mm.descr', 'caltab_100mm.descr', 'caltab_200mm.descr', 'caltab_800mm.descr', 'caltab_small.descr', 'caltab_big.descr'}
  Dateiendung: .cpd, .descr
- ▷ **CameraParam** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **CalPlatePose** (input_control) ..... pose ⇝ *real* / integer
  externe Kameraparameter (3D-Lage des Kalibrierkörpers in Kamerakoordinaten).
  Parameteranzahl: 7
- ▷ **GrayBackground** (input_control) ..... integer ⇝ *integer*
  Hintergrundgrauwert des Bildes.
  Defaultwert: 128
  Wertevorschläge: GrayBackground ∈ {0, 32, 64, 96, 128, 160}
  Restriktion: 0 <= GrayBackground <= 255
- ▷ **GrayPlate** (input_control) ..... integer ⇝ *integer*
  Grauwert der Kalibrierkörperplatte.
  Defaultwert: 80
  Wertevorschläge: GrayPlate ∈ {144, 160, 176, 192, 208, 224, 240}
  Restriktion: 0 <= GrayPlate <= 255
- ▷ **GrayMarks** (input_control) ..... integer ⇝ *integer*
  Grauwert der Kalibrierkörpermarkierungen.
  Defaultwert: 224
  Wertevorschläge: GrayMarks ∈ {16, 32, 48, 64, 80, 96, 112}
  Restriktion: 0 <= GrayMarks <= 255
- ▷ **ScaleFac** (input_control) ..... real ⇝ *real*
  Skalierungsfaktor zum Herabsetzen der Überabtastung.
  Defaultwert: 1.0
  Wertevorschläge: ScaleFac ∈ {1.0, 0.5, 0.25, 0.125}
  Empfohlene Schrittweite: 0.05
  Restriktion: 1.0 >= ScaleFac

Beispiel

```
* Read calibration image.
read_image(Image1, 'calib-01')
* Find calibration pattern.
CameraType := 'area_scan_division'
```

```
StartCamPar := [CameraType, Focus, Kappa, Sx, Sy, Cx, Cy, \
                ImageWidth, ImageHeight]
create_calib_data ('calibration_object', 1, 1, CalibDataID)
set_calib_data_cam_param (CalibDataID, 0, [], StartCamPar)
set_calib_data_calib_object (CalibDataID, 0, 'calplate.cpd')
find_caltab(Image1, CalPlate1, 'caltab.descr', 3, 112, 5)
* Find calibration marks and initial pose.
find_calib_object (Image1, CalibDataID, 0, 0, 0, [], [])
* Camera calibration.
calibrate_cameras (CalibDataID, Error)
* Simulate calibration image.
get_calib_data (CalibDataID, 'calib_obj_pose', [0, 0], 'pose', FinalPose)
get_calib_data (CalibDataID, 'camera', 0, 'params', CameraParam)
sim_caltab(Image1Sim, 'calplate.cpd', CameraParam, FinalPose, 128, \
           80, 224, 1)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `sim_caltab` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`camera_calibration`, `find_marks_and_pose`, `read_pose`, `read_cam_par`, `hom_mat3d_to_pose`

Nachfolger

`find_caltab`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `create_pose`, `hom_mat3d_to_pose`, `project_3d_point`, `gen_caltab`

Modul

Calibration

14.5 Kameraparameter

cam_mat_to_cam_par (: : CameraMatrix, Kappa, ImageWidth, ImageHeight : CameraParam)

Berechnen der internen Kameraparameter aus einer Kameramatrix.

`cam_mat_to_cam_par` berechnet interne Kameraparameter aus der Kameramatrix `CameraMatrix`, dem radialen Verzeichnungskoeffizienten `Kappa`, der Bildbreite `ImageWidth` und der Bildhöhe `ImageHeight`. Die Kameraparameter werden in `CameraParam` zurückgegeben. Die Parameter `CameraMatrix` und `Kappa` können mit `stationary_camera_self_calibration` bestimmt werden. `cam_mat_to_cam_par` konvertiert diese Darstellung der internen Kameraparameter dann in die Darstellung, die von `camera_calibration` verwendet wird. Dabei kann die Konversion nur erfolgen, falls in `stationary_camera_self_calibration` die Schrägstellung der Bildachsen auf 0 gesetzt wird, also der Parameter *'skew' nicht* bestimmt wird.

Parameter

- ▷ **CameraMatrix** (input_control) . . . . . hom_mat2d ⇝ *real*
  3 × 3 projektive Kameramatrix, die die internen Kameraparameter bestimmt.
- ▷ **Kappa** (input_control) . . . . . number ⇝ *real*
  Kappa.
- ▷ **ImageWidth** (input_control) . . . . . extent.x ⇝ *integer*
  Breite zu CameraMatrix gehörigen Bilder.
  Restriktion: ImageWidth > 0
- ▷ **ImageHeight** (input_control) . . . . . extent.y ⇝ *integer*
  Höhe zu CameraMatrix gehörigen Bilder.
  Restriktion: ImageHeight > 0
- ▷ **CameraParam** (output_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter.

Beispiel

```
* For the input data to stationary_camera_self_calibration, please
* refer to the example for stationary_camera_self_calibration.
stationary_camera_self_calibration (4, 640, 480, 1, From, To, \
                                    HomMatrices2D, Rows1, Cols1, \
                                    Rows2, Cols2, NumMatches, \
                                    'gold_standard', \
                                    ['focus','principal_point','kappa'], \
                                    'true', CameraMatrix, Kappa, \
                                    RotationMatrices, X, Y, Z, Error)
cam_mat_to_cam_par (CameraMatrix, Kappa, 640, 480, CameraParam)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert cam_mat_to_cam_par den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

stationary_camera_self_calibration

Siehe auch

camera_calibration, cam_par_to_cam_mat

Modul

Calibration

cam_par_to_cam_mat (: : CameraParam : CameraMatrix, ImageWidth, ImageHeight)

Berechnen einer Kameramatrix aus internen Kameraparametern.

cam_par_to_cam_mat berechnet die Kameramatrix CameraMatrix sowie die Bildbreite ImageWidth und die Bildhöhe ImageHeight aus internen Kameraparametern CameraParam. Die internen Kameraparameter CameraParam können mit camera_calibration bestimmt werden. cam_par_to_cam_mat konvertiert diese Darstellung der internen Kameraparameter dann in die Darstellung, die von stationary_camera_self_calibration verwendet wird. Dabei kann die Konversion nur erfolgen, falls die Kamera eine Lochkamera mit Flächensensor ist und die Verzeichnungskoeffizienten in CameraParam 0 sind. Falls nötig muss change_radial_distortion_cam_par verwendet werden, um die Verzeichnungskoeffizienten auf 0 zu setzen.

Parameter

- ▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **CameraMatrix** (output_control) . . . . . hom_mat2d ⇝ *real*
  3×3 projektive Kameramatrix, die CameraParam entspricht.
- ▷ **ImageWidth** (output_control) . . . . . extent.x ⇝ *integer*
  Breite zu CameraMatrix gehörigen Bilder.
  Zusicherung: ImageWidth > 0
- ▷ **ImageHeight** (output_control) . . . . . extent.y ⇝ *integer*
  Höhe zu CameraMatrix gehörigen Bilder.
  Zusicherung: ImageHeight > 0

Beispiel

```
* For the input data to calibrate_cameras, please refer to the
* example for calibrate_cameras.
calibrate_cameras (CalibDataID, Error)
get_calib_data (CalibDataID, 'camera', 0, 'params', CameraParam)
cam_par_to_cam_mat (CameraParam, CameraMatrix, ImageWidth, ImageHeight)

* Alternatively, the following calls can be used.
change_radial_distortion_cam_par ('adaptive', CameraParam, 0, CamParamOut)
cam_par_to_cam_mat (CamParamOut, CameraMatrix, ImageWidth, ImageHeight)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert cam_par_to_cam_mat den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

camera_calibration

Siehe auch

stationary_camera_self_calibration, cam_mat_to_cam_par

Modul

Calibration

deserialize_cam_par (: : SerializedItemHandle : CameraParam)

Deserialisieren von serialisierten internen Kameraparametern.

deserialize_cam_par deserialisiert die internen Kameraparameter, welche mit serialize_cam_par serialisiert wurden (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierten Kameraparameter werden in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Tupel mit dem Handle CameraParam gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **CameraParam** (output_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_cam_par` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_cam_par`

Modul

Foundation

read_cam_par (: : CamParFile : CameraParam)

Lesen von internen Kameraparametern aus einer Datei.

`read_cam_par` liest interne Kameraparameter aus der Datei `CamParFile` ein. Die Datei muss mit `write_cam_par` geschrieben worden sein.

Die Dateiendung für die Kameraparameter ist in HALCON 'dat'.

Die Anzahl der Werte in `CameraParam` richtet sich nach dem Kameratyp. Siehe die Beschreibung von `set_calib_data_cam_param` für eine Liste der Werte und Kalibrierung für detaillierte Informationen über Kameratypen und Kameraparameter.

Parameter

▷ **CamParFile** (input_control) ..... filename.read ⇝ *string*
Dateiname der Kameraparameterdatei.
Defaultwert: 'campar.dat'
Werteliste: `CamParFile` ∈ {'campar.dat', 'campar.initial', 'campar.final'}
Dateiendung: `.dat`

▷ **CameraParam** (output_control) ..... campar ⇝ *real* / integer / string
Interne Kameraparameter.

Beispiel

```
* Create sample camera parameters and write them to file.
gen_cam_par_area_scan_division (0.01, -731, 5.2e-006, 5.2e-006, \
                                654, 519, 1280, 1024, CameraParamTmp)
write_cam_par (CameraParamTmp, 'campar_tmp.dat')
* Read internal camera parameters.
read_cam_par('campar_tmp.dat', CameraParam)
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich gelesen werden, dann liefert `read_cam_par` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`find_marks_and_pose`, `sim_caltab`, `gen_caltab`, `disp_caltab`, `camera_calibration`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`, `write_cam_par`, `write_pose`, `read_pose`, `project_3d_point`, `get_line_of_sight`

Modul

Foundation

serialize_cam_par (: : CameraParam : SerializedItemHandle)

Serialisieren von internen Kameraparametern.

`serialize_cam_par` serialisiert die Daten von internen Kameraparametern (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_cam_par` in eine Datei geschrieben werden. Die Kameraparameter werden in dem Handle `CameraParam` übergeben. Die serialisierten Kameraparameter werden in dem Handle `SerializedItemHandle` zurückgegeben und können mit `deserialize_cam_par` wieder deserialisiert werden.

Parameter

- ▷ **CameraParam** (input_control) ..... campar ⇝ *real / integer / string*
  Interne Kameraparameter.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_cam_par` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_cam_par`

Modul

Foundation

write_cam_par (: : CameraParam, CamParFile :)

Speichern der internen Kameraparameter in eine Datei.

`write_cam_par` schreibt die internen Kameraparameter `CameraParam` in die Datei `CamParFile`.

Die Anzahl der Werte in `CameraParam` richtet sich nach dem Kameratyp. Siehe die Beschreibung von `set_calib_data_cam_param` für eine Liste der Werte und Kalibrierung für detaillierte Informationen über Kameratypen und Kameraparameter.

Die Dateiendung für die Kameraparameter ist in HALCON 'dat'.

Die internen Kameraparameter können später mit dem Operator `read_camera_setup_model` gelesen werden.

Parameter

▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
Interne Kameraparameter.

▷ **CamParFile** (input_control) . . . . . filename.write ⇝ *string*
Dateiname der Kameraparameterdatei.
Defaultwert: 'campar.dat'
Werteliste: `CamParFile` ∈ {'campar.dat', 'campar.initial', 'campar.final'}
Dateiendung: `.dat`

Beispiel

```
*
* Calibrate the camera.
*
StartCamPar := ['area_scan_division', 0.016, 0, 0.0000074, 0.0000074, \
                326, 247, 652, 494]
create_calib_data ('calibration_object', 1, 1, CalibDataID)
set_calib_data_cam_param (CalibDataID, 0, [], StartCamPar)
set_calib_data_calib_object (CalibDataID, 0, 'caltab_30mm.descr')
NumImages := 10
for I := 1 to NumImages by 1
  read_image (Image, '3d_machine_vision/calib/calib_' + I$'02d')
  find_calib_object (Image, CalibDataID, 0, 0, I, [], [])
  get_calib_data_observ_contours (Caltab, CalibDataID, 'caltab', 0, 0, I)
endfor
calibrate_cameras (CalibDataID, Error)
get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)
* Write the internal camera parameters to a file.
write_cam_par (CamParam, 'camera_parameters.dat')
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich geschrieben werden, dann liefert `write_cam_par` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`camera_calibration`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`, `read_cam_par`, `write_pose`, `read_pose`, `project_3d_point`, `get_line_of_sight`

Modul

Foundation

14.6 Mehrbild

Dieses Kapitel beschreibt den Kalibriervorgang für verschiedene Mehrbild-Kameraaufbauten.

Um eine höchstmögliche Messgenauigkeit zu erreichen muss ein Kamerasystem entsprechend kalibriert werden. Im Gegensatz zu einem Kameraaufbau mit einer Kamera gelten für einen Mehrbild-Kameraaufbau einige zusätzliche Anforderungen. Die folgenden Abschnitte enthalten Informationen über die Kalibrierung von Mehrbild-Kameraaufbauten. Für generelle Informationen zur Kalibrierung von Kameras siehe die Kapitelreferenz Kalibrierung.

Vorbereitung der Eingabekalibrierdaten

Vor dem Aufruf des Operators `calibrate_cameras` sollten die benötigten Daten im Kalibrierdatenmodell gesammelt werden (wie in Kalibrierung beschrieben). Für Mehrbild-Kameraaufbauten sollten diese zusätzlichen Aspekte berücksichtigt werden:

- Die Anzahl an Kameras und die Anzahl an Kalibrierobjekten für den Aufbau kann mit `create_calib_data` gesetzt werden.
- Beim Spezifizieren des Kameratyps mit `set_calib_data_cam_param` ist zu beachten, dass nur Kameras vom selben Typ (d.h. Flächen- oder Zeilenkameras) in einem einzelnen Aufbau kalibriert werden können.
- Mit `set_calib_data` kann der Kalibriervorgang konfiguriert werden, z.B. durch die Bestimmung der Referenzkamera. Dabei können Posen von Kalibrierobjekten oder auch Parameter sowohl für den gesamten Kameraaufbau als auch für einzelne Kameras gesetzt werden.

Die tatsächliche Kalibrierung durchführen

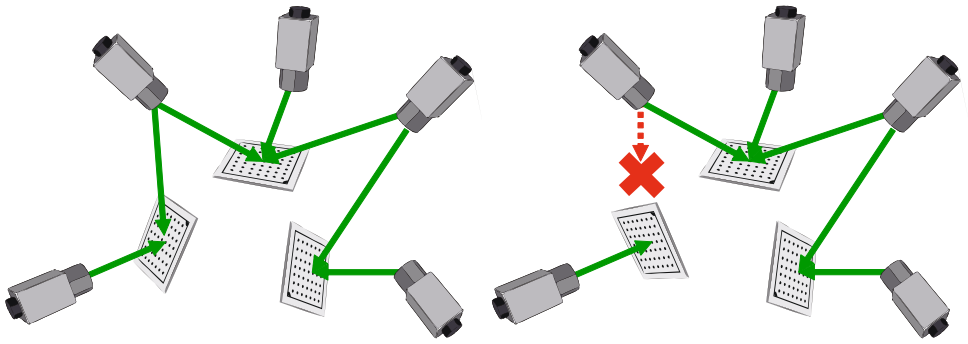
Je nach Kameratypen, welche im Aufbau kalibriert werden, führt `calibrate_cameras` die Kalibrierung auf unterschiedliche Weisen durch. Während sich unterschiedliche Kameraaufbauten zum Teil in ihren Anforderungen an die Bildaufnahme unterscheiden, sind die grundsätzlichen Schritte des Kalibriervorgangs für Aufbauten mit projektiven und/oder telezentrischen Kameras gleich:

1. **Kette von Beobachtungen bilden:** Im ersten Schritt versucht der Operator `calibrate_cameras` eine durchgehende Kette von Beobachtungsposen zu bilden, welche alle Kameras und Kalibrierobjektposen miteinander verbindet. Je nach Kameraaufbau unterscheiden sich die Bedingungen für eine gültige Kette von Posen. Die entsprechenden Anforderungen an die speziellen Aufbauten werden in den entsprechenden Abschnitten unterhalb dieses Paragraphs erläutert.

   Falls eine Kamera nicht erreicht werden kann (d.h. die Kamera beobachtet keine Kalibrierkörperpose, die in der Kette eingegliedert ist), wird der Kalibrierprozess abgebrochen und ein Fehler zurückgegeben. Ansonsten initialisiert der Algorithmus die Posen aller Objekte im Aufbau beim Durchlaufen der Kette.
2. **Erste Optimierung:** In diesem Schritt optimiert `calibrate_cameras` alle Kamera- und Kalibrierkörperposeparameter, die nicht explizit von der Optimierung ausgeschlossen wurden.
3. **Zweite Optimierung:** Hierbei korrigiert der Algorithmus anhand des soweit kalibrierten Kameraaufbaus jene Beobachtungen, die Konturdaten enthalten (siehe `find_calib_object`), und kalibriert den Aufbau zur Berücksichtigung der Korrekturen erneut. Sollten keine Konturbeobachtungsdaten im Modell vorhanden sein, wird dieser Schritt übersprungen.
4. **Berechnung der Qualität der Parameterschätzung:** Im letzten Schritt berechnet `calibrate_cameras` die Standardabweichungen und die Kovarianzen der kalibrierten Kameraparameter.

Die folgenden Abschnitte enthalten weitere Details zur Kalibrierung verschiedener Kameraaufbauten. Außerdem werden Informationen zur Kalibrierung von Zeilenkameras aufgeführt.

Projektive Flächenkameras Für einen Aufbau mit projektiven Flächenkameras wird die Kalibrierung in den oben erläuterten vier Schritten durchgeführt. Im ersten Schritt versucht der Algorithmus dabei eine Kette von Beobachtungsposen zu bilden, welche alle Kameras und Kalibrierobjektposen verbindet (siehe Abbildung).



(1) (2)

(1) Alle Kameras können über eine Kette von Beobachtungen miteinander verbunden werden. (2) Die linke Kamera ist von der Beobachtungskette ausgeschlossen, da die linke Kalibrierplatte von keiner anderen Kamera gesehen wird.

Mögliche projektive Flächenkameras sind:

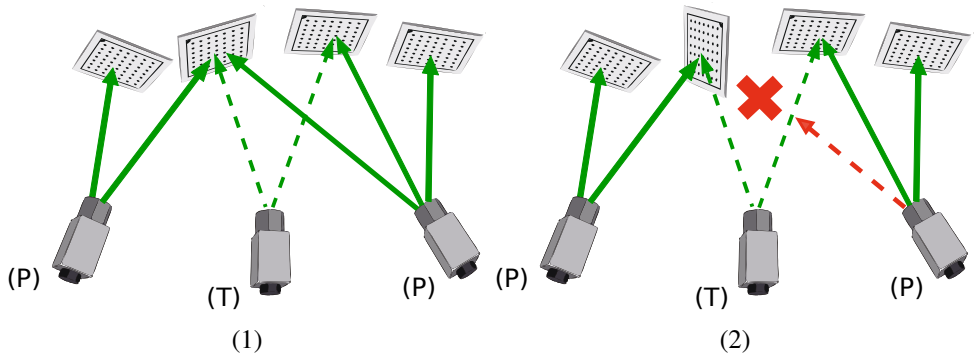
- *'area_scan_division'*
- *'area_scan_polynomial'*
- *'area_scan_tilt_division'*
- *'area_scan_tilt_polynomial'*
- *'area_scan_tilt_image_side_telecentric_division'*
- *'area_scan_tilt_image_side_telecentric_polynomial'*
- *'area_scan_hypercentric_division'*
- *'area_scan_hypercentric_polynomial'*

Telezentrische Flächenkameras Für einen Aufbau mit telezentrischen Flächenkameras werden dieselben vier Schritte durchlaufen wie oben aufgelistet. Für den ersten Schritt (Aufbau einer Kette von Beobachtungen, die alle Kameras und Kalibrierobjektposen verbindet) gelten hier jedoch zusätzliche Voraussetzungen. Da die Lage eines Objektes mit nur einer Kamera nur bis auf eine Verschiebung entlang der optischen Achse bestimmt werden kann, muss ein Kalibrierkörper von mindestens zwei Kameras beobachtet werden, damit seine relative Lage bestimmt werden kann. Sonst wird er aus der Kalibrierung entfernt. Außerdem gilt: Da ein planares Kalibrierobjekt aus jeweils zwei Winkeln gleich erscheint, kann die relative Lage der Kameras untereinander nicht eindeutig bestimmt werden. Es gibt immer zwei gültige Varianten der relativen Lage der zweiten Kamera. Beide Alternativen stellen ein konsistentes Kamerasetup dar und können zum Messen benutzt werden. Da die Mehrdeutigkeit nicht aufgelöst werden kann, wird die erste Lösung zurückgeliefert. Wenn die zurückgegebene Pose nicht die reell vorliegende sondern die alternative Lösung ist, so sind die damit rekonstruierten Daten gespiegelt.

Mögliche telezentrische Flächenkameras sind:

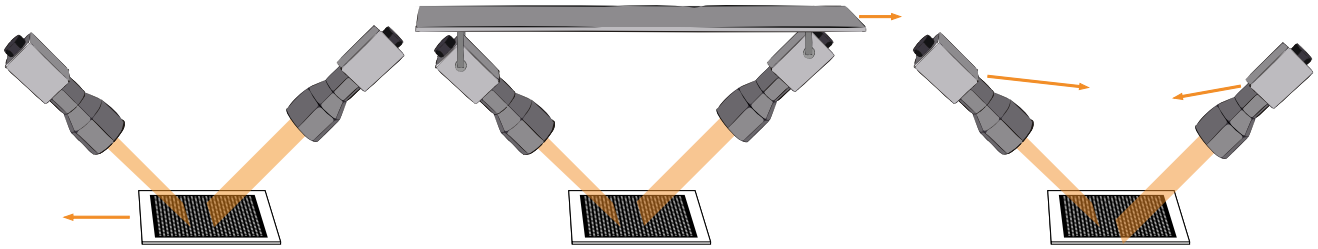
- *'area_scan_telecentric_division'*
- *'area_scan_telecentric_polynomial'*
- *'area_scan_tilt_bilateral_telecentric_division'*
- *'area_scan_tilt_bilateral_telecentric_polynomial'*
- *'area_scan_tilt_object_side_telecentric_division'*
- *'area_scan_tilt_object_side_telecentric_polynomial'*

Projektive and telezentrische Flächenkameras Für einen gemischten Aufbau mit projektiven und telezentrischen Flächenkameras durchläuft der Algorithmus dieselben Schritte wie oben aufgelistet. Mögliche Mehrdeutigkeiten während des ersten Schrittes (Aufbau einer Kette von Beobachtungen, die alle Kameras und Kalibrierobjektposen verbindet), die bei einem reinen telezentrischen Flächenkameraaufbau auftreten, können bei einem gemischten Aufbau verhindert werden. Voraussetzung dafür ist die Existenz einer Kette, die aus allen projektiven Flächenkameras und einer ausreichenden Anzahl an Kalibrierobjekten besteht. Hierbei bedeutet eine ausreichende Anzahl, dass jede telezentrische Flächenkamera mindestens zwei Kalibrierobjekte dieser Kette beobachtet.



Gemischter Aufbau mit projektiven (P) und telezentrischen (T) Flächenkameras. (1) Alle projektiven Kameras sind über eine Kette von Beobachtungen, welche nur projektive Kameras beinhaltet, verbunden. (2) Die zweite Kalibrierplatte von links ist von der rechten projektiven Kamera nicht sichtbar, weshalb die Lage der projektiven Kameras zueinander nicht eindeutig bestimmt werden kann.

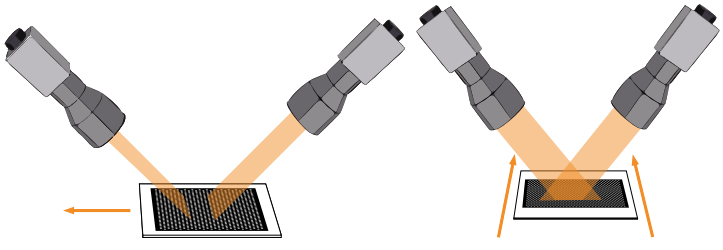
Zeilenkameras Aufbauten mit telezentrischen Zeilenkameras (*'line_scan_telecentric'*) verhalten sich identisch wie Aufbauten mit telezentrischen Flächenkameras und es gelten dieselben Einschränkungen und Mehrdeutigkeiten, die oben beschrieben sind. Für diese Art von Aufbau können zwei Konfigurationen unterschieden werden. In der ersten Konfiguration sind alle Kameras starr und unbeweglich montiert und das Objekt wird linear vor den Kameras bewegt. Alternativ dazu können alle Kameras starr zueinander montiert sein und mit demselben Linearmechanismus über das Objekt bewegt werden. In beiden Fällen besitzen alle Kameras denselben Bewegungsvektor, der im Kamerakoordinatensystem der Referenzkamera modelliert wird und mittels des Rotationsteils der Pose der jeweiligen Kamera in das Koordinatensystem aller anderen Kameras transformiert wird. Diese Konfiguration wird als Voreinstellung angenommen. In der zweiten Konfiguration werden die Kameras durch unabhängige Linearmechanismen in unterschiedliche Richtungen bewegt. In diesem Fall besitzt jede Kamera einen unabhängigen Bewegungsvektor. Die Art des Aufbaus kann mit `set_calib_data` festgelegt werden.



Es können verschiedenen Konfigurationen für Aufbauten mit telezentrischen Zeilenkameras unterschieden werden: (1) Falls die Kameras fest montiert sind und das Objekt linear darunter bewegt wird, (2) oder falls die Kameras linear über das Objekt bewegt werden, ohne ihre Positionen relativ zueinander zu verändern, muss nur ein einzelner Bewegungsvektor berechnet werden. (3) Andernfalls, falls sich die Kameras unabhängig voneinander bewegen, wird für jede Kamera ein entsprechender Bewegungsvektor bestimmt.

Es sind zwei verschiedene Aufbauten für Stereosysteme, die Zeilenkameras mit telezentrischen Objektiven verwenden, üblich. Für beide Typen von Aufbauten wird vorausgesetzt, dass sich Objekt beziehungsweise das Kamerasystem linear und mit konstanter Geschwindigkeit bewegt.

- Bei Along-Track-Setups (Aufbauten entlang der Bewegungsrichtung) wird die erste Kamera vorne, mit Blickrichtung nach hinten montiert, während die zweite Kamera dahinter, mit Blickrichtung nach vorne platziert wird. Dabei sollte die Kameraausrichtung jeweils schräg zur Bewegungsrichtung sein.
- Die Kameras bei einem Across-Track-Setup (Aufbau quer zur Bewegungsrichtung) sind alle rechtwinklig zur Bewegungsrichtung angebracht, wobei die Sichtebenen möglichst koplanar sein sollten. Dadurch ist allerdings die Tiefenschärfe des Systems eingeschränkt, wodurch dieser Ansatz nur für Messungen im Schnittbereich der jeweiligen Tiefenschärfebereiche geeignet ist.



Stereo-Aufbauten für telezentrische Zeilenkameras: (1) Aufbau entlang der Bewegungsrichtung (Along-Track Setup) und (2) Aufbau quer zur Bewegungsrichtung (Across-Track Setup).

Für projektive Zeilenkameras (*'line_scan'*) existieren folgende Beschränkungen: Es kann nur eine Kamera pro Aufbau kalibriert werden und nur ein Kalibrierobjekt pro Aufbau verwendet werden.

Schließlich muss beachtet werden, dass bei Kalibrierplatten mit rechteckiger Markenanordnung (siehe `gen_caltab`) keine Beobachtungen akzeptiert werden, in denen nicht alle Kalibrierkörpermarken extrahiert wurden. Bei Kalibrierplatten mit hexagonaler Markenanordnung (siehe `create_caltab`) gilt diese Einschränkung nicht. Weitere Informationen zu Kalibrierplatten und der Aufnahme von Kalibrierbildern finden sich im Abschnitt „Zusätzliche Information über die Kalibrierung“ des Kapitels Kalibrierung.

Ergebnisse der Kalibrierung überprüfen

Bei der gleichzeitigen Kalibrierung mehrerer Kameras ist der Wert des Fehlers schwieriger zu beurteilen als bei einer einzelnen Kamera. Als Faustregel sollte der Fehler so klein wie möglich sein oder zumindest kleiner als 1.0 sein, da dies ein Hinweis ist, dass mit den kalibrierten Parametern eine subpixelgenaue Evaluierung der Daten noch möglich ist. Bei bestimmten Konfigurationen kann es schwierig sein, diesen Wert zu erreichen. Eine weitere Analyse der Kalibrierqualität kann anhand der zurückgegebenen Standardabweichungen und Kovarianzen der Kameraparameter erfolgen.

Kalibrierergebnisse abfragen

Alle Ergebnisse der Kalibrierung, d.h. die internen Kameraparameter, die Kameraposen (externe Parameter), die Kalibrierkörperposen usw., können mit `get_calib_data` abgefragt werden.

Es ist zu beachten, dass bei telezentrischen Kameras die Lage nur bis auf ihre Verschiebung entlang der z-Achse des Koordinatensystems der jeweiligen Kamera bekannt ist. Daher werden die Kameraposen entlang dieser Richtung so verschoben, dass alle Kameras auf einer gemeinsamen Kugel liegen. Der Mittelpunkt der Kugel wird in den ersten Kalibrierkörper gelegt. Der Radius dieser Kugel variiert je nach gegebenem Aufbau. Falls ein gemischter Aufbau mit projektiven und telezentrischen Kameras vorliegt, wird als Radius das Maximum über alle Abstände zwischen projektiven Kameras und dem ersten Kalibrierkörper gewählt. Falls nur telezentrische Kameras im Aufbau vertreten sind, beträgt der Radius 1m.

`calibrate_cameras` (: : `CalibDataID` : `Error`)

Bestimmen aller Kameraparameter durch simultane Ausgleichsrechnung.

Der Operator `calibrate_cameras` berechnet die internen und externen Kameraparameter eines Kalibrierdatenmodells `CalibDataID`. Das Kalibrierdatenmodell beschreibt einen Aufbau mit einer oder mehreren Kameras und wird beim Erstellen des Modells festgelegt. Detaillierte Informationen zum Kalibriervorgang sind im Kapitel Kalibrierung zu finden.

Nach erfolgreicher Kalibrierung gibt der Operator `calibrate_cameras` die Wurzel des mittleren quadratischen Rückprojektionsfehlers (RMSE) der Optimierung im Parameter `Error` (in Pixel) zurück. Dieser Fehler gibt einen allgemeinen Hinweis darauf, ob die Optimierung erfolgreich war. Weitere Informationen siehe Abschnitt „Ergebnisse der Kalibrierung überprüfen“ im oben erwähnten Kapitel.

Parameter

▷ **`CalibDataID`** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **`Error`** (output_control) . . . . . number ⇝ *real*
Durchschnittlicher verbleibender Fehler nach der Optimierung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_calib_data`, `set_calib_data_cam_param`, `set_calib_data_calib_object`, `set_calib_data_observ_points`, `find_calib_object`, `set_calib_data`, `remove_calib_data_observ`

Nachfolger

`get_calib_data`

Literatur

Carsten Steger: „A Comprehensive and Versatile Camera Model for Cameras with Tilt Lenses“; International Journal of Computer Vision, vol. 123, no. 2, pp. 121-159, 2017.

Carsten Steger, Markus Ulrich, Christian Wiedemann: „Machine Vision Algorithms and Applications“; Wiley-VCH, Weinheim, 2nd Edition, 2018.

Markus Ulrich, Carsten Steger: „A Camera Model for Cameras with Hypercentric Lenses and Some Example Applications“; Machine Vision and Applications, vol. 30, no. 6, pp. 1013-1028, 2019.

Modul

Calibration

clear_calib_data (: : CalibDataID :)

Freigeben des Speichers eines Kalibrierdatenmodells.

`clear_calib_data` gibt den Speicher eines Kalibrierdatenmodells, das mit `create_calib_data` oder `read_calib_data` angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `CalibDataID` ist nach dem Aufruf ungültig.

Parameter

▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Modul

Calibration

clear_camera_setup_model (: : CameraSetupModelID :)

Freigeben des Speichers eines Kameraaufbaumodells.

`clear_camera_setup_model` gibt den Speicher eines Kameraaufbaumodells, das mit `create_camera_setup_model`, `read_camera_setup_model` oder als Ausgabe einer Kamerakalibrierung vom Operator `get_calib_data` angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `CameraSetupModelID` ist nach dem Aufruf ungültig.

Parameter

▷ **CameraSetupModelID** (input_control) . . . . . camera_setup_model ⇝ *handle*
Handle des Kameraaufbaumodells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CameraSetupModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Modul

Calibration

create_calib_data (: : CalibSetup, NumCameras,
NumCalibObjects : CalibDataID)

Anlegen eines generischen HALCON-Kalibrierdatenmodells.

Der Operator create_calib_data erstellt ein generisches Modell für Kalibrierdaten. Das Modell speichert:

- Beschreibung des Kameraaufbaus,
- Einstellungen des Kalibrierverfahrens,
- Kalibrierdaten, und
- Ergebnisse der Kamerakalibrierung oder der Hand-Auge-Kalibrierung.

Im Parameter CalibSetup wird der generische Typ des Kalibrieraufbaus gesetzt. Derzeit werden fünf Typen unterstützt. Ein Modell vom Typ *'calibration_object'* wird benutzt um die internen Kameraparameter und -posen von einer oder mehreren Kameras anhand der metrische Information, die aus Beobachtungen von Kalibrierkörpern gemessen wird, zu kalibrieren.

Ein Modell vom Typ *'hand_eye_moving_cam'*, *'hand_eye_stationary_cam'*, *'hand_eye_scara_moving_cam'* oder *'hand_eye_scara_stationary_cam'* wird benutzt um eine Hand-Auge-Kalibrierung basierend auf Beobachtungen eines Kalibrierkörpers und den zugehörigen Posen des Robotertools im Roboterbasiskoordinatensystem durchzuführen. Diese vier zuletzt genannten Modelle unterscheiden zum einen, ob die Kamera oder der Kalibrierkörper vom Roboter bewegt wird und zum anderen, ob ein anthropomorpher oder ein SCARA-Roboter kalibriert wird. Der Arm eines anthropomorphen Roboters hat 3 Drehgelenke, die typischerweise 6 Freiheitsgrade abdecken (3 Translationen und 3 Rotationen). SCARA-Roboter besitzen dagegen nur zwei parallele Drehgelenke und ein paralleles Schubgelenk, die nur 4 Freiheitsgrade (3 Translationen und 1 Rotation) abdecken. Grob gesagt ist ein anthropomorpher Roboter im Gegensatz zum SCARA-Roboter in der Lage, seinen Endeffektor zu verkippen.

NumCameras setzt die Anzahl der Kameras ein, die gleichzeitig im Aufbau kalibriert werden. NumCalibObjects setzt die Anzahl der Kalibrierkörper ein, die von den Kameras beobachtet werden. Bei Kamerakalibrierungen ist zu beachten, dass für Zeilenkameras mit perspektivischen Objektiven nur ein Kalibrierkörper unterstützt wird (NumCalibObjects=1). Bei Hand-Auge-Kalibrierungen ist zu beachten, dass derzeit nur zwei Setups unterstützt werden: eine Kamera (NumCameras=1) und ein Kalibrierkörper (NumCalibObjects=1) oder ein allgemeiner Sensor ohne Kalibrierobjekt (NumCameras=0, NumCalibObjects=0).

Der Ausgabeparameter CalibDataID gibt einen Handle des neuen Kalibrierdatenmodell zurück. Dieser Handle wird im Anschluss weiteren Operatoren übergeben, um die Aufbaubeschreibung, die Kalibriereinstellungen und -data ins Modell zu speichern. Für Kamerakalibrierungen wird der Handle in calibrate_cameras eingegeben. Dieser führt die eigentliche Kamerakalibrierung aus und speichert die Kalibrierergebnisse wiederum

im Modell. Für weitere Information über den Kalibrierprozess, siehe das Kapitel Kalibrierung. Für Hand-Auge-Kalibrierungen, wird der Handle an `calibrate_hand_eye` weitergegeben. Dieser führt die eigentliche Hand-Auge-Kalibrierung durch und speichert die Kalibrierergebnisse wiederum im Modell. Für weitere Information über den Kalibrierprozess, siehe die Beschreibung vom Operator `calibrate_hand_eye`.

Parameter

- ▷ **`CalibSetup`** (input_control) . . . . . string ⇝ *string*
  Generischer Typ des Kalibrieraufbaus.
  Defaultwert: 'calibration_object'
  Werteliste: `CalibSetup` ∈ {'calibration_object', 'hand_eye_moving_cam', 'hand_eye_stationary_cam', 'hand_eye_scara_moving_cam', 'hand_eye_scara_stationary_cam'}
- ▷ **`NumCameras`** (input_control) . . . . . number ⇝ *integer*
  Gesetzte Anzahl der Kameras im Kalibrieraufbau.
  Defaultwert: 1
  Restriktion: NumCameras >= 0
- ▷ **`NumCalibObjects`** (input_control) . . . . . number ⇝ *integer*
  Gesetzte Anzahl der Kalibrierkörper im Kalibrieraufbau.
  Defaultwert: 1
  Restriktion: NumCalibObjects >= 0
- ▷ **`CalibDataID`** (output_control) . . . . . calib_data ⇝ *handle*
  Handle des Kalibriermodells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_calib_data_cam_param`, `set_calib_data_calib_object`

Modul

Calibration

`create_camera_setup_model` (: : NumCameras : CameraSetupModelID)

Anlegen eines HALCON-Kameraaufbaumodells.

Der Operator `create_camera_setup_model` erstellt ein neues Kameraaufbaumodell und gibt seinen Handle in `CameraSetupModelID` zurück. Die Anzahl von Kameras im Aufbau wird durch den Parameter `NumCameras` festgelegt und kann im Nachhinein nicht mehr geändert werden. Das Modell speichert interne Parameter und die Standardabweichungen der Parameter (optional) für jede Kamera, sowie die Pose der Kamera.

Das Koordinatensystem, in dem die Kameraposen angegeben werden, kann mit `set_camera_setup_param` verändert werden: Entweder man wählt eine Kamera, dann werden die Posen der anderen Kameras relativ zu der gewählten Kamera angegeben. Oder man übergibt eine allgemeine Transformation und verschiebt damit das Koordinatensystem des Aufbaus in eine beliebige Lage. Letzteres ist insbesondere dann nützlich, wenn man die Kameras beispielsweise im Koordinatensystem eines Objekts darstellen will, das von den Kameras beobachtet wird (siehe auch das unten folgende Beispiel).

Die internen Kameraparameter und ihre Pose werden gesetzt, bzw. geändert mit dem Operator `set_camera_setup_cam_param`, weitere Kameraparameter mit dem Operator `set_camera_setup_param`. Alle Kameraparameter können mit dem Operator `get_camera_setup_param` abgefragt werden.

Ein Kameraaufbaumodell kann mit dem Operator `write_camera_setup_model` in eine Datei gespeichert werden. Im Nachhinein kann es mit dem Operator `read_camera_setup_model` wieder gelesen werden.

Parameter

▷ **NumCameras** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Kameras im Kalibrieraufbau.
Defaultwert: 2
Wertevorschläge: NumCameras ∈ {1, 2, 3, 4}
Restriktion: NumCameras >= 1

▷ **CameraSetupModelID** (output_control) . . . . . camera_setup_model ⇝ *handle*
Handle des Kameraaufbaumodells.

Beispiel

```
* Create camera setup of three cameras.
create_camera_setup_model (3, CameraSetupModelID)
gen_cam_par_area_scan_division (0.006, 0, 8.3e-6, 8.3e-6,\
              512, 384, 1024, 768, StartCamPar)
* Camera 0 is located in the origin.
set_camera_setup_cam_param (CameraSetupModelID, 0, [], StartCamPar,\
                            [0, 0, 0, 0, 0, 0, 0])

* Camera 1 is shifted 0.07 m in positive x-direction relative to
* camera 0.
set_camera_setup_cam_param (CameraSetupModelID, 1, [], StartCamPar,\
                            [0.07, 0, 0, 0, 0, 0, 0])

* Camera 2 is shifted 0.1 m in negative y-direction relative to
* camera 0.
set_camera_setup_cam_param (CameraSetupModelID, 2, [], StartCamPar,\
                            [0.0, -0.1, 0, 0, 0, 0, 0])

* There is an object, which is 0.5 away from the origin in
* z-direction, and is facing the origin.
ObjectPose := [0, 0, 0.5, 180, 0, 0, 0]
* Place the setup's origin in the object.
set_camera_setup_param (CameraSetupModelID, 'general', 'coord_transf_pose',\
                        ObjectPose)

* Now the camera poses are given relative to the object.
get_camera_setup_param (CameraSetupModelID, 0, 'pose', CamPose0)
* CamPose0 is equivalent to [0.0, 0.0, 0.5, 180.0, 0.0, 0.0, 0]

get_camera_setup_param (CameraSetupModelID, 1, 'pose', CamPose1)
* CamPose1 is equivalent to [0.07, 0.0, 0.5, 180.0, 0.0, 0.0, 0]

get_camera_setup_param (CameraSetupModelID, 2, 'pose', CamPose2)
* CamPose2 is equivalent to [0.0, 0.1, 0.5, 180.0, 0.0, 0.0, 0]
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

set_camera_setup_param

Modul

Calibration

deserialize_calib_data (: : SerializedItemHandle : CalibDataID)

Deserialisieren eines serialisierten Kalibrierdatenmodells.

deserialize_calib_data deserialisiert ein Kalibrierdatenmodell, welches mit serialize_calib_data serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Kalibrierdatenmodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Kalibrierdatenmodell mit dem Handle CalibDataID gespeichert.

Beachten Sie, dass serialize_calib_data keine Kalibrierergebnisse serialisiert. Für ein vollständig konfiguriertes Modell kann allerdings calibrate_cameras direkt nach der Deserialisierung aufgerufen werden und die Kalibrierergebnisse sind danach wieder abfragbar.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **CalibDataID** (output_control) ..... calib_data ⇝ *handle*
  Handle des Kalibrierdatenmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_calib_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_calib_data

Modul

Calibration

deserialize_camera_setup_model (
: : SerializedItemHandle : CameraSetupModelID)

Deserialisieren eines serialisierten Kameraaufbaumodells.

deserialize_camera_setup_model deserialisiert ein Kameraaufbaumodell, welches mit serialize_camera_setup_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Kameraaufbaumodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Kameraaufbaumodell mit dem Handle CameraSetupModelID gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **CameraSetupModelID** (output_control) ..... camera_setup_model ⇝ *handle*
  Handle des Kameraaufbaumodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_camera_setup_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_camera_setup_model`

Modul

Calibration

get_calib_data (: : CalibDataID, ItemType, ItemIdx, DataName : DataValue)

Abfragen der Kalibrierdaten, -einstellungen und -ergebnisse eines Kalibrierdatenmodells.

Mit dem Operator `get_calib_data` können Kalibrierdaten aus dem Kalibrierdatenmodell `CalibDataID` abgefragt werden.

Es ist zu beachten, dass im Folgenden alle 3D-Pose-Ergebnisse relativ zur *Referenzkamera* des Modells zurückgegeben werden. Die Referenzkamera wird mit dem Operator `set_calib_data` eingestellt und mit dem Operator `get_calib_data` abgefragt. Standardmäßig wird die erste Kamera (Kameraindex 0) als Referenzkamera benutzt.

Das Kalibrierdatenmodell beinhaltet verschiedene Arten von Daten. Wie die einzelnen Daten abgefragt werden, wird für verschiedene Kategorien von Daten beschrieben:

- Modelldaten (`ItemType` *'model'*)
- Kameradaten (`ItemType` *'camera'*)
- Kalibrierkörperdaten (`ItemType` *'calib_obj'*)
- Daten zu den Kalibrierkörperposen (`ItemType` *'calib_obj_pose'*)
- Daten zur Hand-Auge-Kalibrierung (verschiedene Werte für `ItemType`)

Bevor die einzelnen abzufragenden Daten im Detail beschrieben werden, erfolgt eine Übersicht über die Daten, die nach den jeweiligen Schritten der Kalibrierprozesse im Modell verfügbar sind. Werden Kameras oder Hand-Auge-Systeme kalibriert, werden verschiedene Operatoren aufgerufen, die Schritt für Schritt das Kalibrierdatenmodell mit Inhalt füllen. Im Folgenden wird pro Prozess für jeden Operator tabellarisch aufgelistet, welche Daten zum Modell zugefügt werden und wie die Werte für `ItemType`, `ItemIdx` und `DataName` kombiniert werden müssen, um die Information mit `get_calib_data` wieder abzurufen. Für die verschiedenen Indices in den Tabellen werden folgende Abkürzungen bzw. mögliche Variablennamen benutzt:

- Kameraindex: `CameraIdx`
- Kalibrierkörperindex: `CalibObjIdx`
- Kalibrierkörperposeindex: `CalibObjPoseIdx`

Detailliertere Beschreibungen zu den abzufragenden Daten finden sich dann in den Abschnitten zu den jeweiligen Datenkategorien.

Ausführliche Informationen über den Kalibriervorgang von Kameraaufbauten befinden sich im Kapitel Kalibrierung.

Inhalt des Kalibrierdatenmodells während der Kamerakalibrierung

`create_calib_data`:

Zum Modell zugefügte Daten	`ItemType`	`ItemIdx`	`DataName`
Typ des Kalibrierdatenmodells	*'model'*	*'general'*	*'type'*
Anzahl der Kameras	*'model'*	*'general'*	*'num_cameras'*
Anzahl der Kalibrierkörper	*'model'*	*'general'*	*'num_calib_objs'*

`set_calib_data_cam_param`:

Zum Modell zugefügte Daten	ItemType	ItemIdx	DataName
Kameratypen	*'camera'*	CameraIdx	*'type'*
Initiale interne Kameraparameter	*'camera'*	CameraIdx	*'init_params'*

set_calib_data_calib_object:

Zum Modell zugefügte Daten	ItemType	ItemIdx	DataName
Anzahl der Kalibriermarken des Kalibrierkörpers	*'calib_obj'*	CalibObjIdx	*'num_marks'*
Koordinaten der Kalibriermarken des Kalibrierkörpers relativ zum Kalibrierkörperkoordinatensystem	*'calib_obj'*	CalibObjIdx	*'x'*, *'y'*, *'z'*

Für Standard-HALCON-Kalibrierplatten werden dem Modell noch weitere Kalibrierplatten-spezifische Informationen zugefügt, die nicht über get_calib_data abgerufen werden können, aber direkt aus der jeweiligen Kalibrierkörperbeschreibungsdatei entnommen werden können (für Details zu den Kalibrierkörperbeschreibungsdateien siehe die Beschreibung von create_caltab für eine Kalibrierplatte mit hexagonal angeordneten Kalibriermarken und gen_caltab für eine Kalibrierplatte mit rechtwinklig angeordneten Kalibriermarken).

find_calib_object (für Standard-HALCON-Kalibrierplatten):

Zum Modell zugefügte Daten	abrufbar mit
Beobachtete Bildkoordinaten der Kalibriermarken	get_calib_data_observ_points
Beobachtete Konturen der Kalibriermarken	get_calib_data_observ_contours
Beobachtete Posen der Kalibrierplatte relativ zum Kamerakoordinatensystem	get_calib_data_observ_pose oder get_calib_data_observ_points

set_calib_data_observ_points (für andere Kalibrierkörper als die Standard-HALCON-Kalibrierplatten):

Zum Modell zugefügte Daten	abrufbar mit
Beobachtete Bildkoordinaten der Kalibriermarken	get_calib_data_observ_points

set_calib_data:

Zum Modell zugefügte Daten	ItemType	ItemIdx	DataName
Referenzkamera	*'model'*	*'general'*	*'reference_camera'*
Interne und externe Kameraparameter, die (zusätzlich) kalibriert werden sollen	*'camera'*	*'general'* oder CameraIdx	*'calib_settings'*
Interne und externe Kameraparameter, die nicht kalibriert werden sollen	*'camera'*	*'general'* oder CameraIdx	*'excluded_settings'*
Für Stereosysteme, die Zeilenkameras mit telezentrischen Objektiven verwenden: Besitzen die beiden Kameras einen **gemeinsamen Bewegungsvektor**?	*'model'*	*'general'*	*'common_motion_vector'*
Kalibrierkörperpose-Parameter, die (zusätzlich) optimiert werden sollen	*'calib_obj_pose'*	*'general'* oder [CalibObjIdx, CalibObjPoseIdx]	*'calib_settings'*
Kalibrierkörperpose-Parameter, die nicht optimiert werden sollen	*'calib_obj_pose'*	*'general'* oder [CalibObjIdx, CalibObjPoseIdx]	*'excluded_settings'*

calibrate_cameras:

Zum Modell zugefügte Daten	ItemType	ItemIdx	DataName
Kameraaufbaumodell (benötigt für eine Mehrbild-Stereorekonstruktion)	*'model'*	*'general'*	*'camera_setup_model'*
Optimierte interne Kameraparameter	*'camera'*	CameraIdx	*'params'*
Standardabweichung der optimierten internen Kameraparameter	*'camera'*	CameraIdx	*'params_deviations'*
Kovarianzmatrizen der optimierten internen Kameraparameter	*'camera'*	CameraIdx	*'params_covariances'*
Liste mit den Namen der internen Kameraparameter	*'camera'*	CameraIdx	*'params_labels'*
Initiale externe Kameraparameter (Kameraposen)	*'camera'*	CameraIdx	*'init_pose'*
Optimierte externe Kameraparameter (Kameraposen)	*'camera'*	CameraIdx	*'pose'*
Liste mit den Namen der externen Kameraparameter (Kameraposen)	*'camera'*	CameraIdx	*'pose_labels'*
Initiale Kalibrierkörperposen	*'calib_obj_pose'*	[CalibObjIdx, CalibObjPoseIdx]	*'init_pose'*
Optimierte Kalibrierkörperposen	*'calib_obj_pose'*	[CalibObjIdx, CalibObjPoseIdx]	*'pose'*
Liste mit den Namen der Kalibrierkörperpose-Parameter	*'calib_obj_pose'*	[CalibObjIdx, CalibObjPoseIdx]	*'pose_labels'*

Inhalt des Kalibrierdatenmodells während der Hand-Auge-Kalibrierung

create_calib_data:

Siehe Abschnitt **'Inhalt des Kalibrierdatenmodells während der Kamerakalibrierung'**.

set_calib_data:

Zum Modell zugefügte Daten	ItemType	ItemIdx	DataName
Optimierungsmethode für die Hand-Auge-Kalibrierung	*'model'*	*'general'*	*'optimization_method'*
Posen des Robotertools in Roboterbasiskoordinaten	*'tool'*	CalibObjPoseIdx	*'tool_in_base_pose'*

set_calib_data_observ_pose (Beobachtungen, erhalten durch 3D-Sensoren):

Zum Modell zugefügte Daten	abrufbar mit
Beobachtete Kalibrierkörperposen	get_calib_data_observ_pose

set_calib_data_cam_param, set_calib_data_calib_object und find_calib_object oder set_calib_data_observ_points (Beobachtungen, erhalten durch Kameras):

Siehe Abschnitt **'Inhalt des Kalibrierdatenmodells während der Kamerakalibrierung'**.

calibrate_hand_eye:

Szenario „bewegte Kamera“:

Zum Modell zugefügte Daten	ItemType	ItemIdx	DataName
Pose des Robotertools im Kamerakoordinatensystem	*'camera'*	0	*'tool_in_cam_pose'*
Pose des Kalibrierkörpers im Roboterbasiskoordinatensystem	*'calib_obj'*	0	*'obj_in_base_pose'*
Standardabweichung der Pose des Robotertools im Kamerakoordinatensystem	*'camera'*	0	*'tool_in_cam_pose_deviations'*
Kovarianzmatrizen der Pose des Robotertools im Kamerakoordinatensystem	*'camera'*	0	*'tool_in_cam_pose_covariances'*
Standardabweichung der Pose des Kalibrierkörpers im Roboterbasiskoordinatensystem	*'calib_obj'*	0	*'obj_in_base_pose_deviations'*
Kovarianzmatrizen der Pose des Kalibrierkörpers im Roboterbasiskoordinatensystem	*'calib_obj'*	0	*'obj_in_base_pose_covariances'*

Szenario „stationäre Kamera“:

Zum Modell zugefügte Daten	ItemType	ItemIdx	DataName
Pose der Roboterbasis im Kamerakoordinatensystem	*'camera'*	0	*'base_in_cam_pose'*
Pose des Kalibrierkörpers im Robotertoolkoordinatensystem	*'calib_obj'*	0	*'obj_in_tool_pose'*
Standardabweichung der Pose der Roboterbasis im Kamerakoordinatensystem	*'camera'*	0	*'base_in_cam_pose_deviations'*
Kovarianzmatrizen der Pose der Roboterbasis im Kamerakoordinatensystem	*'camera'*	0	*'base_in_cam_pose_covariances'*
Standardabweichung der Pose des Kalibrierkörpers im Robotertoolkoordinatensystem	*'calib_obj'*	0	*'obj_in_tool_pose_deviations'*
Kovarianzmatrizen der Pose des Kalibrierkörpers im Robotertoolkoordinatensystem	*'calib_obj'*	0	*'obj_in_tool_covariances'*

Beide Hand-Auge Szenarios:

Zum Modell zugefügte Daten	ItemType	ItemIdx	DataName
Kalibrierte Posen des Kalibrierkörpers im Kamerakoordinatensystem (nicht verfügbar für SCARA-Roboter)	*'calib_obj_pose'*	`[0, CalibObjPoseIdx]`	*'pose'*
Wurzel des mittleren quadratischen Rückprojektionsfehlers (RMSE) nach der Optimierung des Kamerasystems	*'model'*	*'general'*	*'camera_calib_error'*
Lagefehler über die komplette Transformationskette	*'model'*	*'general'*	*'hand_eye_calib_error'*
Kameraaufbaumodell (benötigt für eine Mehrbild-Stereorekonstruktion)	*'model'*	*'general'*	*'camera_setup_model'*

Die folgenden Abschnitte beschreiben die Parameter für die spezifischen Datenkategorien im Detail.

Modelldaten

ItemType='model': ItemIdx muss auf den Wert *'general'* gesetzt werden.

Die folgenden Modelldaten werden im Parameter DataValue je nach der Auswahl im Parameter DataName zurückgegeben:

'type': Generischer Typ des Kalibrierdatenmodells. Derzeit werden die fünf Typen *'calibration_object'*, *'hand_eye_moving_cam'*, *'hand_eye_stationary_cam'*, *'hand_eye_scara_moving_cam'* und *'hand_eye_scara_stationary_cam'* unterstützt.

'reference_camera': Referenzkamera für das Kalibrierdatenmodell. Alle 3D-Posen, sowohl für die Kameras, als auch für die Kalibrierkörper, werden relativ zu diesem Kamerakoordinatensystem ausgegeben.

'num_cameras': Anzahl von Kameras im Kalibrierdatenmodell (siehe create_calib_data).

'num_calib_objs': Anzahl von Kalibriermarken im Kalibrierdatenmodell (siehe create_calib_data).

'common_motion_vector': Für Stereosysteme, die Zeilenkameras mit telezentrischen Objektiven verwenden: ein String mit einem Booleschen Wert (d.h. *'true'* oder *'false'*), der festlegt, ob die Zeilenkameras einen gemeinsamen Bewegungsvektor besitzen.

'camera_setup_model': Generiert ein neues Kameraaufbaumodell und kapselt eine Kopie der kalibrierten Kameraparameter und -posen darin.

'camera_calib_error': Die Wurzel des mittleren quadratischen Rückprojektionsfehlers (RMSE) der Optimierung des Kamerasystems. Üblicherweise fragt man diesen Fehlerwert nach einer Hand-Auge-Kalibrierung (calibrate_hand_eye) ab, da bei einer Hand-Auge-Kalibrierung intern eine Kamerakalibrierung durchgeführt wird und der Fehler dieser Kamerakalibrierung nicht zurückgegeben wird. Der zurückgegebene Fehlerwert ist identisch mit dem von calibrate_cameras zurückgegebenen Wert.

'hand_eye_calib_error': Nach einer erfolgreichen Hand-Auge-Kalibrierung wird der Lagefehler über die komplette Transformationskette zurückgegeben. Genau genommen wird ein Tupel mit vier Elementen zurückgegeben, welches zuerst den mittleren, quadratischen Fehler der Translation, dann den mittleren, quadratischen Fehler der Rotation, an dritter Stelle den maximalen Fehler der Translation und schließlich an vierter Stelle den maximalen Fehler der Rotation angibt. Die zurückgegebenen Fehlerwerte sind identisch mit den von calibrate_hand_eye zurückgegebenen Werten.

'optimization_method': Optimierungsmethode, die für die Hand-Auge-Kalibrierung gesetzt wurde (siehe set_calib_data).

Die Parameter *'reference_camera'*, *'common_motion_vector'* und *'optimization_method'* können mit set_calib_data gesetzt werden. Die anderen Parameter werden während der Modellgenerierung gesetzt oder sind Ergebnisse des Kalibrierungsprozesses und können nicht modifiziert werden.

Kameradaten

ItemType='camera': ItemIdx legt fest, ob generische Kameradaten oder Daten einzelner Kameras abgefragt werden. Mit ItemIdx='general' wird der Standardwert eines Parameter für alle Kameras zurückgegeben. Wenn im Gegensatz dazu ein gültiger Kameraindex, d.h. eine Zahl zwischen 0 und NumCameras-1 (NumCameras wurde bei der Erstellung des Modells mit create_calib_data gesetzt), eingegeben wird, wird nur der Parameterwert für die ausgewählte Kamera zurückgegeben.

Beim Auswählen eines der folgenden Parameter in DataName kann abgefragt werden, welche Kameraparameter für die Optimierung im Zuge der Kalibrierung mit calibrate_cameras markiert, bzw. optimiert worden sind.

'calib_settings': Zu optimierende Kameraparameter.

'excluded_settings': Von der Optimierung ausgeschlossene Kameraparameter.

Diese Einstellungen können mit einem entsprechenden Aufruf von set_calib_data geändert werden.

Die folgenden Parameter können nur für einzelne Kameras abgefragt werden, d.h. ein gültiger Kameraindex muss in ItemIdx übergeben werden:

'type': Der Kameratyp, der mit set_calib_data_cam_param gesetzt wurde.

'init_params': Initiale interne Kameraparameter (siehe set_calib_data_cam_param).

'params': Optimierte interne Kameraparameter.

'params_deviations': Standardabweichungen der optimierten Kameraparameter, berechnet am Ende der Kamerakalibrierung. Wenn das Tupel, das für *'params'* zurückgegeben wird, n Elemente enthält, enthält das Tupel, das für *'params_deviations'* zurückgegeben wird, $(n-1)$ Elemente, da der Kameratyp im ersten Element des Kameraparametertupels gespeichert wird, während dies bei*'params_deviations'* nicht der Fall ist.

'params_covariances': Kovarianzmatrix der optimierten Kameraparameter, berechnet am Schluss der Kamerakalibrierung. Wenn das Tupel, das für *'params'* zurückgegeben wird, n Elemente enthält, enthält das Tupel, das für *'params_covariances'* zurückgegeben wird, $(n-1) \times (n-1)$ Elemente, da der Kameratyp im ersten Element des Kameraparametertupels gespeichert wird, während dies bei *'params_covariances'* nicht der Fall ist.

'params_labels': Kameratyp-spezifische Liste mit den Namen einzelner Kameraparameter, die von *'params'* zurückgegeben wird. Diese Liste enthält den Wert *'camera_type'* als erstes Element. Wenn das erste Element der Liste entfernt wird, bezieht sich die Liste von Namen auf *'params_deviations'* und die Zeilen und Spalten von *'params_covariances'*.

'init_pose': Initiale externe Kameraparameter, berechnet am Anfang der Kamerakalibrierung (siehe auch Kalibrierung). Die zurückgegebene 3D-Pose ist relativ zur Referenzkamera.

'pose': Optimierte Kamerapose, relativ zur Referenzkamera. Bei Kalibrierung einer einzelnen telezentrischen Kamera wird die Translation in z-Richtung auf den Wert 0.0 gesetzt. Bei mehr als einer telezentrischen Kamera, werden die Kameraposen so lange entlang ihrer z-Achse verschoben, bis sie alle auf einer Kugel mit Zentrum in der ersten beobachteten Kalibrierplatte liegen. Der Radius der Kugel entspricht dabei dem größten Abstand einer Kamera zur ersten beobachteten Kalibrierplatte. Sollte dieser berechnete Abstand kleiner als 1 m sein, so wird der Radius auf 1 m gesetzt.

'pose_labels': Liste mit den Namen einzelner Poseparameter.

Die kalibrierten Kameraparameter (*'params'* und *'pose'*) können nur nach einem erfolgreichen Durchlauf von `calibrate_cameras` abgerufen werden. Die initialen Kameraparameter *'init_params'* können nach einem erfolgreichen Durchlauf von `set_calib_data_cam_param` abgerufen werden.

Kalibrierkörperdaten

`ItemType`=*'calib_obj'*: `ItemIdx` muss auf den Index des Kalibrierkörpers, für den die Daten abgerufen werden, gesetzt werden, d.h., auf eine Zahl zwischen 0 und `NumCalibObjects-1` (`NumCalibObjects` wurde bei der Erstellung des Modells mit `create_calib_data` gesetzt).

Die folgenden Kalibrierkörperdaten werden im Parameter `DataValue` je nach der Auswahl im Parameter `DataName` zurückgegeben:

'num_marks': Anzahl der Kalibrierkörpermarken.

'x', *'y'*, *'z'*: Koordinaten der Kalibrierkörpermarken.

Diese Parameter werden mit dem Operator `set_calib_data_calib_object` gesetzt.

Daten zu den Kalibrierkörperposen

`ItemType`=*'calib_obj_pose'*: `ItemIdx` legt fest, ob generische Einstellungen aller Kalibrierkörperposen oder Einstellungen einzelner Kalibrierkörperposen abgefragt werden. Mit `ItemIdx`=*'general'* wird der Standardwert eines Parameters für alle Posen des Kalibrierkörpers zurückgegeben. Wenn im Gegensatz dazu ein gültiger Kalibrierkörperposeindex, d.h. ein Tupel mit gültigem Indexpaar `[CalibObjIdx, CalibObjPoseIdx]`, eingegeben wird, wird nur der Parameterwert für die ausgewählte Kalibrierkörperpose zurückgegeben.

Beim Auswählen eines der folgenden Parameter in `DataName` kann abgefragt werden, welche Kalibrierkörperposeparameter für die Optimierung markiert, bzw. optimiert worden sind.

'calib_settings': Zu optimierende Poseparameter.

'excluded_settings': Von der Optimierung ausgeschlossene Poseparameter.

Diese Einstellungen können mit einem entsprechenden Aufruf von `set_calib_data` geändert werden.

Die folgenden Parameter können nur für einzelne Posen eines Kalibrierkörpers abgefragt werden, d.h. ein gültiges Poseindexpaar `[CalibObjIdx, CalibObjPoseIdx]` muss in `ItemIdx` eingegeben werden:

'init_pose': Initiale 3D-Pose des Kalibrierkörpers. Diese Pose wird anhand der Posen in den Beobachtungsdaten von diesem Körper am Anfang der Kalibrierung berechnet (siehe Kalibrierung). Die zurückgegebene 3D-Pose ist relativ zur Referenzkamera.

'pose': Optimierte 3D-Pose des Kalibrierkörpers relativ zur Referenzkamera.

'pose_labels': Liste mit den Namen einzelner Poseparameter.

Diese Parameter können nicht geändert werden und sind erst nach der Kalibrierung mit `calibrate_cameras` abfragbar.

Daten zur Hand-Auge-Kalibrierung

Nach einer erfolgreichen Hand-Auge-Kalibrierung mittels `calibrate_hand_eye`, können je nach Modelltyp die folgenden Posen abgerufen werden:

'hand_eye_moving_cam', *'hand_eye_scara_moving_cam'*: Ist `ItemType`=*'camera'* und `DataName`=*'tool_in_cam_pose'* gesetzt, so wird die Pose des Robotertools in Kamerakoordinaten in `DataValue` zurückgegeben. Ist `ItemType`=*'calib_obj'* und `DataName`=*'obj_in_base_pose'* gesetzt, so wird die Pose des Kalibrierkörpers in Roboterbasiskoordinaten in `DataValue` zurückgegeben.

Es ist zu beachten, dass die Z-Translation von *'obj_in_base_pose'* im Rahmen der Kalibrierung von SCARA-Robotern nicht eindeutig bestimmt werden kann. Um diese Mehrdeutigkeit aufzulösen, wird die Z-Translation von *'obj_in_base_pose'* intern auf 0.0 gesetzt und *'tool_in_cam_pose'* entsprechend berechnet. Es ist notwendig, die tatsächliche Translation in Z nach der Kalibrierung zu bestimmen (siehe `calibrate_hand_eye`).

Die Standardabweichungen und Kovarianzmatrizen der 6 Poseparameter beider Posen können mit *'tool_in_cam_pose_deviations'*, *'tool_in_cam_pose_covariances'* (`ItemType`=*'camera'*), *'obj_in_base_pose_deviations'* und *'obj_in_base_pose_covariances'* (`ItemType`=*'calib_obj'*) abgefragt werden. Wie die Poseparameter selbst werden auch diese in den Einheiten [m] und [°] angegeben.

'hand_eye_stationary_cam', *'hand_eye_scara_stationary_cam'*: Ist `ItemType`=*'camera'* und `DataName`=*'base_in_cam_pose'* gesetzt, so wird die Pose der Roboterbasis in Kamerakoordinaten in `DataValue` zurückgegeben. Ist `ItemType`=*'calib_obj'* und `DataName`=*'obj_in_tool_pose'* gesetzt, so wird die Pose des Kalibrierkörpers in Toolkoordinaten in `DataValue` zurückgegeben.

Es ist zu beachten, dass die Z-Translation von *'obj_in_tool_pose'* im Rahmen der Kalibrierung von SCARA-Robotern nicht eindeutig bestimmt werden kann. Um diese Mehrdeutigkeit aufzulösen, wird die Z-Translation von *'obj_in_tool_pose'* intern auf 0.0 gesetzt und *'base_in_cam_pose'* entsprechend berechnet. Es ist notwendig, die tatsächliche Translation in Z nach der Kalibrierung zu bestimmen (siehe `calibrate_hand_eye`).

Die Standardabweichungen und Kovarianzmatrizen der 6 Poseparameter beider Posen können mit *'base_in_cam_pose_deviations'*, *'base_in_cam_pose_covariances'* (`ItemType`=*'camera'*), *'obj_in_tool_pose_deviations'* und *'obj_in_tool_pose_covariances'* (`ItemType`=*'calib_obj'*) abgefragt werden. Wie die Poseparameter selbst werden auch diese in den Einheiten [m] und [°] angegeben.

Setzt man `ItemType`=*'tool'* und `DataName`=*'tool_in_base_pose'*, so wird die Pose des Robotertools mit Index `ItemIdx` in Roboterbasiskoordinaten zurückgegeben. Diese Posen wurden vorher mit `set_calib_data` gesetzt und dienten als Eingabedaten der Hand-Auge-Kalibrierung.

Parameter

▷ **`CalibDataID`** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **`ItemType`** (input_control) . . . . . string ⇝ *string*
Typ des abgefragten Elements.
Defaultwert: 'camera'
Werteliste: `ItemType` ∈ {'model', 'camera', 'calib_obj', 'calib_obj_pose', 'tool'}

▷ **`ItemIdx`** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Index des abgefragten Elements (je nach Auswahl in `ItemType`).
Defaultwert: 0
Wertevorschläge: `ItemIdx` ∈ {0, 1, 2, 'general'}

▷ **`DataName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name der abgefragten Daten.
Defaultwert: 'params'
Werteliste: `DataName` ∈ {'type', 'reference_camera', 'num_cameras', 'num_calib_objs', 'camera_setup_model', 'camera_calib_error', 'hand_eye_calib_error', 'optimization_method', 'num_marks', 'x', 'y', 'z', 'params', 'pose', 'init_params', 'init_pose', 'params_deviations', 'params_covariances', 'params_labels', 'pose_labels', 'calib_settings', 'excluded_settings', 'common_motion_vector', 'tool_in_cam_pose', 'obj_in_base_pose', 'base_in_cam_pose', 'obj_in_tool_pose', 'tool_in_base_pose', 'tool_in_cam_pose_deviations', 'obj_in_base_pose_deviations', 'base_in_cam_pose_deviations', 'obj_in_tool_pose_deviations', 'tool_in_cam_pose_covariances', 'obj_in_base_pose_covariances', 'base_in_cam_pose_covariances', 'obj_in_tool_pose_covariances'}

▷ **`DataValue`** (output_control) . . . . . attribute.value(-array) ⇝ *real* / integer / string
Werte der abgefragten Daten.

Beispiel

```
* Get the camera type of camera 0.
get_calib_data (CalibDataID, 'camera', 0, 'type', CameraType)

* Get the optimized (calibrated) pose of pose 1 of the
* calibration object 2.
get_calib_data (CalibDataID, 'calib_obj_pose', [2,1], 'pose', CalobjPose)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`calibrate_cameras`, `calibrate_hand_eye`, `create_calib_data`, `read_calib_data`

Modul

Calibration

get_calib_data_observ_contours (: Contours : CalibDataID, ContourName, CameraIdx, CalibObjIdx, CalibObjPoseIdx :)

Abfragen konturbasierter Beobachtungsdaten aus dem Kalibrierdatenmodell.

Der Operator `get_calib_data_observ_contours` liest konturbasierte Beobachtungsdaten aus dem Kalibrierdatenmodell `CalibDataID` aus und gibt diese in `Contours` zurück. Diese Konturen stammen aus einem vorherigen Aufruf des Operators `find_calib_object`. Die Parameter `CameraIdx`, `CalibObjIdx` und `CalibObjPoseIdx` sind Indizes der im Modell abgespeicherten beobachtenden Kamera, Kalibrierplatte und Pose. Zusammen bezeichnen sie eine Beobachtung des Kalibrierdatenmodells. Es ist zu beachten, dass Beobachtungen, welche von `set_calib_data_observ_points` im Kalibrierdatenmodell `CalibDataID` gespeichert worden sind, keine konturbasierten Daten enthalten.

Je nach Auswahl im Parameter `ContourName` werden die folgenden Konturen zurückgeliefert:

'marks': Die Konturen der Marken der Kalibrierplatte

'caltab': Die Kontur des Suchmusters der Kalibrierplatte.

'last_caltab': Die Kontur des Suchmusters jener Kalibrierplatte, die beim letzten erfolgreichen Aufruf von `find_calib_object` extrahiert wurde. Es ist zu beachten, dass dabei die Beobachtung des erfolgreichen Aufrufs von `find_calib_object` verwendet wird und somit die Werte in `CameraIdx`, `CalibObjIdx` und `CalibObjPoseIdx` ignoriert werden.

Das erwähnte Suchmuster hängt von der Kalibrierplatte ab:

- Kalibrierplatten mit hexagonaler Markenanordnung: Hexagon von Marken (also einer Marke und deren sechs Nachbarmarken), das vier oder sechs Marken mit einem Loch enthält, siehe `create_caltab`.
- Kalibrierplatten mit rechteckiger Markenanordnung: Rahmen der Kalibrierplatte mit Dreieck in einer Ecke.

Parameter

▷ **Contours** (output_object) ..... xld_cont(-array) ⇝ *object*
Konturbasierte Daten.

▷ **CalibDataID** (input_control) ..... calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **ContourName** (input_control) ..... string ⇝ *string*
Name der Konturobjekte.
Defaultwert: 'marks'
Werteliste: `ContourName` ∈ {'marks', 'caltab', 'last_caltab'}

▷ **CameraIdx** (input_control) . . . . . number ⇝ *integer*
Index der beobachtenden Kamera.
Defaultwert: 0

▷ **CalibObjIdx** (input_control) . . . . . number ⇝ *integer*
Index der beobachteten Kalibrierplatte.
Defaultwert: 0

▷ **CalibObjPoseIdx** (input_control) . . . . . number ⇝ *integer*
Index der beobachteten Pose.
Defaultwert: 0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Calibration

get_calib_data_observ_points (: : CalibDataID, CameraIdx, CalibObjIdx, CalibObjPoseIdx : Row, Column, Index, Pose)

Abfragen punktbasierter Beobachtungsdaten aus dem Kalibrierdatenmodell.

Mit dem Operator get_calib_data_observ_points werden punktbasierte Beobachtungsdaten aus dem Kalibrierdatenmodell CalibDataID abgefragt. Der Operator gibt die Daten zurück, die mit den Operatoren set_calib_data_observ_points oder find_calib_object gespeichert wurden. Siehe set_calib_data_observ_points für weitere Information über die einzelnen Parameter.

Für den Fall, dass set_calib_data_observ_points für die Kalibrierung verwendet wird, sind die zurückgelieferten Werte von Row und Column die originalen Werte, die mit set_calib_data_observ_points festgelegt wurden. Falls find_calib_object für die Extraktion verwendet wird, stimmen die Werte von Row und Column, die von get_calib_data_observ_points zurückgegeben werden, mit den Koordinaten der gefundenen Punkte, die von find_calib_object ermittelt werden, überein.

Beachten Sie, dass get_calib_data_observ_points die Pose von unkalibrierten Modellen zurück gibt. Um an die Pose eines kalibrierten Modells zu gelangen, nutzen Sie get_calib_data.

Parameter

▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **CameraIdx** (input_control) . . . . . number ⇝ *integer*
Index der beobachtenden Kamera.
Defaultwert: 0

▷ **CalibObjIdx** (input_control) . . . . . number ⇝ *integer*
Index des beobachteten Kalibrierkörpers.
Defaultwert: 0

▷ **CalibObjPoseIdx** (input_control) . . . . . number ⇝ *integer*
Index der beobachteten Pose.
Defaultwert: 0

▷ **Row** (output_control) . . . . . number-array ⇝ *real* / integer
Zeilenkoordinaten der extrahierten Punktmerkmale.

▷ **Column** (output_control) . . . . . number-array ⇝ *real* / integer
Spaltenkoordinaten der extrahierten Punktmerkmale.

▷ **Index** (output_control) . . . . . number-array ⇝ *integer* / real
Zuordnung der extrahierten Punktmerkmale zu den Kalibrierkörpermarken.

▷ **Pose** (output_control) . . . . . number-array ⇝ *real* / integer
Grob geschätzte 3D-Pose der Kalibrierkörper relativ zur beobachtenden Kamera.
Parameteranzahl: 7

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Calibration

get_camera_setup_param (: : CameraSetupModelID, CameraIdx, GenParamName : GenParamValue)

Abfragen generischer Kameraaufbaumodellparameter.

Mit dem Operator get_camera_setup_param können generische Parameter des Kameraaufbaumodells CameraSetupModelID abgefragt werden. Dabei werden zwei Typen von Parameter unterschieden:

Generische Modellparameter:

Mit CameraIdx='*general*' werden, je nach Auswahl im Parameter GenParamName, die folgenden Kameraaufbauparameter im Parameter GenParamValue zurückgegeben:

'num_cameras': Im Modell gesetzte Anzahl der Kameras. Die Anzahl der Kameras im Modell wird bei dessen Erstellung festgelegt und kann im Nachhinein nicht mehr geändert werden (siehe create_camera_setup_model).

'camera_calib_error': Die Wurzel des mittleren quadratischen Rückprojektionsfehlers (RMSE) der Optimierung des Kamerasystems. Dieser Fehlerwert ist identisch mit dem von calibrate_cameras zurückgegebenen Fehlerwert.

'reference_camera': Gibt den Index der Kamera zurück, die als Referenzkamera des Systems festgelegt wurde. Wurde mit set_camera_setup_param) keine Kamera spezifiziert, so wird als Index die 0-te Kamera zurückgegeben. Für den Fall, dass das Koordinatensystem durch Setzen einer Pose (siehe *'coord_transf_pose'* in set_camera_setup_param) verschoben wurde und somit der Ursprung in keiner der Kameras liegt, so wird als Index -1 zurückgegeben.

'coord_transf_pose': Gibt die Pose zurück in die das Koordinatensystem des Aufbaus verschoben wurde. Dabei ist zu beachten, dass beim Setzen einer Referenzkamera (siehe set_camera_setup_param) die Pose zurückgegeben wird, die der Pose der gesetzten Kamera entspricht. Eine nachträgliche Verschiebung dieses Koordinatensystem mit Hilfe von *'coord_transf_pose'* in set_camera_setup_param liefert eine Pose zurück, die die Position und Orientierung des gewünschten Koordinatensystems relativ zum aktuellen darstellt.

Kameraparameter:

Wenn im Parameter CameraIdx einen gültigen Kameraindex (d.h. eine Zahl zwischen 0 und NumCameras-1) eingegeben wird, können die folgenden Parameter, je nach Auswahl im Parameter GenParamName, für die ausgewählte Kamera abgefragt werden, ihr Wert wird im Parameter GenParamValue zurückgegeben:

'type': Kameratyp (siehe set_camera_setup_cam_param).

'params': Ein Tupel mit den internen Kameraparameter. Die Länge dieses Tupels hängt vom Kameratyp ab.

'params_deviations': Ein Tupel mit den Standardabweichungen der internen Kameraparameter. Die Länge dieses Tupels hängt von dem Kameratyp ab.

'params_covariances': Ein Tupel mit der Kovarianzmatrix der internen Kameraparameter. Das Tupel stellt eine quadratische symmetrische Matrix dar, das dieselbe Größe hat wie das Tupel mit den entsprechenden Standardabweichungen.

'pose': Kamerapose relativ zum Koordinatensystem des Kameraaufbaus (siehe create_camera_setup_model für weitere Information).

Es ist zu beachten, dass eine Kamera im Modell bereits definiert sein muss, um ihre Parameter mit `get_camera_setup_param` abfragen zu können. Wenn `CameraIdx` der Index einer undefinierten Kamera ist, gibt der Operator einen Fehler zurück.

Weitere detaillierte Informationen zur Kalibrierung von Kameraaufbauten befinden sich im Kapitel Kalibrierung.

Parameter

- ▷ **`CameraSetupModelID`** (input_control) . . . . . camera_setup_model ⇝ *handle*
  Handle des Kameraaufbaumodells.
- ▷ **`CameraIdx`** (input_control) . . . . . integer(-array) ⇝ *integer* / string
  Index der Kamera im Modell.
  Defaultwert: 0
  Wertevorschläge: `CameraIdx` ∈ {0, 1, 2, 'general'}
- ▷ **`GenParamName`** (input_control) . . . . . attribute.name ⇝ *string*
  Namen der generischen Parameter.
  Werteliste: `GenParamName` ∈ {'camera_calib_error', 'type', 'params', 'params_deviations', 'params_covariances', 'pose', 'reference_camera', 'coord_transf_pose', 'num_cameras'}
- ▷ **`GenParamValue`** (output_control) . . . . . attribute.value(-array) ⇝ *real* / integer / string
  Werte der generischen Parameter.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Calibration

query_calib_data_observ_indices (: : CalibDataID, ItemType, ItemIdx : Index1, Index2)

Abfragen von Information zur Beziehung zwischen Kameras, Kalibrierkörpern und Kalibrierkörperposen.

Ein Kalibrierdatenmodell (`CalibDataID`) speichert einen Satz von Beobachtungsdaten, die mit `set_calib_data_observ_points` zum Modell hinzugefügt werden. Jede Beobachtung wird einer Kamera und einer Kalibrierkörperpose zugeordnet. Der Operator `query_calib_data_observ_indices` liefert die Indizes von Beobachtungsdaten zurück, welche einer bestimmten Kamera bzw. einer bestimmten Kalibrierkörperpose zugeordnet sind.

Für `ItemType`=*'camera'* muss ein gültiger Kameraindex in `ItemIdx` eingegeben werden. Das ist eine Zahl, die zwischen 0 und `NumCameras-1` liegt, wobei `NumCameras` mit dem Operator `create_calib_data` bereits bei der Erstellung des Modells gesetzt wurde. Dann liefern die Ausgabeparameter `Index1` und `Index2` eine Liste von Kalibrierkörperindizes bzw. eine Liste von Posenindizes zurück. Die Einträge beider Listen bilden Schlüsselpaare `[Index1[I],Index2[I]]` der 'beobachteten' Kalibrierkörperposen mit `I=0..|Index1|-1` und `|Index1| = |Index2|`.

Für `ItemType`=*'calib_obj_pose'* muss ein gültiger Kalibrierkörperindex in `ItemIdx` eingegeben werden. Das ist eine Zahl, die zwischen 0 und `NumCalibObjects-1` liegt, wobei `NumCalibObjects` mit dem Operator `create_calib_data` bereits bei der Erstellung des Modells gesetzt wurde. Dann liefern die Ausgabeparameter `Index1` und `Index2` eine Liste von Kameraindizes bzw. eine Liste von Kalibrierkörperposen-Indizes zurück. Jedes Paar `[Index1[I],Index2[I]]` mit `I=0..|Index1|-1` und `|Index1| = |Index2|`, bedeutet, dass die Kamera `Index1[I]` die Kalibrierkörperpose `[ItemIdx, Index2[I]]` 'beobachtet'.

Dieser Operator ist insbesondere dann nützlich, wenn die Beobachtungsdaten eines Kalibrierdatenmodells untersucht werden sollen. Das ist typischerweise der Fall bei Kalibrierdatenmodellen, die aus einer Datei gelesen werden und deren Konfiguration und Inhalt noch unbekannt sind. Darüber hinaus wird der Operator zum Abfragen einer Liste aller Posen, welche einem Kalibrierkörper zugeordnet sind, benutzt (siehe Beispiel).

Parameter

▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **ItemType** (input_control) . . . . . string ⇝ *string*
Typ des abgefragten Elements.
Defaultwert: 'camera'
Werteliste: `ItemType` ∈ {'camera', 'calib_obj'}

▷ **ItemIdx** (input_control) . . . . . number ⇝ *integer*
Index einer Kamera ID oder eines Kalibrierkörpers (je nach Auswahl in `ItemType`).
Defaultwert: 0
Wertevorschläge: `ItemIdx` ∈ {0, 1, 2}

▷ **Index1** (output_control) . . . . . number-array ⇝ *integer*
Liste von Kalibriermarkennummer oder Liste von Kameranummer (je nach Auswahl in `ItemType`).

▷ **Index2** (output_control) . . . . . number-array ⇝ *integer*
Kalibrierkörpernummer.

Beispiel

```
* Read a calibration model from a file.
read_calib_data ('calib_data.ccd',CalibDataID)

* Get calibration object indices assigned to calibration object 0.
query_calib_data_observ_indices (CalibDataID, 'calib_obj', 0, _, \
                                 CalibObjPoseIndices)
* CalibObjPoseIndices contains the list of pose indices of calibration
* object 0. In order to be stored in the model, each calibration object
* needs to be observed by at least one camera in the setup (a calibration
* object pose that is not observed by any camera cannot be stored in
* the model). Typically, a calibration object pose can be observed by more
* than one camera. Hence, some calibration object pose indices might appear
* repeatedly in CalibObjPoseIndices. We use tuple_sort and tuple_uniq to
* extract a unique list of calibration object pose indices for calibration
* object 0.
tuple_sort (CalibObjPoseIndices, CalibObjPoseIndices)
tuple_uniq (CalibObjPoseIndices, CalibObjPoseIndices)

* Get poses of calibration objects observed by camera 2.
calibrate_cameras (CalibDataID, Error)
query_calib_data_observ_indices (CalibDataID, 'camera', 2, CalibObjIndices,\
                                 CalibObjPoseIndices)
for I := 0 to |CalibObjIndices|-1 by 1
  get_calib_data (CalibDataID, 'calib_obj_pose', \
                  [CalibObjIndices[I], CalibObjPoseIndices[I]], \
                  'pose', CalibObjPose)
endfor
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Modul

Calibration

read_calib_data (: : FileName : CalibDataID)

Einlesen eines Kalibrierdatenmodells aus einer Datei.

Der Operator `read_calib_data` liest ein Kalibrierdatenmodell aus der Datei `FileName` und gibt das Handle des nun im Speicher angelegten Modells im `CalibDataID` zurück. Die Modelldatei wird mit dem Operator `write_calib_data` erzeugt.

Beachten Sie, dass `write_calib_data` keine Kalibrierergebnisse in die Datei speichert. Für ein vollständig konfiguriertes Modell kann allerdings `calibrate_cameras` direkt nach dem Lesen aufgerufen werden und die Kalibrierergebnisse sind danach wieder abfragbar.

Parameter

- ▷ **FileName** (input_control) ..... filename.read ⇝ *string*
  Dateiname des Modells.
  Dateiendung: `.ccd`
- ▷ **CalibDataID** (output_control) ..... calib_data ⇝ *handle*
  Handle des Kalibriermodells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Modul

Calibration

read_camera_setup_model (: : FileName : CameraSetupModelID)

Einlesen eines Kameraaufbaumodells aus einer Datei.

Der Operator `read_camera_setup_model` liest ein Kameraaufbaumodell aus der Datei `FileName` und gibt das Handle des nun im Speicher angelegten Modells im `CameraSetupModelID` zurück. Die Modelldatei wird mit dem Operator `write_camera_setup_model` erzeugt.

Parameter

- ▷ **FileName** (input_control) ..... filename.read ⇝ *string*
  Dateiname des Modells.
  Dateiendung: `.csm`
- ▷ **CameraSetupModelID** (output_control) ..... camera_setup_model ⇝ *handle*
  Handle des Kameraaufbaumodells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Modul

Calibration

remove_calib_data (: : CalibDataID, ItemType, ItemIdx :)

Entfernen eines Datensatzes aus dem Kalibrierdatenmodell.

Mittels des Operators remove_calib_data können Daten aus dem Kalibrierdatenmodell CalibDataID entfernt werden. Momentan kann nur der Datensatz für die Hand-Auge-Kalibrierung geändert werden. Mit ItemType='tool' kann die Robotertoolpose entfernt werden, die benutzt wurde um die Beobachtung der Pose des Kalibrierobjektes mit dem gleichen Index ItemIdx zu erlangen. (ItemIdx entspricht dem Parameter CalibObjPoseIdx aus find_calib_object, set_calib_data_observ_pose, oder set_calib_data_observ_pose). Zu beachten ist, dass die entsprechende Beobachtung des Kalibrierobjektes mit demselben Index ItemIdx auch gelöscht werden muss. Sonst liefert der Operator calibrate_hand_eye eine Fehlermeldung.

Parameter

- ▷ **CalibDataID** (input_control) ..... calib_data ⇝ *handle*
  Handle des Kalibrierdatenmodells.
- ▷ **ItemType** (input_control) ..... string ⇝ *string*
  Typ des abgefragten Elements.
  Defaultwert: 'tool'
  Werteliste: ItemType ∈ {'tool'}
- ▷ **ItemIdx** (input_control) ..... number(-array) ⇝ *integer* / string
  Index des abgefragten Elements.
  Defaultwert: 0
  Wertevorschläge: ItemIdx ∈ {0, 1, 2}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

set_calib_data, remove_calib_data_observ

Nachfolger

calibrate_hand_eye

Siehe auch

calibrate_cameras

Modul

Calibration

remove_calib_data_observ (: : CalibDataID, CameraIdx, CalibObjIdx, CalibObjPoseIdx :)

Entfernen einer Beobachtung aus dem Kalibrierdatenmodell

Der Operator remove_calib_data_observ entfernt Beobachtungen aus dem Kalibrierdatenmodell CalibDataID, die vorher mittels find_calib_object, set_calib_data_observ_points oder set_calib_data_observ_pose gesetzt wurden. Die Parameter CameraIdx, CalibObjIdx und CalibObjPoseIdx müssen eine gültige Beobachtung bezeichnen. Falls das Kalibrierdatenmodell CalibDataID zusammen mit calibrate_hand_eye benutzt wird, muss auch die entsprechende Pose des Robotertools mit remove_calib_data gelöscht werden.

Parameter

- ▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
  Handle des Kalibrierdatenmodells.
- ▷ **CameraIdx** (input_control) . . . . . number ⇝ *integer*
  Index der beobachtenden Kamera.
  Defaultwert: 0
- ▷ **CalibObjIdx** (input_control) . . . . . number ⇝ *integer*
  Index des beobachteten Kalibrierkörpers.
  Defaultwert: 0
- ▷ **CalibObjPoseIdx** (input_control) . . . . . number ⇝ *integer*
  Index der beobachteten Pose.
  Defaultwert: 0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

find_calib_object, set_calib_data_observ_points, set_calib_data_observ_pose

Nachfolger

remove_calib_data, calibrate_cameras, calibrate_hand_eye

Modul

Calibration

serialize_calib_data (: : CalibDataID : SerializedItemHandle)

Serialisieren eines Kalibrierdatenmodells.

serialize_calib_data serialisiert die Daten eines Kalibrierdatenmodells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_calib_data in eine Datei geschrieben werden. Das Kalibrierdatenmodell wird in dem Handle CalibDataID übergeben. Das serialisierte Kalibrierdatenmodell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_calib_data wieder deserialisiert werden.

Beachten Sie, dass keine Ergebnisse der Kalibrierung serialisiert werden. Diese können stattdessen mit get_calib_data abgefragt werden und einzeln oder gemeinsam in Form eines Kameraaufbaumodells gespeichert werden.

Parameter

▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_calib_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_calib_data`

Modul

Calibration

serialize_camera_setup_model

```
serialize_camera_setup_model ( : : CameraSetupModelID : SerializedItemHandle )
```

Serialisieren eines Kameraaufbaumodells.

`serialize_camera_setup_model` serialisiert die Daten eines Kameraaufbaumodells (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_camera_setup_model` in eine Datei geschrieben werden. Das Kameraaufbaumodell wird in dem Handle `CameraSetupModelID` übergeben. Das serialisierte Kameraaufbaumodell wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_camera_setup_model` wieder deserialisiert werden.

Parameter

▷ **CameraSetupModelID** (input_control) . . . . . camera_setup_model ⇝ *handle*
Handle des Kameraaufbaumodells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_camera_setup_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_camera_setup_model`

Modul

Calibration

set_calib_data (: : CalibDataID, ItemType, ItemIdx, DataName, DataValue :)

Setzen von Parametern für das Kalibrierdatenmodell.

Mit dem Operator set_calib_data können Daten im Kalibrierdatenmodell CalibDataID gespeichert werden.

Eine Übersicht wie das Kalibrierdatenmodell im Laufe des Prozesses einer Kamerakalibrierung oder einer Hand-Auge-Kalibrierung mit Daten gefüllt wird, ist in der Beschreibung von get_calib_data gegeben.

Das Kalibrierdatenmodell kann unterschiedliche Arten von Daten enthalten. Wie bestimmte Daten im Kalibrierdatenmodell gesetzt werden, wird für verschiedene Kategorien von Daten beschrieben:

- Modelldaten (ItemType *'model'*)
- Kameradaten (ItemType *'camera'*)
- Daten zu den Kalibrierkörperposen (ItemType *'calib_obj_pose'*)
- Daten für Hand-Auge-Kalibrierung (ItemType *'tool'*)

Der Parameter ItemIdx bezeichnet, ob der neue Parameterwert für alle oder nur für ein Element gesetzt werden soll. Die Daten, die geändert werden sollen, werden in DataName eingegeben, ihre Werte in DataValue.

Weitere detaillierte Informationen zur Kalibrierung von Kameraaufbauten befinden sich im Kapitel Kalibrierung.

Modelldaten

ItemType=*'model'***:** ItemIdx muss auf den Wert *'general'* gesetzt werden.

Je nach Auswahl in DataName werden die folgenden Modellparameter auf den in DataValue übergebenen Wert gesetzt:

*'reference_camera'***:** Referenzkamera für das Kalibrierdatenmodell. Alle 3D-Posen, sowohl für die Kameras, als auch für die Kalibrierkörper, werden relativ zu diesem Kamerakoordinatensystem angegeben (siehe get_calib_data).

*'common_motion_vector'***:** Für Stereosysteme, die Zeilenkameras mit telezentrischen Objektiven verwenden: ein String mit einem Booleschen Wert (d.h. *'true'* oder *'false'*), der festlegt, ob die Zeilenkameras einen gemeinsamen Bewegungsvektor besitzen.

*'optimization_method'***:** Setzen der Optimierungsmethode für die Hand-Auge-Kalibrierung. Falls DataValue=*'linear'* gesetzt ist, wird eine lineare Methode verwendet. Falls DataValue=*'nonlinear'* gesetzt ist, wird eine nichtlineare Methode verwendet (siehe calibrate_hand_eye für mehr Details).

Kameradaten

ItemType=*'camera'***:** ItemIdx legt fest, ob die Kalibriereinstellungen für alle Kameras oder für einzelne Kameras gespeichert werden sollen. Mit ItemIdx=*'general'* werden die neuen Einstellungen als Standardwert für alle Kameras im Modell gesetzt. Wenn dagegen ein gültiger Kameraindex in ItemIdx eingegeben wird, d.h. eine Zahl zwischen 0 und NumCameras-1 (NumCameras wurde bei der Erstellung des Modells mit create_calib_data gesetzt), dann werden nur die Einstellungen der ausgewählten Kamera geändert.

Beim Auswählen eines der folgenden Parameter in DataName kann eingegeben werden, **welche Kameraparameter optimiert werden** sollen:

*'calib_settings'***:** Zusätzlich zu den bereits für die Optimierung ausgewählten Kameraparametern zu optimierende Kameraparameter. Standardmäßig sind alle Kameraparameter für die Optimierung ausgewählt. D.h., *'calib_settings'* ist hauptsächlich dazu geeignet, vorher ausgeschlossene Parameter wieder zur Auswahl hinzuzufügen.

*'excluded_settings'***:** Von der Optimierung ausgeschlossene Kameraparameter.

Die folgenden Kameraparameter können in DataValue verwendet werden:

Interne Kameraparameter

*'focus'***:** Brennweite des Kameraobjektivs (nur perspektivische Kameras, d.h. für Kameras vom Typ 'area_scan_division', 'area_scan_polynomial', 'area_scan_tilt_division', 'area_scan_tilt_polynomial', 'area_scan_tilt_image_side_telecentric_division', 'area_scan_tilt_image_side_telecentric_polynomial', 'area_scan_hypercentric_division', 'area_scan_hypercentric_polynomial', 'line_scan_division' und 'line_scan_polynomial').

'magnification': Vergrößerungsmaßstab des Kameraobjektivs (nur für telezentrische Kameras, d.h. Kameras vom Typ 'area_scan_telecentric_division', 'area_scan_telecentric_polynomial', 'area_scan_tilt_bilateral_telecentric_division', 'area_scan_tilt_bilateral_telecentric_polynomial', 'area_scan_tilt_object_side_telecentric_division' 'area_scan_tilt_object_side_telecentric_polynomial', 'line_scan_telecentric_division' und 'line_scan_telecentric_polynomial').

'kappa': Verzeichnungskoeffizient zur Modellierung der Verzeichnung mit dem Divisionsmodell (nur für Kameras vom Typ 'area_scan_division', 'area_scan_telecentric_division', 'area_scan_tilt_division', 'area_scan_tilt_bilateral_telecentric_division', 'area_scan_tilt_object_side_telecentric_division', 'area_scan_hypercentric_division', 'line_scan_division' und 'line_scan_telecentric_division').

'k1',*'k2'*, *'k3'*: Radiale Verzeichnungskoeffizienten zur Modellierung der Verzeichnung mit dem Polynommodell (nur für Kameras vom Typ 'area_scan_polynomial', 'area_scan_telecentric_polynomial', 'area_scan_tilt_polynomial', 'area_scan_telecentric_tilt_polynomial', 'area_scan_hypercentric_polynomial', 'line_scan_polynomial' und 'line_scan_telecentric_polynomial').

'poly_tan_2': Allgemeiner Parametername, der für alle tangentialen Verzeichnungskoeffizienten des Polynommodells steht, d.h. für p1 und p2 (nur für Kameras vom Typ 'area_scan_polynomial', 'area_scan_telecentric_polynomial', 'area_scan_tilt_polynomial', 'area_scan_tilt_bilateral_telecentric_polynomial', 'area_scan_tilt_object_side_telecentric_polynomial', 'area_scan_hypercentric_polynomial', 'line_scan_polynomial' und 'line_scan_telecentric_polynomial').

'poly': Allgemeiner Parametername, der für alle Verzeichnungskoeffizienten des Polynommodells steht, d.h. k1, k2, k3, p1 und p2 (nur für Kameras vom Typ 'area_scan_polynomial', 'area_scan_telecentric_polynomial', 'area_scan_tilt_polynomial', 'area_scan_tilt_bilateral_telecentric_polynomial', 'area_scan_tilt_object_side_telecentric_polynomial', 'area_scan_hypercentric_polynomial', 'line_scan_polynomial' und 'line_scan_telecentric_polynomial').

'image_plane_dist': Der Abstand der verkippten Bildebene vom Zentrum der perspektivischen Projektion (nur für Kameras mit Tiltobjektiven, die eine perspektivische Projektion auf der Bildseite des Objektivs durchführen, d.h. Kameras vom Typ 'area_scan_tilt_division', 'area_scan_tilt_polynomial', 'area_scan_tilt_object_side_telecentric_division' und 'area_scan_tilt_object_side_telecentric_polynomial').

'tilt': Verkippung und Rotation des Tiltobjektivs (nur für Kameras mit Tiltobjektiven, d.h. Kameras vom Typ 'area_scan_tilt_division', 'area_scan_tilt_polynomial', 'area_scan_tilt_bilateral_telecentric_division', 'area_scan_tilt_bilateral_telecentric_polynomial', 'area_scan_tilt_object_side_telecentric_division' und 'area_scan_tilt_object_side_telecentric_polynomial').

'cx',*'cy'*: Koordinaten des Kamerahauptpunktes.

'principal_point': Allgemeiner Parametername, der für *'cx'* und *'cy'* steht.

'sx',*'sy'*: Größe der Sensorelemente des Kamerasensors.

'params': Allgemeiner Parametername, der für alle internen Kameraparameter steht.

Bitte beachten Sie, dass die Bezeichnung *Brennweite* (Focus) nicht ganz richtig ist, da der Parameter Focus im Allgemeinen nicht der realen Brennweite entspricht, wenn die Objektweite nicht unendlich ist. Der Einfachheit halber wird hier aber nicht zwischen Brennweite und Bildweite unterschieden.

Externe Kameraparameter

'alpha',*'beta'*,*'gamma'*: Rotationswerte der Kamerapose.

'transx',*'transy'*,*'transz'*: Translationsvektor der Kamerapose.

'pose': Allgemeiner Parametername, der für alle Kameraposeparameter steht.

Weitere Kameraparameter

'vx',*'vy'*,*'vz'*: Kamerabewegungsvektor (nur für Kameras von Typ 'line_scan_division', 'line_scan_polynomial', 'line_scan_telecentric_division' und 'line_scan_telecentric_polynomial'). Es ist zu beachten, dass für Stereosysteme, die Zeilenkameras mit telezentrischen Objektiven verwenden und die einen gemeinsamen Bewegungsvektor besitzen (d.h. *'common_motion_vector'* = *'true'*, was die Voreinstellung ist), die Optimierungsparameter des Bewegungsvektors der Referenzkamera festlegen, welche Parameter des gemeinsamen Bewegungsvektors optimiert werden. Für diese Art von System wird empfohlen, keine Parameter des Bewegungsvektors von der Optimierung auszuschließen.

'all': Allgemeiner Parametername, der für alle Kameraparameter, sowohl die internen als auch die externen, steht.

Als Standardeinstellung werden alle Parameter zum Optimieren gekennzeichnet. Eine Ausnahme davon ist es, wenn nur eine Kamera im Aufbau kalibriert wird (`NumCameras`=1). Für diesen Fall wird die Kamerapose von der Optimierung ausgeschlossen. Diese Sondereinstellung macht den Kalibrierprozess identisch mit demjenigen vom `camera_calibration`.

Daten zu den Kalibrierkörperposen

`ItemType`=*'calib_obj_pose'*: `ItemIdx` legt fest, ob Kalibriereinstellungen für alle Posen oder für einzelne Posen der Kalibrierkörper eingestellt werden sollen. Mit `ItemIdx`=*'general'* werden die neuen Einstellungen als Standardwert für alle Kalibrierkörperposen im Modell gesetzt. Wenn dagegen ein gültiger Kalibrierkörperposeindex, d.h. ein Tupel mit einem gültigen Indexpaar `[CalibObjIdx, CalibObjPoseIdx]`, in `ItemIdx` eingegeben wird, werden nur die Einstellungen der gekennzeichneten Pose geändert.

Beim Auswählen eines der folgenden Parameter in `DataName` kann eingegeben werden, **welche Parameter der Kalibrierkörperpose optimiert werden** sollen:

'calib_settings': Zusätzlich zu den bereits für die Optimierung ausgewählten Parametern zu optimierende Parameter der Kalibrierkörperpose. Standardmäßig sind alle Kameraparameter für die Optimierung ausgewählt. D.h., *'calib_settings'* ist hauptsächlich dazu geeignet, vorher ausgeschlossene Parameter wieder zur Auswahl hinzuzufügen.

'excluded_settings': Von der Optimierung ausgeschlossene Parameter der Kalibrierkörperpose.

Die folgenden Poseparameter können in `DataValue` aufgelistet werden:

'alpha',*'beta'*,*'gamma'*: Rotationswerte der Kalibrierkörperpose.

'transx',*'transy'*,*'transz'*: Translationsvektor der Kalibrierkörperpose.

'pose': Allgemeiner Parametername, der für alle Poseparameter steht.

'all': Alle Kalibrierkörperparameter, d.h. dieselben wie *'pose'*.

Die aktuellen Einstellungen der Optimierungsparameter eines Elements können mit `get_calib_data` abgefragt werden.

Daten für Hand-Auge-Kalibrierung

`ItemType`=*'tool'*: `ItemIdx` muss auf einen gültigen Kalibrierkörperposeindex gesetzt werden.

Beim Auswählen des folgenden Parameters in `DataName`, kann die Pose des Robotertools gesetzt werden:

'tool_in_base_pose': Setzen der Pose des Robotertools (in Roboterbasiskoordinaten), die für die Beobachtung der Kalibrierkörperpose mit demselben Index `ItemIdx` (entspricht dem Parameter `CalibObjPoseIdx` eines der Operatoren `find_calib_object`, `set_calib_data_observ_pose`, oder `set_calib_data_observ_points`) benutzt wurde.

Parameter

▷ **`CalibDataID`** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **`ItemType`** (input_control) . . . . . string ⇝ *string*
Typ des abgefragten Elements.
Defaultwert: 'model'
Werteliste: `ItemType` ∈ {'model', 'camera', 'calib_obj_pose', 'tool'}

▷ **`ItemIdx`** (input_control) . . . . . number(-array) ⇝ *integer* / string
Index des abgefragten Elements (je nach Auswahl in `ItemType`).
Defaultwert: 'general'
Wertevorschläge: `ItemIdx` ∈ {0, 1, 2, 'general'}

▷ **`DataName`** (input_control) . . . . . attribute.name ⇝ *string*
Name der zu setzenden Daten.
Defaultwert: 'reference_camera'
Werteliste: `DataName` ∈ {'reference_camera', 'calib_settings', 'excluded_settings', 'common_motion_vector', 'optimization_method', 'tool_in_base_pose'}

▷ **`DataValue`** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer
Werte der zu setzenden Daten.
Defaultwert: 0
Wertevorschläge: `DataValue` ∈ {0, 1, 2, 'all', 'pose', 'params', 'alpha', 'beta', 'gamma', 'transx', 'transy', 'transz', 'focus', 'magnification', 'kappa', 'poly', 'poly_tan_2', 'k1', 'k2', 'k3', 'image_plane_dist', 'tilt', 'principal_point', 'cx', 'cy', 'sx', 'sy', 'vx', 'vy', 'vz', 'true', 'false', 'linear', 'nonlinear'}

Beispiel

```
* Here, the cell size is known exactly, thus it is excluded from
* the optimization.
set_calib_data (CalibDataID, 'camera', 'general', 'excluded_settings', \
                ['sx','sy'])
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`set_calib_data_observ_points`, `find_calib_object`

Nachfolger

`calibrate_cameras`, `calibrate_hand_eye`

Modul

Calibration

set_calib_data_calib_object (: : CalibDataID, CalibObjIdx, CalibObjDescr :)

Definieren des Kalibrierkörpers im Kalibrierdatenmodell.

Der Operator `set_calib_data_calib_object` definiert den Kalibrierkörper mit Index `CalibObjIdx` im Kalibrierdatenmodell `CalibDataID`. Dementsprechend muss `CalibObjIdx` zwischen 0 und `NumCalibObjects`-1 liegen, wobei `NumCalibObjects` mit dem Operator `create_calib_data` bereits bei der Erstellung des Modells gesetzt wurde. Falls ein Kalibrierkörper mit derselben Index bereits definiert ist, wird er durch den neuen 'ausgetauscht'. Es ist zu beachten, dass alle `NumCalibObjects` Kalibrierkörper definiert werden müssen, bevor die Kamerakalibrierung (`calibrate_cameras`) durchgeführt werden kann.

Der Parameter `CalibObjDescr` wird auf zwei verschieden Weisen benutzt:

als Dateiname: Dateiname einer Kalibrierkörperbeschreibung die mit `create_caltab` oder `gen_caltab` erzeugt wurde.

als numerisches Tupel: Ein Tupel mit den 3D-Koordinaten aller Marken des Kalibrierkörpers. Die X-, Y- und Z-Koordinate müssen im Format `[X, Y, Z]`, also in der Form `[X1, ..., Xn, Y1, ..., Yn, Z1, ..., Zn]` eingegeben werden, wobei `|X| = |Y| = |Z|` und die Koordinaten in Metern angegeben werden.

Die Kalibrierkörperparameter, die bereits im Modell gespeichert sind, können mit dem Operator `get_calib_data` abgefragt werden.

Weitere detaillierte Informationen zur Kalibrierung von Kameraaufbauten befinden sich im Kapitel Kalibrierung.

Parameter

▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **CalibObjIdx** (input_control) . . . . . number ⇝ *integer*
Index des Kalibrierkörpers im Modell.
Defaultwert: 0
Wertevorschläge: `CalibObjIdx` ∈ {0, 1, 2}

▷ **CalibObjDescr** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
3D-Koordinaten oder Beschreibungsdateiname.
Werteliste: `CalibObjDescr` ∈ {'calplate.cpd', 'calplate_20mm.cpd', 'calplate_40mm.cpd', 'calplate_80mm.cpd', 'caltab_160mm.cpd', 'caltab.descr'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_calib_data`, `set_calib_data_cam_param`

Nachfolger

`set_calib_data_cam_param`, `set_calib_data_observ_points`, `find_calib_object`

Modul

Calibration

set_calib_data_cam_param (: : CalibDataID, CameraIdx, CameraType, CameraParam :)

Definieren des Typs und der Startwerte für die internen Parameter der Kamera.

Der Operator `set_calib_data_cam_param` definiert die Startwerte für die internen Parameter `CameraParam` der Kamera mit Index `CameraIdx` im Kalibrierdatenmodell `CalibDataID`. Dementsprechend muss `CameraIdx` eine Zahl zwischen 0 und `NumCameras`-1 sein, wobei `NumCameras` mit dem Operator `create_calib_data` bereits beim Erstellen des Modells festgelegt wird. Falls eine Kamera mit dem Index `CameraIdx` bereits definiert ist, werden die Kameraparameter mit `CameraParam` überschrieben. Die Auswahl der zu optimierenden Kameraparameter wird in diesem Fall aufgehoben und muss eventuell neu gesetzt werden. Es ist zu beachten, dass alle `NumCameras` Kameras definiert werden müssen, bevor die Kamerakalibrierung (`calibrate_cameras`) durchgeführt werden kann. Weitere detaillierte Informationen zur Kalibrierung von Kameraaufbauten befinden sich im Kapitel Kalibrierung.

Unten sehen Sie eine Übersicht über alle verfügbaren Kameratypen und deren jeweilige Parameter. In der Liste bezieht sich „projektive Flächenkameras“ auf die Eigenschaft, dass das Objektiv eine perspektivische Projektion auf der Objektseite des Objektivs durchführt, während sich „telezentrische Flächenkameras“ auf die Eigenschaft bezieht, dass das Objektiv eine telezentrische Projektion auf der Objektseite des Objektivs durchführt.

`CameraType` wird nur aus Gründen der Rückwärtskompatibilität als Parameter zur Verfügung gestellt. Die Information zum Kameratyp wird als erstes Element von `CameraParam` übergeben. Entsprechend muss der Parameter `CameraType` entweder wie empfohlen seinen Standardwert *[]* oder den gleichen Wert wie das erste Element von `CameraParam` haben. Für alle anderen Werte wird ein Fehler ausgegeben.

Flächenkameras mit normalen Objektiven

Projektive Flächenkameras

'area_scan_division' ['area_scan_division', Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_polynomial' ['area_scan_polynomial', Focus, K1, K2, K3, P1, P2, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

Telezentrische Flächenkameras

'area_scan_telecentric_division' ['area_scan_telecentric_division', Magnification, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_telecentric_polynomial' ['area_scan_telecentric_polynomial', Magnification, K1, K2, K3, P1, P2, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

Flächenkameras mit Tiltobjektiven

Projektive Flächenkameras

'area_scan_tilt_division' ['area_scan_tilt_division', Focus, Kappa, ImagePlaneDist, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_tilt_polynomial' ['area_scan_tilt_polynomial', Focus, K1, K2, K3, P1, P2, ImagePlaneDist, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_tilt_image_side_telecentric_division' ['area_scan_tilt_image_side_telecentric_division', Focus, Kappa, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_tilt_image_side_telecentric_polynomial' ['area_scan_tilt_image_side_telecentric_polynomial', Focus, K1, K2, K3, P1, P2, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

Telezentrische Flächenkameras

'area_scan_tilt_bilateral_telecentric_division' ['area_scan_tilt_bilateral_telecentric_division', Magnification, Kappa, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_tilt_bilateral_telecentric_polynomial' ['area_scan_tilt_bilateral_telecentric_polynomial', Magnification, K1, K2, K3, P1, P2, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_tilt_object_side_telecentric_division' ['area_scan_tilt_object_side_telecentric_division', Magnification, Kappa, ImagePlaneDist, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_tilt_object_side_telecentric_polynomial' ['area_scan_tilt_object_side_telecentric_polynomial', Magnification, K1, K2, K3, P1, P2, ImagePlaneDist, Tilt, Rot, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

Flächenkameras mit hyperzentrischen Objektiven

Projektive Flächenkameras mit hyperzentrischen Objektiven

'area_scan_hypercentric_division' ['area_scan_hypercentric_division', Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

'area_scan_hypercentric_polynomial' ['area_scan_hypercentric_polynomial', Focus, K1, K2, K3, P1, P2, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

Zeilenkameras

Projektive Flächenkameras

'line_scan_division' ['line_scan_division', Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight, Vx, Vy, Vz]

'line_scan_polynomial' ['line_scan_polynomial', Focus, K1, K2, K3, P1, P2, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight, Vx, Vy, Vz]

Telezentrische Flächenkameras

'line_scan_telecentric_division' ['line_scan_telecentric_division', Magnification, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight, Vx, Vy, Vz]

'line_scan_telecentric_polynomial' ['line_scan_telecentric_polynomial', Magnification, K1, K2, K3, P1, P2, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight, Vx, Vy, Vz]

Eine genauere Beschreibung der einzelnen Kameratypen, der Kameraparameter und vorgeschlagener Startwerte ist im Kapitel Kalibrierung zu finden.

Der Typ einer Kamera im Kalibriermodell wird im Nachhinein wieder mit dem Operator `get_calib_data` und Argumenten `ItemType`=*'camera'* und `GenParamName`=*'type'* abgefragt. Die Startwerte für die internen Parameter der Kamera werden mit dem Operator `get_calib_data` und Argumenten `ItemType`=*'camera'* und `ResultName`=*'init_params'* abgefragt.

Parameter

- ▷ **`CalibDataID`** (input_control) . . . . . calib_data ⇝ *handle*
  Handle des Kalibrierdatenmodells.
- ▷ **`CameraIdx`** (input_control) . . . . . number-array ⇝ *integer* / string
  Index der Kamera im Modell.
  Defaultwert: 0
  Wertevorschläge: `CameraIdx` ∈ {'all', 0, 1, 2}
- ▷ **`CameraType`** (input_control) . . . . . string(-array) ⇝ *string*
  Kameratyp.
  Defaultwert: []
  Werteliste: `CameraType` ∈ {[]}
- ▷ **`CameraParam`** (input_control) . . . . . campar ⇝ *real* / integer / string
  Startwerte für die internen Kameraparameter.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_calib_data`, `set_calib_data_calib_object`

Nachfolger

`set_calib_data_calib_object`, `set_calib_data_observ_points`, `find_calib_object`

Modul

Calibration

`set_calib_data_observ_points` (: : CalibDataID, CameraIdx, CalibObjIdx, CalibObjPoseIdx, Row, Column, Index, Pose :)

Punktbasierte Beobachtungsdaten im Kalibrierdatenmodell speichern.

Für Kalibrierdatenmodelle von Aufbautyp `CalibSetup`=*'calibration_object'* (see `create_calib_data`) erfolgt die Kalibrierung anhand metrischer Information, welche aus so genannten Beobachtungen der Kalibrierkörper extrahiert wird. Solche Beobachtungen werden mit dem Operator `set_calib_data_observ_points` im Modell gespeichert. Eine Beobachtung besteht aus folgenden Daten:

`CameraIdx`: Index der beobachtenden Kamera

`CalibObjIdx`: Index des beobachteten Kalibrierkörpers

`CalibObjPoseIdx`: Index der beobachteten Pose des Kalibrierkörpers. Dieser Index kann frei ausgewählt werden und muss keiner strengen Ordnung folgen. Wenn dieser Index für den Kalibrierkörper `CalibObjIdx` bereits im Modell gespeichert ist, werden die zugeordneten Beobachtungsdaten durch die neuen ersetzt. Im Gegensatz dazu kann derselbe Index Posen unterschiedlicher Kalibrierkörper problemlos zugeordnet werden.

Row, **Column**, **Index**: Extrahierte Bildkoordinaten und entsprechender Index der Kalibrierkörpermarker. Row und Column sind Tupel, die dieselbe Anzahl von Elementen enthalten. Index kann entweder ein Tupel (von derselben Länge) oder den Wert *'all'* enthalten, womit bezeichnet wird, dass die Punkte [Row, Column] eins-zu-eins den Kalibrierkörpermarker entsprechen.

Pose: Grob geschätzte 3D-Pose des Kalibrierkörpers relativ zur beobachtenden Kamera.

Wenn eine HALCON-Kalibrierplatte als Kalibrierkörper eingesetzt wird, empfehlen wir den Operator find_calib_object statt dem Operator find_marks_and_pose zu verwenden, um Bildpunkte zu extrahieren und die Kalibrierkörperpose zu schätzen, da dieser Konturinformation im Kalibrierdatenmodell speichert und so eine präzisere Kalibrierung mit calibrate_cameras ermöglicht.

Die Beobachtungsdaten können mit get_calib_data_observ_points abgefragt werden.

Parameter

▷ **CalibDataID** (input_control) ..... calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **CameraIdx** (input_control) ..... number ⇝ *integer*
Index der beobachtenden Kamera.
Defaultwert: 0
Wertevorschläge: CameraIdx ∈ {0, 1, 2}

▷ **CalibObjIdx** (input_control) ..... number ⇝ *integer*
Index des beobachteten Kalibrierkörpers.
Defaultwert: 0
Wertevorschläge: CalibObjIdx ∈ {0, 1, 2}

▷ **CalibObjPoseIdx** (input_control) ..... number ⇝ *integer*
Index der beobachteten Pose.
Defaultwert: 0
Wertevorschläge: CalibObjPoseIdx ∈ {0, 1, 2}
Restriktion: CalibObjPoseIdx >= 0

▷ **Row** (input_control) ..... number-array ⇝ *real* / integer
Zeilenkoordinaten der extrahierten Punktmerkmale.

▷ **Column** (input_control) ..... number-array ⇝ *real* / integer
Spaltenkoordinaten der extrahierten Punktmerkmale.

▷ **Index** (input_control) ..... number-array ⇝ *integer* / string
Zuordnung der extrahierten Punktmerkmale zu den Kalibrierkörpermarken.
Defaultwert: 'all'
Wertevorschläge: Index ∈ {'all', 0, 1, 2}

▷ **Pose** (input_control) ..... number-array ⇝ *real* / integer
Grob geschätzte 3D-Pose des Kalibrierkörpers relativ zur beobachtenden Kamera.
Parameteranzahl: 7

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

find_marks_and_pose, set_calib_data_cam_param, set_calib_data_calib_object

Nachfolger

set_calib_data, calibrate_cameras

Alternativen

find_calib_object

Modul

Calibration

set_camera_setup_cam_param (: : CameraSetupModelID, CameraIdx, CameraType, CameraParam, CameraPose :)

Definieren des Typs, der internen Parameter und der Pose einer Kamera im Kameraaufbaumodell.

Der Operator set_camera_setup_cam_param definiert die internen Parameter und die Pose der Kamera mit Index CameraIdx im Kameraaufbaumodell CameraSetupModelID. Der Parameter CameraIdx muss ein gültiger Kameraindex sein, d.h. eine Zahl zwischen 0 und NumCameras-1 (siehe get_camera_setup_param mit Parameter *'num_cameras'*). Wenn die Kamera mit Index CameraIdx bereits im Modell definiert ist, werden alle ihrer Parameter durch den neuen ersetzt (die Kamera wird 'ausgetauscht').

Die Anzahl der Werte in CameraParam richtet sich nach dem Kameratyp. Siehe die Beschreibung von set_calib_data_cam_param für eine Liste der Werte und Kalibrierung für detaillierte Informationen über Kameratypen und Kameraparameter.

CameraType wird nur aus Gründen der Rückwärtskompatibilität als Parameter zur Verfügung gestellt. Die Information zum Kameratyp wird als erstes Element von CameraParam übergeben. Entsprechend muss der Parameter CameraType entweder wie empfohlen seinen Standardwert *[]* oder den gleichen Wert wie das erste Element von CameraParam haben. Für alle anderen Werte wird ein Fehler ausgegeben.

Der Parameter CameraPose definiert die Pose der Kamera relativ zum Koordinatensystem des Kameraaufbaus (siehe create_camera_setup_model für eine genauere Beschreibung dieses Koordinatensystems).

Alle Parameter, die mit dem Operator set_camera_setup_cam_param gesetzt werden, können mit dem Operator get_camera_setup_param wieder abgefragt werden. Während der Kameratyp nur mit einem neuen Aufruf des Operators set_camera_setup_cam_param geändert werden kann, können alle andere Kameraparameter mit dem Operator set_camera_setup_param geändert werden.

Weitere Parameter können mit dem Operator set_camera_setup_param gesetzt werden: Standardabweichungen und/oder Kovarianzen der internen Kameraparameter.

Parameter

▷ **CameraSetupModelID** (input_control) . . . . . camera_setup_model ⇝ *handle*
Handle des Kameraaufbaumodells.

▷ **CameraIdx** (input_control) . . . . . number-array ⇝ *integer*
Index der Kamera im Modell.
Wertevorschläge: CameraIdx ∈ {0, 1, 2}

▷ **CameraType** (input_control) . . . . . string(-array) ⇝ *string*
Kameratyp.
Defaultwert: []
Werteliste: CameraType ∈ {[]}

▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
Interne Kameraparameter.

▷ **CameraPose** (input_control) . . . . . number-array ⇝ *real* / integer
Kamerapose relativ zum Koordinatensystem des Aufbaus.
Parameteranzahl: 7

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Calibration

set_camera_setup_param (: : CameraSetupModelID, CameraIdx, GenParamName, GenParamValue :)

Setzen von generischen Parametern im Kameraaufbaumodell.

Mit dem Operator set_camera_setup_param können diverse Parameter und Transformationen des Setups CameraSetupModelID gesetzt werden. Dabei werden zwei Typen von Parameter unterschieden:

Koordinatensystem des Aufbaus und Transformation der Kameraposen:

Mit CameraIdx='general' wird, je nach Auswahl im Parameter GenParamName, das Koordinatensystem des Aufbaus neu festgelegt und die Kameraposen entsprechend transformiert:

'reference_camera': Wird in GenParamValue ein gültiger Kameraindex übergeben, so werden alle Kameraposen relativ zum Koordinatensystem der gewählten Kamera neu berechnet.

'coord_transf_pose': Wird in GenParamValue ein Tupel im HALCON-Pose-Format übergeben, so wird das Koordinatensystem des Aufbaus in diese Pose verschoben. Die Pose in GenParamValue stellt dabei die Position und Orientierung des gewünschten Koordinatensystems relativ zum aktuellen dar. Alle Kameraposen werden dann relativ zum neuen Koordinatensystem neu berechnet.

Die neu berechneten Kameraposen können mit dem Operator get_camera_setup_param abgefragt werden.

Kameraparameter:

Wenn im Parameter CameraIdx eine gültiger Kameraindex (d.h. eine Zahl zwischen 0 und NumCameras-1) eingegeben wird, werden die folgenden Parameter, je nach Auswahl im Parameter GenParamName, für die ausgewählte Kamera auf den Wert im Parameter GenParamValue gesetzt:

'params': Ein Tupel mit den internen Kameraparametern.

'params_deviations': Ein Tupel mit den Standardabweichungen der internen Kameraparameter. Das Tupel soll einen Wert für jeden internen Kameraparameter (siehe *'params'*) bis auf die Parameter CameraType, Width und Height enthalten, d.h. |params_deviations|=|params|-3.

'params_covariances': Ein Tupel mit der Kovarianzmatrix der internen Kameraparameter. Das Tupel muss eine quadratische symmetrische Matrix darstellen, deren beide Dimensionen gleich groß sind wie die Anzahl der Standardabweichungswerte, d.h., |params_covariances|=|params_deviations|^2=(|params|-3)^2.

'pose': Ein Tupel mit der Kamerapose in Form einer HALCON-Pose, relativ zum Koordinatensystem des Aufbaus. Siehe oben für mehr Information.

Es ist zu beachten, dass eine Kamera im Modell bereits definiert sein muss, um ihre Parameter mit set_camera_setup_param ändern zu können. Wenn CameraIdx der Index einer undefinierten Kamera ist, gibt der Operator einen Fehler zurück.

Alle Parameter können mit dem Operator get_camera_setup_param wieder abgefragt werden.

Parameter

▷ **CameraSetupModelID** (input_control) . . . . . camera_setup_model ⇝ *handle*
Handle des Kameraaufbaumodells.

▷ **CameraIdx** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Index der Kamera im Modell.
Defaultwert: 0
Wertevorschläge: CameraIdx ∈ {0, 1, 2, 'general'}

▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
Namen der generischen Parameter.
Werteliste: GenParamName ∈ {'params', 'params_deviations', 'params_covariances', 'pose', 'reference_camera', 'coord_transf_pose'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *real* / integer / string
Werte der generischen Parameter.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_camera_setup_model`, `read_camera_setup_model`

Modul

Calibration

write_calib_data (: : CalibDataID, FileName :)

Schreiben des Kalibrierdatenmodells in eine Datei.

Der Operator `write_calib_data` schreibt ein Kalibrierdatenmodell `CalibDataID` in die Datei `FileName`. Folgende Information wird in der Datei gespeichert:

- Initiale Kameraeinstellungen
- Beschreibung der Kalibrierkörper
- Beobachtungsdaten
- Modelleinstellungen: die generischen und die spezifischen Optimierungseinstellungen für die Kameras bzw. Kalibrierkörper

Beachten Sie, dass keine Ergebnisse der Kalibrierung gespeichert werden. Diese können stattdessen mit `get_calib_data` abgefragt werden und einzeln oder gemeinsam in Form eines Kameraaufbaumodells gespeichert werden.

Das Kalibrierdatenmodell kann mit dem Operator `read_calib_data` eingelesen werden.

Parameter

▷ **CalibDataID** (input_control) . . . . . calib_data ⇝ *handle*
Handle des Kalibrierdatenmodells.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Der Dateiname des zu speichernden Modells.
Dateiendung: `.ccd`

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- CalibDataID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Modul

Calibration

write_camera_setup_model (: : CameraSetupModelID, FileName :)

Schreiben eines Kameraaufbaumodells in eine Datei.

Der Operator `write_camera_setup_model` schreibt ein Kameraaufbaumodell `CameraSetupModelID` in die Datei `FileName`.

Das Kameraaufbaumodell wird im Nachhinein mit dem Operator `read_camera_setup_model` gelesen.

Parameter

▷ **CameraSetupModelID** (input_control) . . . . . camera_setup_model ⇝ *handle*
Handle des Kameraaufbaumodells.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Der Dateiname des Modells.
Dateiendung: `.csm`

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Calibration

14.7 Monokular

```
camera_calibration ( : : NX, NY, NZ, NRow, NCol, StartCamParam,
    NStartPose, EstimateParams : CameraParam, NFinalPose, Errors )
```

Bestimmen aller Kameraparameter durch simultane Ausgleichsrechnung.

`camera_calibration` führt die Kalibrierung einer einzelnen Kamera durch. Dazu werden bekannte 3D-Modellpunkte (mit Koordinaten `NX`, `NY`, `NZ`) ins Bild projiziert und die Summe der Abstandsquadrate zwischen den projizierten 3D-Koordinaten und ihren korrespondierenden Bildpunktkoordinaten (`NRow`, `NCol`) minimiert.

Als Startwerte für das Ausgleichsverfahren dienen die externen (`NStartPose`) und internen (`StartCamParam`) und Kameraparameter. Dabei wird die 3D-Lage `NStartPose` mit den initialen Werten für die externen Kameraparameter in der Form ${}^{ccs}\mathbf{P}_{wcs}$ erwartet. Dabei steht *ccs* für das Kamerakoordinatensystem und *wcs* für das Weltkoordinatensystem (siehe auch Transformationen / Posen und „`Solution Guide III-C - 3D Vision`"). Bestimmte Kameraparameter können mit `EstimateParams` ausdrücklich in die Minimierung ein- oder von ihr ausgeschlossen werden. Eine ausführliche Beschreibung der verfügbaren Kameramodelle sowie der dazugehörigen Kameraparameter findet sich in Kalibrierung.

Für eine erfolgreiche Kalibrierung wird mindestens ein Kalibrierobjekt mit genau bekannten metrischen Abmessungen benötigt, z.B. eine HALCON- Kalibrierplatte. Vor dem Aufruf von `camera_calibration` nehmen Sie eine Serie von Bildern des Kalibrierobjektes in unterschiedlichen Orientierungen auf, so dass das gesamte Blickfeld bzw. Messvolumen ausgefüllt wird. Der Erfolg der Kalibrierung hängt im hohen Maße von der Qualität der Kalibrierbilder ab. Deshalb wird bei der Aufnahme der Kalibrierbilder hohe Sorgfalt empfohlen. Für weitere Informationen sei auf den Abschnitt „Wie nehme ich geeignete Bilder auf?'"in Kalibrierung verwiesen.

Nach erfolgreicher Kalibrierung gibt `camera_calibration` die optimierten internen (`CameraParam`) und externen (`NFinalPose` ${}^{ccs}\mathbf{P}_{wcs}$) Kameraparameter der Kamera zurück. Zusätzlich liefert er die Wurzel des mittleren quadratischen Rückprojektionsfehlers (RMSE) der Optimierung im Parameter `Errors` (in Pixel) zurück. Dieser Fehler gibt einen allgemeinen Hinweis darauf, ob die Optimierung erfolgreich war.

Hinweise zur Kalibrierung

Wie extrahiert man die Markenmittelpunkte aus den Bildern? Wird eine HALCON-Kalibrierplatte verwendet, so können mit Hilfe des Operators `find_calib_object` für jedes Bild sowohl die Koordinaten der Markenmittelpunkte als auch eine grobe Schätzung der externen Kameraparameter bestimmt werden. Bei HALCON-Kalibrierplatten mit rechteckiger Markenanordnung (siehe `gen_caltab`) kann alternativ auch eine Kombination der Operatoren `find_caltab` und `find_marks_and_pose` verwendet werden. In beiden Fällen können die auf diese Weise erhaltenen Werte dann direkt als Startwerte für die externen Kameraparameter (`NStartPose`) verwendet werden.

Diejenigen Bilder, auf denen die Suche nach dem Kalibrierkörper (`find_caltab`) gescheitert ist bzw. für die der Operator `find_marks_and_pose` oder `find_calib_object` nicht die Markenpunkte detektieren konnte, sollten nicht in die Eingabeparameter von `camera_calibration` eingehen.

Wie erhält man die benötigten Startwerte für die Kalibrierung? Bei Verwendung einer HALCON-Kalibrierplatte sind die Eingabeparameter NX, NY und NZ in der dazugehörigen Beschreibungsdatei gespeichert. Sie können einfach mit caltab_points ausgelesen werden. Startwerte für die internen Kameraparameter (StartCamParam) sind in der Regel den Spezifikationen der Kamera zu entnehmen. Für weitere Informationen sei auf das Kapitel Kalibrierung verwiesen. Startwerte für die Lagen des Kalibrierkörpers sowie der Markenmittelpunkte (NRow und NCol) können für jedes Bild mit dem Operator find_calib_object ermittelt werden. Das Tupel NStartPose ergibt sich dann als Konkatenation dieser einzelnen Kameralagen.

Welche der Kameraparameter werden geschätzt? Mit dem Eingabeparameter EstimateParams kann angeben werden, welche der gesuchten Kameraparameter tatsächlich bestimmt werden sollen. Üblicherweise ist dieser Wert gleich *'all'*, d.h. alle 6 externen Kameraparameter (Translation und Rotation) und alle internen Kameraparameter werden in der Ausgleichung geschätzt. Falls die internen Parameter der Kamera bereits bekannt sind (z.B. durch einen früheren Aufruf von camera_calibration), ist es oftmals ausreichend, lediglich die 3D-Lage (Pose) des Weltkoordinatensystems in Kamerakoordinaten zu bestimmen. In diesem Fall kann in EstimateParams der Wert *'pose'* übergeben werden. Um denselben Effekt zu erzielen, kann man als Alternative dem Parameter EstimateParams auch alle zu bestimmenden Werte übergeben: EstimateParams = *['alpha','beta','gamma','transx','transy','transz']*. Andernfalls enthält EstimateParams ein Tupel von Strings, die die Kombination von Parametern, die geschätzt werden soll, enthält. Es ist auch möglich, Parameter die nicht geschätzt werden sollen über ein vorgesetztes '~' Zeichen im String auszuschließen. Die Werte *['pose','~transx']* beispielsweise haben denselben Effekt wie *['alpha','beta','gamma','transy','transz']*. *['all','~focus']* zum Beispiel schätzt alle internen und externen Parameter bis auf die Brennweite. Mit Ausnahme von *'all'* kann das Präfix '~' jedem Parameterwert vorangestellt werden.

Welche Beschränkungen bestehen bei der Bestimmung der Kameraparameter? Weiterführende Informationen über generelle Beschränkungen bei der Bestimmung der Kameraparameter finden sich im Abschnitt „Weitere Beschränkungen in Bezug auf spezifische Kameratypen'"des Kapitels Kalibrierung.

Wie sieht die Reihenfolge der einzelnen Parameter aus? Die Länge des Tupels NStartPose korrespondiert mit der Anzahl der Kalibrierbilder, d.h. werden beispielsweise 15 Aufnahmen eines Kalibrierkörpers verwendet, so ergibt sich die Länge des Tupels NStartPose zu $15 \cdot 7 = 105$ (15 mal je 7 externe Kameraparameter). Die ersten 7 Werte entsprechen daher der Lage des Kalibrierkörpers im 1. Bild, die nächsten 7 derjenigen im 2. Bildes usw.

Die dadurch festgelegte Anzahl der Kalibrierbilder muss auch bei den Tupeln mit den Koordinaten der 3D-Modellmarkenpunkte bzw. der extrahierten 2D-Markenpunkte berücksichtigt werden. Bei einer Anzahl von 15 Kalibrierbildern müssen daher die Tupel NRow und NCol 15 mal so viele Werte aufweisen wie die Tupel mit den 3D-Modellmarkenpunkten (NX, NY und NZ). Sind beispielsweise 49 Markenpunkte pro Bild vorhanden, so haben die Tupel NRow und NCol jeweils die Länge $15 \cdot 49 = 735$, die Tupel NX, NY und NZ haben dagegen die Länge 49. Die Reihenfolge der Werte in NRow und NCol erfolgt dabei bildweise, d.h. bei 49 Marken korrespondiert der erste 3D-Modellpunkt mit dem 1., 50., 99., 148., 197., 246. etc. extrahierten 2D-Markenpunkt.

Was bedeuten die Ausgabeparameter? Nach erfolgreicher Kalibrierung sind in den Ausgabeparametern CameraParam und NFinalPose die ausgeglichenen Werte der internen und externen Kameraparameter enthalten. Die Länge des Tupels NFinalPose entspricht dann derjenigen von NStartPose.

Die Darstellungstypen von NFinalPose entsprechen dem Darstellungstyp der des ersten Tupels aus NStartPose (siehe create_pose). Der Darstellungstyp kann jedoch mit convert_pose_type gewandelt werden.

Als weiterer Parameter wird der durchschnittliche Fehler (Errors) ausgegeben, mit welchem die erzielte Genauigkeit der Kalibrierung abgeschätzt werden kann. Der Fehler (Root-Mean-Square-Fehler der Position) wird in Pixel angegeben. Bei der Kalibrierung von nur einer einzelnen Kamera ist ein Fehler in der Größenordnung von 0.1 Pixel (der typische Detektionsfehler beim Extrahieren der Koordinaten der projizierten Kalibrierkörpermarken) ein Hinweis dafür, dass die Optimierung gut zu den Beobachtungsdaten passt. Wenn der Fehler deutlich über 0.1 Pixel liegt, wurde die Kalibrierung nicht gut ausgeführt. Die Gründe hierfür können z.B. eine schlechte Bildqualität, eine unzureichende Anzahl an Kalibrierbildern oder auch eine fehlerhafte Kalibrierplatte sein.

Muss man einen planaren Kalibrierkörper verwenden? Es ist **nicht** nötig, einen planaren Kalibrierkörper zu verwenden. Der Operator camera_calibration ist so ausgelegt, dass er mit den Eingabetupeln NX, NY, NZ, NRow und NCol beliebige 3D/2D-Korrespondenzen verarbeiten kann. Der Aufbau der Parameter kann dem Abschnitt „Wie sieht die Reihenfolge der einzelnen Parameter aus¿‘ entnommen werden.

Es spielt daher prinzipiell keine Rolle, wie man die nötigen 3D-Modellmarkenpunkte und die korrespondierenden 2D-Markenpunkte ermittelt. Somit ist es einerseits möglich, auch einen 3D-Kalibrierkörper zu verwenden. Andererseits kann man auch beliebige markante Punkte (z.B. natürliche Landmarken) verwenden, deren Positionen in der Welt bekannt sind. Falls man dann `EstimateParams` auf *'pose'* setzt, kann man auf diese Weise mit `camera_calibration` die Lage eines beliebigen Objektes in Kamerakoordinaten ermitteln! Hierzu sind dann mindestens drei 3D/2D-Korrespondenzen als Eingabe anzugeben. `NStartPose` kann z.B. direkt über `create_pose` generiert werden wie es im dazugehörigen Programmbeispiel gezeigt wird.

Achtung

Das Ergebnis der Kalibrierung hängt von den Startwerten für die internen (`StartCamParam`) und externen (`NStartPose`) Kameraparameter ab. Anhand der zurückgegebenen durchschnittlichen Fehler (`Errors`) kann die Genauigkeit der Kalibrierung abgeschätzt werden. Die Fehler (Abweichungen in x- und y-Koordinate) werden in Pixel angegeben.

Bei der Kalibrierung von Kameras mit Zeilensensor kann der Startwert für den internen Kameraparameter Sy auf den den Wert 0.0 gesetzt werden. In diesem Fall ist es allerdings nicht möglich, die Position des Bildhauptpunktes in y-Richtung zu bestimmen. Im Eingabeparameter `EstimateParams` muss aus diesem Grund explizit die y-Koordinate des Bildhauptpunktes ausgeschlossen werden: *'~cy'*. Der effektive Abstand des Bildhauptpunktes von der Sensorzeile ist dann immer $p_v = S_y \cdot C_y = 0.0$. Weitere Informationen können dem Abschnitt „Weitere Beschränkungen in Bezug auf spezifische Kameratypen'"in Kalibrierung entnommen werden.

Parameter

- ▷ **NX** (input_control) ..... number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen x-Koordinaten der Kalibriermarken (in Meter).
- ▷ **NY** (input_control) ..... number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen y-Koordinaten der Kalibriermarken (in Meter).
  Parameteranzahl: NY == NX
- ▷ **NZ** (input_control) ..... number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen z-Koordinaten der Kalibriermarken (in Meter).
  Parameteranzahl: NZ == NX
- ▷ **NRow** (input_control) ..... number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Zeilen-Koordinaten der extrahierten Kalibriermarken (in Pixel).
- ▷ **NCol** (input_control) ..... number-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Spalten-Koordinaten der extrahierten Kalibriermarken (in Pixel).
  Parameteranzahl: NCol == NRow
- ▷ **StartCamParam** (input_control) ..... campar ⇝ *real* / integer / string
  Startwerte für die internen Kameraparameter.
- ▷ **NStartPose** (input_control) ..... pose(-array) ⇝ *real* / integer
  Geordnetes Tupel mit allen Startwerten der externen Kameraparameter.
  Parameteranzahl: NStartPose == 7 * NRow / NX
- ▷ **EstimateParams** (input_control) ..... string-array ⇝ *string*
  Zu schätzenden Kameraparameter.
  Defaultwert: 'all'
  Werteliste: `EstimateParams` ∈ {'all', 'pose', 'camera', 'alpha', 'beta', 'gamma', 'transx', 'transy', 'transz', 'focus', 'magnification', 'kappa', 'poly', 'k1', 'k2', 'k3', 'poly_tan_2', 'image_plane_dist', 'tilt', 'cx', 'cy', 'sx', 'sy', 'vx', 'vy', 'vz'}
- ▷ **CameraParam** (output_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **NFinalPose** (output_control) ..... pose(-array) ⇝ *real* / integer
  Geordnetes Tupel mit allen externen Kameraparametern.
  Parameteranzahl: NFinalPose == 7 * NRow / NX
- ▷ **Errors** (output_control) ..... real(-array) ⇝ *real*
  Durchschnittlicher Fehler in Pixel.

Beispiel

```
* Read calibration images.
read_image(Image1, 'calib/grid_space.cal.k.000')
```

```
read_image(Image2, 'calib/grid_space.cal.k.001')
read_image(Image3, 'calib/grid_space.cal.k.002')
*  Find calibration pattern.
find_caltab(Image1, CalPlate1, 'caltab_big.descr', 3, 112, 5)
find_caltab(Image2, CalPlate2, 'caltab_big.descr', 3, 112, 5)
find_caltab(Image3, CalPlate3, 'caltab_big.descr', 3, 112, 5)
*  Find calibration marks and start poses.
StartCamPar := ['area_scan_division', 0.008, 0.0, 0.000011, 0.000011, \
                384, 288, 768, 576]
find_marks_and_pose(Image1, CalPlate1, 'caltab_big.descr', StartCamPar, \
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1, \
                    StartPose1)
find_marks_and_pose(Image2, CalPlate2, 'caltab_big.descr', StartCamPar, \
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord2, CCoord2, \
                    StartPose2)
find_marks_and_pose(Image3, CalPlate3, 'caltab_big.descr', StartCamPar, \
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord3, CCoord3, \
                    StartPose3)
*  Read 3D positions of calibration marks.
caltab_points('caltab_big.descr', NX, NY, NZ)
*  Camera calibration.
camera_calibration(NX, NY, NZ, [RCoord1, RCoord2, RCoord3], \
                   [CCoord1, CCoord2, CCoord3], StartCamPar, \
                   [StartPose1, StartPose2, StartPose3], 'all', \
                   CameraParam, NFinalPose, Errors)
*  Write internal camera parameters to file.
write_cam_par(CameraParam, 'campar.dat')
```

Ergebnis

Sind die Parameterwerte korrekt und konnten die gesuchten Kameraparameter durch das Bündelausgleichsverfahren bestimmt werden, dann liefert `camera_calibration` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_marks_and_pose`, `caltab_points`, `read_cam_par`

Nachfolger

`write_pose`, `pose_to_hom_mat3d`, `disp_caltab`, `sim_caltab`

Alternativen

`calibrate_cameras`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `disp_caltab`, `sim_caltab`, `write_cam_par`, `read_cam_par`, `create_pose`, `convert_pose_type`, `write_pose`, `read_pose`, `pose_to_hom_mat3d`, `hom_mat3d_to_pose`, `caltab_points`, `gen_caltab`, `calibrate_cameras`

Modul

Calibration

14.8 Projektion

cam_par_pose_to_hom_mat3d (: : CameraParam, Pose : HomMat3D)

Konvertieren von internen Kameraparametern und einer 3D-Lage (Pose) in eine 3×4 Projektionsmatrix.

cam_par_pose_to_hom_mat3d konvertiert die internen Kameraparameter CameraParam und die 3D-Lage Pose, die die externen Kameraparameter beschreibt, in eine 3×4 Projektionsmatrix HomMat3D, die dazu verwendet werden kann, Punkte von 3D nach 2D zu projizieren. Dabei kann die Konversion nur erfolgen, falls die Verzeichnungskoeffizienten in CameraParam 0 sind. Falls nötig muss change_radial_distortion_cam_par verwendet werden, um dies zu erreichen. Die internen Kameraparameter und die 3D-Lage werden typischerweise mit calibrate_cameras bestimmt.

Parameter

- ▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **Pose** (input_control) . . . . . pose ⇝ *real* / integer
  3D-Lage (Pose).
  Parameteranzahl: 7
- ▷ **HomMat3D** (output_control) . . . . . hom_mat3d ⇝ *real*
  3×4 Projektionsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert cam_par_pose_to_hom_mat3d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

calibrate_cameras, change_radial_distortion_cam_par

Nachfolger

project_point_hom_mat3d, project_hom_point_hom_mat3d

Siehe auch

create_pose, hom_mat3d_to_pose, project_3d_point, get_line_of_sight

Modul

Foundation

project_3d_point (: : X, Y, Z, CameraParam : Row, Column)

Projizieren von 3D-Punkten in Bildkoordinaten (Subpixel).

project_3d_point projiziert einen oder mehrere 3D-Punkte mit den Koordinaten X, Y und Z in das Bildkoordinatensystem (in Pixel) und liefert das Ergebnis in Row und Column zurück. Die Koordinaten X, Y und Z sind Kamerakoordinaten, entsprechen also der Position der Punkte relativ zur Kamera.

Die internen Kameraparameter CameraParam beschreiben dabei die Abbildungseigenschaften der Kamera (siehe Kalibrierung).

Parameter

- ▷ **X** (input_control) . . . . . point3d.x-array ⇝ *real*
  X-Koordinaten der zu projizierenden 3D-Punkte im Kamerakoordinatensystem.
- ▷ **Y** (input_control) . . . . . point3d.y-array ⇝ *real*
  Y-Koordinaten der zu projizierenden 3D-Punkte im Kamerakoordinatensystem.

▷ **Z** (input_control) . . . . . point3d.z-array ⇝ *real*
Z-Koordinaten der zu projizierenden 3D-Punkte im Kamerakoordinatensystem.

▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
Interne Kameraparameter.

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinaten der projizierten Punkte (in Pixel).

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinaten der projizierten Punkte (in Pixel).

Beispiel

```
* Set internal camera parameters and pose of the world coordinate
* system in camera coordinates.
* Note that, typically, these values are the result of a prior
* calibration.
gen_cam_par_area_scan_division (0.01, -731, 5.2e-006, 5.2e-006, \
                                654, 519, 1280, 1024, CameraParam)
create_pose (0.1, 0.2, 0.3, 40, 50, 60, \
             'Rp+T', 'gba', 'point', WorldPose)
* Convert pose into transformation matrix.
pose_to_hom_mat3d(WorldPose, HomMat3D)
* Transform 3D points from world into the camera coordinate system.
affine_trans_point_3d(HomMat3D, [3.0, 3.2], [4.5, 4.5], [3.8, 4.2], X, Y, Z)
* Project 3D points into image.
project_3d_point(X, Y, Z, CameraParam, Row, Column)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `project_3d_point` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_cam_par`, `affine_trans_point_3d`

Nachfolger

`gen_region_points`, `gen_region_polygon`, `disp_polygon`

Siehe auch

`camera_calibration`, `disp_caltab`, `read_cam_par`, `get_line_of_sight`, `affine_trans_point_3d`, `image_points_to_world_plane`

Modul

Calibration

project_hom_point_hom_mat3d (: : HomMat3D, Px, Py, Pz, Pw : Qx, Qy, Qw)

Projizieren eines homogenen 3D-Punktes mittels einer 3×4 Projektionsmatrix.

`project_point_hom_mat3d` wendet die 3×4 Projektionsmatrix `HomMat3D` auf alle homogenen Eingabepunkte (`Px`,`Py`,`Pz`,`Pw`) an und liefert ein Tupel von Ergebnispunkten (`Qx`,`Qy`,`Qw`) zurück. Die Transformation wird durch eine 3×4 Projektionsmatrix beschrieben, die in `HomMat3D` übergeben wird. Dies entspricht den folgenden Gleichungen (Ein- und Ausgabepunkte als homogene Vektoren dargestellt):

$$\begin{pmatrix} \texttt{Qx} \\ \texttt{Qy} \\ \texttt{Qw} \end{pmatrix} = \texttt{HomMat3D} \cdot \begin{pmatrix} \texttt{Px} \\ \texttt{Py} \\ \texttt{Pz} \\ \texttt{Pw} \end{pmatrix}$$

Um die homogenen Koordinaten in euklidische Koordinaten zu transformieren, muss durch Qw geteilt werden:

$$\begin{pmatrix} Ex \\ Ey \end{pmatrix} = \begin{pmatrix} \frac{Qx}{Qw} \\ \frac{Qy}{Qw} \end{pmatrix}$$

Dies kann direkt mit project_point_hom_mat3d erreicht werden. Daher ist project_hom_point_hom_mat3d hauptsächlich in Situationen nützlich, in denen die Ergebnispunkte auf der unendlich fernen Geraden liegen können, d.h. $\texttt{Qw} = 0$ besitzen können, und somit die obige Division nicht ausgeführt werden kann.

Es ist zu beachten, dass, konsistent mit den Konventionen in calibrate_cameras, Qx der Spaltenkoordinate und Qy der Zeilenkoordinate in einem Bild entspricht.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  3×4 Projektionsmatrix.
- ▷ **Px** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (x-Koordinate).
- ▷ **Py** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (y-Koordinate).
- ▷ **Pz** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (z-Koordinate).
- ▷ **Pw** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (w-Koordinate).
- ▷ **Qx** (output_control) . . . . . real(-array) ⇝ *real*
  Ausgabepunkt (x-Koordinate).
- ▷ **Qy** (output_control) . . . . . real(-array) ⇝ *real*
  Ausgabepunkt (y-Koordinate).
- ▷ **Qw** (output_control) . . . . . real(-array) ⇝ *real*
  Ausgabepunkt (w-Koordinate).

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

cam_par_pose_to_hom_mat3d

Alternativen

project_point_hom_mat3d, project_3d_point

Modul

Foundation

project_point_hom_mat3d (: : HomMat3D, Px, Py, Pz : Qx, Qy)

Projizieren eines 3D-Punktes mittels einer 3×4 Projektionsmatrix.

project_point_hom_mat3d wendet die 3×4 Projektionsmatrix HomMat3D auf alle Eingabepunkte (Px,Py,Pz) an und liefert ein Tupel von Ergebnispunkten (Qx,Qy) zurück. Die Transformation wird durch eine

homogene Transformationsmatrix beschrieben, die in HomMat3D übergeben wird. Dies entspricht den folgenden Gleichungen (Ein- und Ausgabepunkte als homogene Vektoren dargestellt):

$$\begin{pmatrix} Tx \\ Ty \\ Tw \end{pmatrix} = \texttt{HomMat3D} \cdot \begin{pmatrix} \texttt{Px} \\ \texttt{Py} \\ \texttt{Pz} \\ 1 \end{pmatrix}$$

project_point_hom_mat3d transformiert hierauf die homogenen Koordinaten in euklidische Koordinaten, indem durch Tw geteilt wird:

$$\begin{pmatrix} \texttt{Qx} \\ \texttt{Qy} \end{pmatrix} = \begin{pmatrix} \frac{Tx}{Tw} \\ \frac{Ty}{Tw} \end{pmatrix}$$

Falls bei dieser Transformation ein Punkt auf der unendlich fernen Geraden erzeugt wird ($Tw = 0$), wird eine Fehlermeldung zurückgeliefert. Falls dies nicht erwünscht ist, kann project_hom_point_hom_mat3d verwendet werden.

Es ist zu beachten, dass, konsistent mit den Konventionen in calibrate_cameras, Qx der Spaltenkoordinate und Qy der Zeilenkoordinate in einem Bild entspricht.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  3×4 Projektionsmatrix.
- ▷ **Px** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (x-Koordinate).
- ▷ **Py** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (y-Koordinate).
- ▷ **Pz** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (z-Koordinate).
- ▷ **Qx** (output_control) . . . . . real(-array) ⇝ *real*
  Ausgabepunkt (x-Koordinate).
- ▷ **Qy** (output_control) . . . . . real(-array) ⇝ *real*
  Ausgabepunkt (y-Koordinate).

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

cam_par_pose_to_hom_mat3d

Alternativen

project_hom_point_hom_mat3d, project_3d_point

Modul

Foundation

14.9 Rektifizierung

change_radial_distortion_cam_par (: : Mode, CamParamIn, DistortionCoeffs : CamParamOut)

Bestimmen neuer Kameraparameter gemäß eines vorgegebenen radialen Verzeichnungskoeffizienten.

change_radial_distortion_cam_par modifiziert die internen Kameraparameter CamParamIn gemäß den vorgegebenen radialen Verzeichnungskoeffizienten DistortionCoeffs und gibt sie in CamParamOut

zurück. Der Operator kann nur für Flächenkameras (mit beliebigen Objektivtypen) und für Zeilenkameras mit telezentrischen Objektiven verwendet werden. Zeilenkameras mit perspektivischen Objektiven werden nicht unterstützt.

Über den Parameter `Mode` werden dabei folgende Modi unterschieden:

'fixed' Die Kameraparameter bleiben bis auf die radialen Verzeichnungskoeffizienten unverändert. Dies bewirkt i.A. eine Änderung des Bildausschnittes.

'fullsize' Für Flächenkameras werden die Skalierungsfaktoren S_x und S_y und der Hauptpunkt $(C_x, C_y)^T$ so angepasst, dass der Bildausschnitt unverändert bleibt. Für Zeilenkameras mit telezentrischen Objektiven werden der Skalierungsfaktor S_x, der Hauptpunkt $(C_x, C_y)^T$ und die V_y-Komponente des Bewegungsvektors angepasst, um diesen Effekt zu erzielen. Konkret wird dabei sichergestellt, dass alle Bildpunkte des Originalkamerabildes auch im modifizierten (entzerrten) Kamerabild abgebildet werden. Im Allgemeinen entstehen dabei undefinierte Pixel im modifizierten Bild.

'adaptive' Ein Kompromiss zwischen den beiden anderen Modi: Der Bildausschnitt wird etwas verkleinert, um undefinierte Bildbereiche zu vermeiden. Hier werden dieselben Parameter wie bei *'fullsize'* modifiziert.

'preserve_resolution' Wie beim Modus *'fullsize'* wird sichergestellt, dass alle Bildpunkte des Originalkamerabildes auch im modifizierten (entzerrten) Kamerabild abgebildet werden. Für Flächenkameras werden die Skalierungsfaktoren S_x und S_y und der Hauptpunkt $(C_x, C_y)^T$ so angepasst, dass der Bildausschnitt unverändert bleibt. Für Zeilenkameras mit telezentrischen Objektiven werden der Skalierungsfaktor S_x, der Hauptpunkt $(C_x, C_y)^T$ und ggf. die V_y-Komponente des Bewegungsvektors angepasst, um diesen Effekt zu erzielen. Im Allgemeinen entstehen dabei undefinierte Pixel im modifizierten Bild. Im Unterschied zum Modus *'fullsize'* wird jedoch zusätzlich die Bildgröße so erhöht, dass sich die Bildauflösung in keinem Teil des Bildes verringert.

In allen Fällen sind die Verzeichnungskoeffizienten in den Ausgabekameraparametern `CamParamOut` gleich `DistortionCoeffs`. Für telezentrische Zeilenkameras beeinflusst der Bewegungsvektor auch die wahrgenommene Verzeichnung. So erzeugt ein $V_x \neq 0$ zum Beispiel Pixel mit schräggestellten Achsen. Weiterhin hat $V_y \neq S_x$/Magnification zur Folge, dass die Pixel nicht quadratisch erscheinen. Daher können für telezentrische Zeilenkameras bis zu drei weitere Komponenten zusätzlich zu κ bzw. $(K_1, K_2, K_3, P_1, P_2)$ in `DistortionCoeffs` übergeben werden. Diese spezifizieren die neuen V_x-, V_y- und V_z-Komponenten des Bewegungsvektors.

Die Rückrechnung der Pixel im modifizierten Bild in die Bildebene gemäß der neuen internen Kameraparameter ergibt die gleichen Punkte wie die Rückrechnung der korrespondierenden Subpixel im Originalbild gemäß der alten internen Kameraparameter `CamParamIn`.

Parameter

▷ **`Mode`** (input_control) . . . . . string ⇝ *string*
Modus
Defaultwert: 'adaptive'
Wertevorschläge: `Mode` ∈ {'fullsize', 'adaptive', 'fixed', 'preserve_resolution'}

▷ **`CamParamIn`** (input_control) . . . . . campar ⇝ *real* / integer / string
Interne Kameraparameter (Original).

▷ **`DistortionCoeffs`** (input_control) . . . . . real(-array) ⇝ *real* / integer
Gewünschte radiale Verzeichnungskoeffizienten.
Parameteranzahl: DistortionCoeffs == 1 || DistortionCoeffs == 5
Defaultwert: 0.0

▷ **`CamParamOut`** (output_control) . . . . . campar ⇝ *real* / integer / string
Interne Kameraparameter (modifiziert).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `change_radial_distortion_cam_par` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

camera_calibration, read_cam_par

Nachfolger

change_radial_distortion_image, change_radial_distortion_contours_xld, gen_radial_distortion_map

Siehe auch

camera_calibration, read_cam_par, change_radial_distortion_image, change_radial_distortion_contours_xld, change_radial_distortion_points

Modul

Calibration

change_radial_distortion_contours_xld (
Contours : ContoursRectified : CamParamIn, CamParamOut :)

Verändern der radialen Verzeichnung von Konturen.

change_radial_distortion_contours_xld verändert die radiale Verzeichnung der Konturen Contours gemäß den übergebenen internen Kameraparametern CamParamIn bzw. CamParamOut. Dazu wird jedes Subpixel der Eingabekonturen mittels CamParamIn in die Bildebene zurückgerechnet und dann mittels CamParamOut in ein Subpixel der entsprechenden Ausgabekontur überführt.

Wird zur Bestimmung der modifizierten Kameraparameter CamParamOut die Routine change_radial_distortion_cam_par verwendet, entsprechen die Ausgabekonturen ContoursRectified den Eingabekonturen bei einer Aufnahmeoptik mit veränderter radialer Verzeichnung κ . Für $\kappa = 0$ werden die Konturen radial entzerrt. Eine etwaige Lagebestimmung für die Kamera (externe Kameraparameter) wird dadurch nicht beeinflusst.

Beachten Sie, dass change_radial_distortion_contours_xld nicht mit Zeilenkameras mit perspektivischen Objektiven funktioniert.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabekonturen.
- ▷ **ContoursRectified** (output_object) ..... xld_cont(-array) ⇝ *object*
  Ausgabekonturen mit veränderter radialer Verzeichnung.
- ▷ **CamParamIn** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter für Contours.
- ▷ **CamParamOut** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter für ContoursRectified.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

change_radial_distortion_cam_par, gen_contours_skeleton_xld, edges_sub_pix, smooth_contours_xld

Nachfolger

gen_polygons_xld, smooth_contours_xld

Siehe auch

change_radial_distortion_cam_par, camera_calibration, read_cam_par, change_radial_distortion_image, change_radial_distortion_points

Modul

Calibration

change_radial_distortion_image (Image,
Region : ImageRectified : CamParamIn, CamParamOut :)

Verändern der radialen Verzeichnung eines Bildes.

change_radial_distortion_image verändert die radiale Verzeichnung des Eingabebildes Image gemäß den übergebenen internen Kameraparametern CamParamIn bzw. CamParamOut. Dazu wird jedes Pixel im Ausgabebild, das innerhalb der Region Region liegt, mittels CamParamOut in die Bildebene zurückgerechnet und dann mittels CamParamIn in ein Subpixel im Eingabebild überführt. Der resultierende Grauwert wird aus dem Eingabebild durch bilineare Interpolation abgeleitet. Liegt das Subpixel außerhalb des Eingabebildes, wird das Pixel im Ausgabebild auf 'schwarz' gesetzt und aus der Ausgaberegion entfernt.

Sollen die Grauwerte für alle Pixel im Ausgabebild berechnet werden, so genügt es, in Region ein leeres Objekt zu übergeben (das vorher z.B. mit gen_empty_obj erzeugt worden ist). Dies ist besonders dann hilfreich, wenn sich die Größe des Ausgabebildes von der des Eingabebildes unterscheidet und somit nicht einfach die Region des Eingabebildes in Region übergeben werden kann.

Wird zur Bestimmung der modifizierten Kameraparameter CamParamOut die Routine change_radial_distortion_cam_par verwendet, entspricht das Ausgabebild ImageRectified dem Eingabebild bei einer Aufnahmeoptik mit geänderter radialer Verzeichnung κ . Für $\kappa = 0$ wird das Bild radial entzerrt. Eine etwaige Lagebestimmung für die Kamera (externe Kameraparameter) wird dadurch nicht beeinflusst.

Beachten Sie, dass change_radial_distortion_image nicht mit Zeilenkameras mit perspektivischen Objektiven funktioniert. Alternativ kann image_to_world_plane verwendet werden.

Achtung

change_radial_distortion_image kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet. Da die OpenCL-Implementierung mit einfacher Genauigkeit rechnet, kann das Ergebnis von dem der CPU Implementierung abweichen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Originalbild.
- ▷ **Region** (input_object) ..... region ⇝ *object*
  Interessierender Bildbereich in ImageRectified.
- ▷ **ImageRectified** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ausgabebild mit veränderter radialer Verzeichnung.
- ▷ **CamParamIn** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter für Image.
- ▷ **CamParamOut** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter für ImageRectified.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert change_radial_distortion_image den Wert TRUE. Das Verhalten bei leerer Eingabe (kein Eingabebild vorhanden lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Kanalebene.

Vorgänger

change_radial_distortion_cam_par, read_image, grab_image

Nachfolger

edges_image, threshold

Siehe auch

change_radial_distortion_cam_par, camera_calibration, read_cam_par, change_radial_distortion_contours_xld, change_radial_distortion_points

Modul

Calibration

change_radial_distortion_points (: : Row, Col, CamParamIn, CamParamOut : RowChanged, ColChanged)

Verändern der Radialen Verzerrung von Pixelkoordinaten.

change_radial_distortion_points ändert die radiale Verzeichnung der übergebenen Bildkoordinaten (Row, Col) entsprechend der internen Kameraparameter CamParamIn und CamParamOut. Jedes Eingabepixel (Row, Col) wird mit Hilfe von CamParamIn in die Bildebene transformiert und anschließend unter Verwendung von CamParamOut in ein anderes Bild projiziert.

Beachten Sie, dass change_radial_distortion_points nicht mit Zeilenkameras mit perspektivischen Objektiven funktioniert.

Parameter

- ▷ **Row** (input_control) ..... real-array ⇝ *real*
  Ursprüngliche Zeilen-Koordinate Pixelkoordinaten.
- ▷ **Col** (input_control) ..... real-array ⇝ *real*
  Ursprüngliche Spalten-Koordinate Pixelkoordinaten.
- ▷ **CamParamIn** (input_control) ..... campar ⇝ *real* / integer / string
  Die internen Kameraparameter der Kamera, mit der die Eingabepixelkoordinaten erstellt wurden.
- ▷ **CamParamOut** (input_control) ..... campar ⇝ *real* / integer / string
  Die internen Kameraparameter einer Kamera.
- ▷ **RowChanged** (output_control) ..... real-array ⇝ *real*
  Zeilenkoordinate der Pixelkoordinaten nach Änderung der radialen Verzerrung.
- ▷ **ColChanged** (output_control) ..... real-array ⇝ *real*
  Spaltenkoordinate der Pixelkoordinaten nach Änderung der radialen Verzerrung.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert change_radial_distortion_points den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

change_radial_distortion_cam_par, camera_calibration, read_cam_par, change_radial_distortion_contours_xld, change_radial_distortion_image

Modul

Calibration

contour_to_world_plane_xld (Contours : ContoursTrans : CameraParam, WorldPose, Scale :)

Transformieren einer XLD-Kontur in die Ebene z=0 eines Weltkoordinatensystems.

contour_to_world_plane_xld transformiert die Kontur Contours in die Ebene z=0 in einem Weltkoordinatensystem und liefert die nun in 3D-Koordinaten beschriebene Kontur in ContoursTrans zurück. Das Weltkoordinatensystem (*wcs*) wird gewählt, indem man seine 3D-Lage in Kamerakoordinaten in WorldPose übergibt. Diese 3D-Lage wird in der Form ${}^{ccs}\mathbf{P}_{wcs}$ erwartet. Dabei steht *ccs* für das Kamerakoordinatensystem,

siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". In CameraParam müssen die internen Kameraparameter übergeben werden (siehe Kalibrierung für ihre Reihenfolge und das zu Grunde liegende Kameramodell).

In vielen Fällen sind CameraParam und WorldPose das Ergebnis einer vorherigen Kamerakalibrierung mit dem Operator calibrate_cameras. Ein Beispiel findet sich weiter unten.

Die erhaltenen 3D-Koordinaten können mit dem Parameter Scale beliebig skaliert werden. Der Parameter Scale entspricht dem Verhältnis *gewünschte Einheit/ursprüngliche Einheit*. Die ursprüngliche Einheit ist durch die Koordinaten des Kalibrierkörpers gegeben. Falls dies, wie bei der Standard-Kalibrierplatte, Meter sind, ist es auch möglich, die gewünschte Einheit direkt durch *'m'*, *'cm'*, *'mm'* oder *'um'* anzugeben.

Intern berechnet der Operator zuerst den Sehstrahl vom Projektionszentrum der Kamera zum Bildpunkt in Kamerakoordinaten, unter Berücksichtigung der radialen Verzeichnung. Anschließend wird dieser Sehstrahl in das in WorldPose spezifizierte Weltkoordinatensystem transformiert. Der Schnittpunkt der Ebene z=0 mit dem Sehstrahl ergibt dann die 3D-Koordinaten der transformierten Kontur ContoursTrans.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
  Zu transformierende Eingabe-XLD-Konturen in Bildkoordinaten.
- ▷ **ContoursTrans** (output_object) ..... xld_cont(-array) ⇝ *object*
  Transformierte XLD-Konturen in Weltkoordinaten.
- ▷ **CameraParam** (input_control) ..... number-array ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **WorldPose** (input_control) ..... pose ⇝ *real* / integer
  3D-Lage des Weltkoordinatensystems in Kamerakoordinaten.
  Parameteranzahl: 7
- ▷ **Scale** (input_control) ..... number ⇝ *string* / integer / real
  Maßstab oder Dimension.
  Defaultwert: 'm'
  Wertevorschläge: Scale ∈ {'m', 'cm', 'mm', 'microns', 'um', 1.0, 0.01, 0.001, 1.0e-6, 0.0254, 0.3048, 0.9144}
  Restriktion: Scale > 0

Beispiel

```
*  Perform camera calibration (with standard calibration plate).
calibrate_cameras (CalibDataID, Error)
get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)
* Get reference pose (pose 2 of calibration object 0).
get_calib_data (CalibDataID, 'calib_obj_pose', [0,2], 'pose', WorldPose)
*  Compensate thickness of plate.
set_origin_pose(ObjInCameraPose, 0, 0, 0.0006, WorldPose)
*  Transform contours into world coordinate system (unit mm).
contour_to_world_plane_xld(Contours, ContoursTrans, CamParam, \
                           WorldPose, 'mm')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert contour_to_world_plane_xld den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_pose, hom_mat3d_to_pose, camera_calibration, hand_eye_calibration, set_origin_pose

Siehe auch

image_points_to_world_plane

Modul

Calibration

gen_image_to_world_plane_map (: Map : CameraParam, WorldPose, WidthIn, HeightIn, WidthMapped, HeightMapped, Scale, MapType :)

Berechnen der Abbildung von der Bildebene in die Ebene z=0 eines Weltkoordinatensystems.

gen_image_to_world_plane_map errechnet die Abbildung Map von der Bildebene in die Ebene z=0 (Messebene) eines Weltkoordinatensystems. Diese Abbildung kann z.B. dazu verwendet werden kann, ein Bild mit dem Operator map_image zu rektifizieren. Das rektifizierte Bild weist dann weder radiale noch perspektivische Verzerrungen auf; es entspricht einem Bild, das von einer verzerrungsfreien Kamera aufgenommen wird, die senkrecht auf die Messebene im Weltkoordinatensystem schaut. Das Weltkoordinatensystem (*wcs*) wird gewählt, indem man seine 3D-Lage in Kamerakoordinaten in WorldPose übergibt. Diese 3D-Lage wird in der Form ${}^{ccs}\mathbf{P}_{wcs}$ erwartet. Dabei steht *ccs* für das Kamerakoordinatensystem, siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision". In CameraParam müssen die internen Kameraparameter übergeben werden (siehe Kalibrierung für ihre Reihenfolge und das zu Grunde liegende Kameramodell).

In vielen Fällen sind CameraParam und WorldPose das Ergebnis einer vorherigen Kamerakalibrierung mit dem Operator calibrate_cameras. Ein Beispiel findet sich weiter unten.

Die Größe der zu transformierenden Bilder kann mit WidthIn und HeightIn angegeben werden. Die Pixel-Position der linken oberen Ecke des Ergebnisbildes ist durch den Ursprung des Weltkoordinatensystems festgelegt. Die Ausdehnung des Ergebnisbildes kann durch die Parameter WidthMapped, HeightMapped und Scale gewählt werden. Dabei sind WidthMapped und HeightMapped in Pixel anzugeben.

Der Parameter Scale kann verwendet werden um die Größe eines Pixels im transformierten Bild festzulegen. Der Parameter kann auf zweit Arten verwendet werden:

Pixel auf metrische Einheiten skalieren:
Das Bild wird so skaliert, dass ein Pixel im transformierten Bild einer metrischen Einheit entspricht, z.B. legt das Setzen von *'mm'* fest, dass ein Pixel im transformierten Bild der Fläche von 1mm $\times$ 1mm auf der Messebene entspricht. Damit dies möglich ist muss die ursprüngliche Einheit Meter sein. Dies ist der Fall wenn eine Standard-Kalibrierplatte verwendet wird.
Werteliste: *'m'*, *'cm'*, *'mm'*, *'microns'*, *'um'*.
Standardwert: *'m'*.

Skalierung manuell steuern:
Die Skalierung des Bildes wird durch das Setzen einer Zahl gesteuert, die das Verhältnis *Länge in ursprünglicher Einheit / gewünschte Anzahl Pixel* bestimmt. Wenn beispielsweise die ursprüngliche Einheit Meter ist und jedes Pixel im transformierten Bild eine Fläche von 3mm $\times$ 3mm der Messebene darstellen soll, dann berechnet man die Skalierung Scale $= 0.003/1 = 0.003$. Für Aufgaben wie formbasiertes Matching auf dem transformierten Bild ist es sinnvoll das Bild so zu skalieren, dass der Inhalt in ähnlicher Größe wie im Originalbild dargestellt wird.
Restriktion: Scale > *0*.

Die Abbildungsvorschrift wird in dem Ausgabebild Map gespeichert. Dieses hat die gleiche Größe wie die Ergebnisbilder nach der Abbildung. Der Parameter MapType gibt an, in welchem Typ die Abbildungsvorschrift gespeichert werden soll. Wurde *'nearest_neighbor'* gewählt, so besteht Map aus einem Bild mit einem Kanal, in dem für jedes Pixel des Ergebnisbildes die linearen Koordinaten desjenigen Pixels des Eingabebildes stehen, welches den transformierten Koordinaten am nächsten liegt. Wurde die bilineare Interpolation (*'bilinear'*) gewählt, so besteht Map aus einem Bild mit fünf Kanälen. Der erste Kanal enthält für jedes Pixel des Ergebnisbildes die linearen Koordinaten desjenigen Pixels des Eingabebildes, welches links oberhalb von den transformierten Koordinaten liegt. Die übrigen vier Kanäle enthalten die Gewichte der einzelnen vier Nachbarpixel der transformierten Koordinaten für die bilineare Interpolation in folgender Reihenfolge:

2	3
4	5

Der zweite Kanal z.B. enthält die Gewichte der Pixel, die links oberhalb der transformierten Koordinaten liegen. Wurde *'coord_map_sub_pix'* gewählt, besteht Map aus einem Vektorfeld, welches für jedes Pixel des Ergebnisbildes die subpixelgenauen Koordinaten im Eingabebild enthält.

Sollen mehrere Bilder unter Verwendung derselben Parameter transformiert werden, so ist die Kombination aus gen_image_to_world_plane_map und map_image wesentlich effizienter als der Operator image_to_world_plane, da die Abbildungsvorschrift nur einmal berechnet werden muss.

Wenn die erzeugte Abbildung in einem anderen Programm wiederverwendet werden soll, kann man sie mit dem Operator write_image als Mehrkanalbild abspeichern (Format: *'tiff'*).

Parameter

- ▷ **Map** (output_object) ..... (multichannel-)image ⇝ *object :* int4 / int8 / uint2 / vector_field
  Bild mit den Abbildungsdaten.
- ▷ **CameraParam** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **WorldPose** (input_control) ..... pose ⇝ *real* / integer
  3D-Lage des Weltkoordinatensystems in Kamerakoordinaten.
  Parameteranzahl: 7
- ▷ **WidthIn** (input_control) ..... extent.x ⇝ *integer*
  Breite der zu transformierenden Bilder.
  Restriktion: WidthIn >= 1
- ▷ **HeightIn** (input_control) ..... extent.y ⇝ *integer*
  Höhe der zu transformierenden Bilder.
  Restriktion: HeightIn >= 1
- ▷ **WidthMapped** (input_control) ..... extent.x ⇝ *integer*
  Breite der Ergebnisbilder in Pixel.
  Restriktion: WidthMapped >= 1
- ▷ **HeightMapped** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Ergebnisbilder in Pixel.
  Restriktion: HeightMapped >= 1
- ▷ **Scale** (input_control) ..... number ⇝ *string* / integer / real
  Maßstab oder Dimension.
  Defaultwert: 'm'
  Wertevorschläge: Scale ∈ {'m', 'cm', 'mm', 'microns', 'um', 1.0, 0.01, 0.001, 1.0e-6, 0.0254, 0.3048, 0.9144}
  Restriktion: Scale > 0
- ▷ **MapType** (input_control) ..... string ⇝ *string*
  Typ der Abbildungsdaten.
  Defaultwert: 'bilinear'
  Werteliste: MapType ∈ {'nearest_neighbor', 'bilinear', 'coord_map_sub_pix'}

Beispiel

```
* Calibrate camera.
calibrate_cameras (CalibDataID, Error)
* Obtain camera parameters.
get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)
* Example values, if no calibration data is available:
CamParam := ['area_scan_division', 0.0087, -1859, 8.65e-006, 8.6e-006, \
             362.5, 291.6, 768, 576]
* Get reference pose (pose 4 of calibration object 0).
get_calib_data (CalibDataID, 'calib_obj_pose',\
                [0,4], 'pose', Pose)
* Example values, if no calibration data is available:
Pose := [-0.11, -0.21, 2.51, 352.73, 346.73, 336.48, 0]
* Compensate thickness of plate.
set_origin_pose (Pose, -1.125, -1.0, 0, PoseNewOrigin)
* Transform the image into the world plane.
read_image (Image, 'calib/calib-3d-coord-04')
gen_image_to_world_plane_map (MapSingle, CamParam, PoseNewOrigin,\
                  CamParam[6], CamParam[7], 900, 800, 0.0025, 'bilinear')
map_image (Image, MapSingle, ImageMapped)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_image_to_world_plane_map den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_pose, hom_mat3d_to_pose, camera_calibration, hand_eye_calibration, set_origin_pose

Nachfolger

map_image

Alternativen

image_to_world_plane

Siehe auch

map_image, contour_to_world_plane_xld, image_points_to_world_plane

Modul

Calibration

gen_radial_distortion_map (: Map : CamParamIn, CamParamOut, MapType :)

Berechnen der Abbildung von Bildern aufgrund wechselnder radialer Verzeichnung.

gen_radial_distortion_map berechnet die Abbildung von Bildern aufgrund wechselnder radialer Verzeichnung gemäß den übergebenen internen Kameraparametern CamParamIn bzw. CamParamOut, die z.B. durch eine Kamerakalibrierung mit Hilfe von calibrate_cameras bestimmt wurden. CamParamIn bzw. CamParamOut stellen dabei die alten bzw. neuen internen Kameraparameter dar, die u.a. auch die alte bzw. neue radiale Verzeichnung enthalten (siehe auch Kalibrierung für die Reihenfolge der Parameter und das zu Grunde liegende Kameramodell). Zur Berechnung der Abbildung wird jedes Pixel im potentiellen Ausgabebild mittels CamParamOut in die Bildebene zurückgerechnet und dann mittels CamParamIn in eine Subpixel-Position im Eingabebild überführt. Bitte beachten Sie, dass gen_radial_distortion_map nur Flächenkameras unterstützt.

Die Abbildungsvorschrift wird in dem Ausgabebild Map gespeichert, dessen Größe durch die Kameraparameter CamParamOut festgelegt ist und somit die Größe der später mit map_image transformierten Ergebnisbilder bestimmt. Die Größe der mit map_image zu transformierenden Bilder wird durch die Kameraparameter CamParamIn bestimmt. Der Parameter MapType gibt an, in welchem Typ die Abbildungsvorschrift gespeichert werden soll. Wurde *'nearest_neighbor'* gewählt, so besteht Map aus einem Bild mit einem Kanal, in dem für jedes Pixel des Ergebnisbildes die linearen Koordinaten desjenigen Pixels des Eingabebildes stehen, welches den transformierten Koordinaten am nächsten liegt. Wurde die bilineare Interpolation (*'bilinear'*) gewählt, so besteht Map aus einem Bild mit fünf Kanälen. Der erste Kanal enthält für jedes Pixel des Ergebnisbildes die linearen Koordinaten desjenigen Pixels des Eingabebildes, welches links oberhalb von den transformierten Koordinaten liegt. Die übrigen vier Kanäle enthalten die Gewichte der einzelnen vier Nachbarpixel der transformierten Koordinaten für die bilineare Interpolation in folgender Reihenfolge:

2	3
4	5

Der zweite Kanal z.B. enthält die Gewichte der Pixel, die links oberhalb der transformierten Koordinaten liegen. Wurde *'coord_map_sub_pix'* gewählt, besteht Map aus einem Vektorfeld, welches für jedes Pixel des Ergebnisbildes die subpixelgenauen Koordinaten im Eingabebild enthält.

Wird zur Bestimmung der modifizierten Kameraparameter CamParamOut die Routine change_radial_distortion_cam_par verwendet, beschreibt die Abbildung eine Aufnahmeoptik

mit geänderter radialer Verzeichnung κ . Für $\kappa = 0$ entspricht die Abbildung somit einer radialen Entzerrung. Eine etwaige Lagebestimmung für die Kamera (externe Kameraparameter) wird dadurch nicht beeinflusst.

Sollen mehrere Bilder unter Verwendung derselben internen Kameraparameter transformiert werden, so ist die Kombination aus `gen_radial_distortion_map` und `map_image` wesentlich effizienter als der Operator `change_radial_distortion_image`, da die Abbildungsvorschrift nur einmal berechnet werden muss.

Wenn die erzeugte Abbildung in einem anderen Programm wiederverwendet werden soll, kann man sie mit dem Operator `write_image` als Mehrkanalbild abspeichern (Format: *'tiff'*).

Parameter

- ▷ **`Map`** (output_object) ..... (multichannel-)image ⇝ *object :* int4 / int8 / uint2 / vector_field
  Bild mit den Abbildungsdaten.
- ▷ **`CamParamIn`** (input_control) ..... campar ⇝ *real* / integer / string
  Alte Kameraparameter.
- ▷ **`CamParamOut`** (input_control) ..... campar ⇝ *real* / integer / string
  Neue Kameraparameter.
- ▷ **`MapType`** (input_control) ..... string ⇝ *string*
  Typ der Abbildungsdaten.
  Defaultwert: 'bilinear'
  Werteliste: `MapType` ∈ {'nearest_neighbor', 'bilinear', 'coord_map_sub_pix'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_radial_distortion_map` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`change_radial_distortion_cam_par`, `camera_calibration`, `hand_eye_calibration`

Nachfolger

`map_image`

Alternativen

`change_radial_distortion_image`

Siehe auch

`change_radial_distortion_contours_xld`

Modul

Calibration

`image_points_to_world_plane` `( : : CameraParam, WorldPose, Rows, Cols, Scale : X, Y )`

Transformieren von Bildpunkten in die Ebene z=0 eines Weltkoordinatensystems.

`image_points_to_world_plane` transformiert Bildpunkte, die in `Rows` und `Cols` gegeben sind, in die Ebene z=0 in einem Weltkoordinatensystem (*wcs*) und liefert ihre 3D-Koordinaten in `X` und `Y` zurück. Das Weltkoordinatensystem wird gewählt, indem man seine 3D-Lage in Kamerakoordinaten in `WorldPose` übergibt. Diese 3D-Lage wird in der Form ${}^{ccs}\mathbf{P}_{wcs}$ erwartet. Dabei steht *ccs* für das Kamerakoordinatensystem, siehe auch Transformationen / Posen und „`Solution Guide III-C - 3D Vision`". In `CameraParam` müssen die internen Kameraparameter übergeben werden (siehe Kalibrierung für ihre Reihenfolge und das zu Grunde liegende Kameramodell).

In vielen Fällen sind `CameraParam` und `WorldPose` das Ergebnis einer vorherigen Kamerakalibrierung mit dem Operator `calibrate_cameras`. Ein Beispiel findet sich weiter unten.

Die erhaltenen 3D-Koordinaten können mit dem Parameter `Scale` beliebig skaliert werden. Der Parameter `Scale` entspricht dem Verhältnis *gewünschte Einheit/ursprüngliche Einheit*. Die ursprüngliche Einheit ist durch

die Koordinaten des Kalibrierkörpers gegeben. Falls dies, wie bei der Standard-Kalibrierplatte, Meter sind, ist es auch möglich, die gewünschte Einheit direkt durch *'m'*, *'cm'*, *'mm'* oder *'um'* anzugeben.

Intern berechnet der Operator zuerst den Sehstrahl vom Projektionszentrum der Kamera zum Bildpunkt in Kamerakoordinaten, unter Berücksichtigung der radialen Verzeichnung. Anschließend wird dieser Sehstrahl in das in `WorldPose` spezifizierte Weltkoordinatensystem transformiert. Der Schnittpunkt der Ebene z=0 mit dem Sehstrahl ergibt dann die 3D-Koordinaten `X` und `Y`.

Es ist ratsam nur solche Bildkoordinaten `Rows` und `Cols` zu spezifizieren, die innerhalb der kalibrierten Bildgröße liegen. Das mathematische Modell funktioniert nur gut innerhalb des kalibrierten Bereichs.

Parameter

- ▷ **`CameraParam`** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **`WorldPose`** (input_control) . . . . . pose ⇝ *real* / integer
  3D-Lage des Weltkoordinatensystems in Kamerakoordinaten.
  Parameteranzahl: 7
- ▷ **`Rows`** (input_control) . . . . . coordinates.y-array ⇝ *real* / integer
  Zeilenkoordinaten der zu transformierenden Punkte.
  Defaultwert: 100.0
- ▷ **`Cols`** (input_control) . . . . . coordinates.x-array ⇝ *real* / integer
  Spaltenkoordinaten der zu transformierenden Punkte.
  Defaultwert: 100.0
- ▷ **`Scale`** (input_control) . . . . . number ⇝ *string* / integer / real
  Maßstab oder Dimension.
  Defaultwert: 'm'
  Wertevorschläge: `Scale` ∈ {'m', 'cm', 'mm', 'microns', 'um', 1.0, 0.01, 0.001, 1.0e-6, 0.0254, 0.3048, 0.9144}
  Restriktion: Scale > 0
- ▷ **`X`** (output_control) . . . . . coordinates.x-array ⇝ *real*
  X-Koordinaten der Punkte im Weltkoordinatensystem.
- ▷ **`Y`** (output_control) . . . . . coordinates.y-array ⇝ *real*
  Y-Koordinaten der Punkte im Weltkoordinatensystem.

Beispiel

```
*  Perform camera calibration (with standard calibration plate).
calibrate_cameras (CalibDataID, Error)
get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)
* Get reference pose (pose 2 of calibration object 0).
get_calib_data (CalibDataID, 'calib_obj_pose',\
                [0,2], 'pose', WorldPose)
*  Compensate thickness of plate.
set_origin_pose(ObjInCameraPose, 0, 0, 0.0006, WorldPose)
*  Transform image points into world coordinate system (unit mm).
image_points_to_world_plane(CamParam, WorldPose, PointRows, PointColumns, \
                            'mm', PointXCoord, PointYCoord)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `image_points_to_world_plane` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_pose`, `hom_mat3d_to_pose`, `camera_calibration`, `hand_eye_calibration`, `set_origin_pose`

Siehe auch

contour_to_world_plane_xld, project_3d_point

Modul

Calibration

image_to_world_plane (Image : ImageWorld : CameraParam, WorldPose, Width, Height, Scale, Interpolation :)

Rektifizieren eines Bildes durch Transformation in die Ebene z=0 eines Weltkoordinatensystems.

image_to_world_plane rektifiziert ein Bild Image durch Transformation in die Ebene z=0 (Messebene) eines Weltkoordinatensystems. Das resultierende Bild ImageWorld weist weder radiale noch perspektivische Verzerrungen auf; es entspricht einem Bild, das von einer verzerrungsfreien Kamera aufgenommen wird, die senkrecht auf die Messebene im Weltkoordinatensystem schaut. Das Weltkoordinatensystem wird gewählt, indem man seine 3D-Lage in Kamerakoordinaten in WorldPose übergibt. Diese 3D-Lage wird in der Form ${}^{ccs}\mathbf{P}_{wcs}$ erwartet. Dabei steht *ccs* für das Kamerakoordinatensystem und *wcs* für das Weltkoordinatensystem (siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision"). In CameraParam müssen die internen Kameraparameter übergeben werden (siehe Kalibrierung für ihre Reihenfolge und das zu Grunde liegende Kameramodell).

In vielen Fällen sind CameraParam und WorldPose das Ergebnis einer vorherigen Kamerakalibrierung mit dem Operator calibrate_cameras. Ein Beispiel findet sich weiter unten.

Die Pixel-Position der linken oberen Ecke des Ergebnisbildes ImageWorld ist durch den Ursprung des Weltkoordinatensystems festgelegt. Die Ausdehnung des Ergebnisbildes ImageWorld kann durch die Parameter Width, Height und Scale gewählt werden. Dabei sind Width und Height in Pixel anzugeben.

Der Parameter Scale kann verwendet werden um die Größe eines Pixels im transformierten Bild festzulegen. Der Parameter kann auf zweit Arten verwendet werden:

Pixel auf metrische Einheiten skalieren:

Das Bild wird so skaliert, dass ein Pixel im transformierten Bild einer metrischen Einheit entspricht, z.B. legt das Setzen von *'mm'* fest, dass ein Pixel im transformierten Bild der Fläche von 1mm $\times$ 1mm auf der Messebene entspricht. Damit dies möglich ist muss die ursprüngliche Einheit Meter sein. Dies ist der Fall wenn eine Standard-Kalibrierplatte verwendet wird.

Werteliste: *'m'*, *'cm'*, *'mm'*, *'microns'*, *'um'*.

Standardwert: *'m'*.

Skalierung manuell steuern:

Die Skalierung des Bildes wird durch das Setzen einer Zahl gesteuert, die das Verhältnis *Länge in ursprünglicher Einheit / gewünschte Anzahl Pixel* bestimmt. Wenn beispielsweise die ursprüngliche Einheit Meter ist und jedes Pixel im transformierten Bild eine Fläche von 3mm $\times$ 3mm der Messebene darstellen soll, dann berechnet man die Skalierung Scale $= 0.003/1 = 0.003$. Für Aufgaben wie formbasiertes Matching auf dem transformierten Bild ist es sinnvoll das Bild so zu skalieren, dass der Inhalt in ähnlicher Größe wie im Originalbild dargestellt wird.

Restriktion: Scale *> 0*.

Der Parameter Interpolation gibt an, ob zwischen den Pixeln des Eingabebildes bilinear interpoliert (*'bilinear'*) oder lediglich der Grauwert des am nächsten liegenden Pixels (*'nearest_neighbor'*) verwendet werden soll.

Sollen mehrere Bilder unter Verwendung derselben Parameter transformiert werden, so ist die Kombination aus gen_image_to_world_plane_map und map_image wesentlich effizienter als der Operator image_to_world_plane, da die Abbildungsvorschrift nur einmal berechnet werden muss.

Achtung

image_to_world_plane kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet. Das Ergebnis kann dabei geringfügig von dem auf der CPU berechneten abweichen.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **ImageWorld** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Transformiertes Bild.
- ▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **WorldPose** (input_control) . . . . . pose ⇝ *real* / integer
  3D-Lage des Weltkoordinatensystems in Kamerakoordinaten.
  Parameteranzahl: 7
- ▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
  Breite des Ergebnisbildes in Pixel.
  Restriktion: Width >= 1
- ▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
  Höhe des Ergebnisbildes in Pixel.
  Restriktion: Height >= 1
- ▷ **Scale** (input_control) . . . . . number ⇝ *string* / integer / real
  Maßstab oder Dimension.
  Defaultwert: 'm'
  Wertevorschläge: `Scale` ∈ {'m', 'cm', 'mm', 'microns', 'um', 1.0, 0.01, 0.001, 1.0e-6, 0.0254, 0.3048, 0.9144}
  Restriktion: Scale > 0
- ▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
  Art der Interpolation.
  Defaultwert: 'bilinear'
  Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear'}

Beispiel

```
* Calibrate camera.
calibrate_cameras (CalibDataID, Error)
* Obtain camera parameters.
get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)
* Example values, if no calibration data is available:
CamParam := ['area_scan_division', 0.0087, -1859, 8.65e-006, 8.6e-006, \
             362.5, 291.6, 768, 576]
* Get reference pose (pose 4 of calibration object 0).
get_calib_data (CalibDataID, 'calib_obj_pose',\
                [0,4], 'pose', Pose)
* Example values, if no calibration data is available:
Pose := [-0.11, -0.21, 2.51, 352.73, 346.73, 336.48, 0]
* Compensate thickness of plate.
set_origin_pose (Pose, -1.125, -1.0, 0, PoseNewOrigin)
* Transform the image into the world plane.
read_image (Image, 'calib/calib-3d-coord-04')
image_to_world_plane (Image, ImageWorld, CamParam, PoseNewOrigin,\
                      900, 800, 0.0025, 'bilinear')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `image_to_world_plane` den Wert TRUE.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

create_pose, hom_mat3d_to_pose, camera_calibration, hand_eye_calibration, set_origin_pose

Alternativen

gen_image_to_world_plane_map, map_image

Siehe auch

contour_to_world_plane_xld, image_points_to_world_plane

Modul

Calibration

14.10 Selbst-Kalibrierung

```
radial_distortion_self_calibration (
    Contours : SelectedContours : Width, Height, InlierThreshold,
    RandSeed, DistortionModel, DistortionCenter,
    PrincipalPointVar : CameraParam )
```

Kalibrieren der radialen Verzeichnung.

radial_distortion_self_calibration berechnet die Verzeichnungsparameter sowie das Verzeichnungszentrum eines Objektivs basierend auf den in Contours übergebenen XLD-Konturen.

Die Verzeichnungswerte werden in CameraParam zurückgegeben. Da keine weiteren Parameter geschätzt werden - insbesondere nicht die Brennweite oder der Vergrößerungsmaßstab - wird ein telezentrisches Kameramodell mit Vergrößerungsmaßstab 1 und Skalierungsfaktor 1 für S_x und S_y zurückgegeben. Siehe Kalibrierung für mehr Informationen zu den verschiedenen Kameramodellen.

Anwendung

Mit dem Ergebnis von radial_distortion_self_calibration kann man die Objektivverzeichnungen in Bildern korrigieren. Hierzu muss man den Parameter CameraParam, der die Verzeichnungsparameter enthält, an die Operatoren change_radial_distortion_cam_par und change_radial_distortion_image weiterreichen.

Grundprinzip

Die Bestimmung der Verzeichnung basiert auf der Annahme, dass in der aufgenommen Szene eine signifikante Anzahl von geraden Kanten oder Linien vorhanden ist. Durch die Linsenverzeichnung werden diese Geraden auf gekrümmte Linien abgebildet. Es werden nun geeignete Parameter gesucht, die diese gekrümmten Linien wieder auf Geraden abbilden und somit die Verzeichnung kompensieren.

Eingabekonturen extrahieren

Um geeignete Eingabekonturen Contours zu erhalten, kann man z.B. edges_sub_pix oder lines_gauss verwenden. Die Konturen sollten im Bild gleichverteilt und möglichst in der Nähe des Bildrands liegen, da sich dort die Verzeichnung am stärksten auswirkt und damit die Kalibrierung am stabilsten wird. Zur Verbesserung von Laufzeit oder Robustheit können Sie versuchen, z.B. mit segment_contours_xld, union_collinear_contours_xld, union_cocircular_contours_xld oder select_shape_xld möglichst lange gerade oder kreisförmige Segmente zu erhalten. Falls in einem einzelnen Bild nicht ausreichend viele gerade Konturen in der Szene vorhanden sind, können Sie auch die Konturen aus mehreren Bildern übergeben (concat_obj).

Parameter für die Konturauswahl setzen

Der Operator bestimmt automatisch über ein robustes RANSAC-Verfahren jene Konturen aus Contours, die das Abbild von Raumgeraden sind. Die Konturen, die diese Forderung nicht erfüllen und somit für den Kalibrierprozess ungeeignet sind, werden als Ausreißer bezeichnet. Der Operator geht von einer maximalen Ausreißerquote von 50 Prozent aus. Eine Kontur wird als Ausreißer klassifiziert, falls die mittlere Abweichung der Kontur von einer Geraden nach der Verzeichnungskorrektur den maximalen Schwellenwert T überschreitet:

$$\frac{1}{m}\sum_{j=1}^{m}|d_j| > \texttt{InlierThreshold}\cdot\frac{m}{100} = T$$

Der Wert InlierThreshold beschreibt die mittlere Abweichung der Kontur von einer Geraden in Pixeln bei einer Kontur mit 100 Konturpunkten. Der eigentliche Schwellenwert T entsteht durch die Skalierung von InlierThreshold mit der Referenzlänge (100) und der Konturpunktanzahl m, um bei geometrisch ähnlichen Konturen die gleiche Kontur-Klassifizierung zu erreichen. Die typischen Werte für InlierThreshold liegen zwischen 0.05 und 0.5. Je höher der Wert, desto mehr Abweichungen werden toleriert. Wird für InlierThreshold der Wert 0 eingegeben, werden sämtliche Konturen Contours zur Kalibrierung verwendet. Die RANSAC-Konturauswahl wird dadurch unterdrückt, so dass eine manuelle Auswahl der Konturen möglich ist. Dies ist beispielsweise dann hilfreich, wenn die Ausreißerquote über 50 Prozent liegt.

Mit dem Parameter RandSeed kann das randomisierte Verhalten des RANSAC-Verfahrens beeinflusst werden, um reproduzierbare Ergebnisse zu erhalten. Wird RandSeed auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit RandSeed initialisiert wird.

radial_distortion_self_calibration gibt die Konturen, die zur Kalibrierung verwendet wurden in SelectedContours die zurück.

Verzeichnungsmodell auswählen

Das zur Kalibrierung verwendete Verzeichnungsmodell kann über DistortionModel gewählt werden. Beim Divisionsmodell (DistortionModel = *'division'*) werden die Verzeichnungen über den Verzeichnungsparameter κ beschrieben. Beim polynomiellen Modell (DistortionModel = *'polynomial'*) werden die Verzeichnungen über die radialen Verzeichnungsparameter K_1, K_2, K_3 und die tangentialen Verzeichnungsparameter P_1, P_2 beschrieben. Siehe Kalibrierung für mehr Informationen zu den verschiedenen Kameramodellen.

Parameter zur Bestimmung des Verzeichnungszentrums einstellen

Als Startpunkt für die Bestimmung des Verzeichnungszentrums $c = (c_x, c_y)$ wird die Bildmitte verwendet; die Bildgröße wird über Width und Height definiert.

Zur Bestimmung der Verzeichnungsparameter (κ, c_x, c_y) beziehungsweise $(K_1, K_2, K_3, P_1, P_2, c_x, c_y)$ stehen die Methoden *'variable'*, *'adaptive'* und *'fixed'* zur Verfügung, die über den Parameter DistortionCenter ausgewählt werden:

'variable' Bei der Vorauswahl *'variable'* wird von Beginn an mit variablen Verzeichnungszentrum c kalibriert. Hierbei sollten viele gleichmäßig verteilte Konturen in der Nähe des Bildrandes vorliegen oder die Verzeichnung möglichst stark sein, da ansonsten das Verzeichnungszentrum nicht bestimmt werden kann und dies zur Instabilität führt.

'adaptive' Bei der Variante *'adaptive'* wird zunächst das Verzeichnungszentrum c fest in der Bildmitte angenommen und die Ausreißer mit dem Schwellenwert InlierThreshold eliminiert. In einem letzten Schritt wird dann eine Kalibrierung mit allen Verzeichnungsparametern (κ, c_x, c_y) beziehungsweise $(K_1, K_2, K_3, P_1, P_2, c_x, c_y)$ durchgeführt, die dann angenommen wird, falls $c = (c_x, c_y)$ als eine stabile Lösung in der Nähe der Bildmitte gefunden wurde. Andernfalls wird c in der Bildmitte angenommen. Diese Methode sollte dann verwendet werden, wenn das Verzeichnungszentrum außerhalb der Bildmitte gesucht werden soll, jedoch wenige Konturen zur Verfügung stehen oder die Lage der Konturen ungünstig ist (Bsp.: Konturen verlaufen in gleiche Richtung oder liegen im selben Bildbereich).

'fixed' Bei *'fixed'* wird schließlich das Verzeichnungszentrum als konstant in der Bildmitte angenommen und nur κ beziehungsweise $(K_1, K_2, K_3, P_1, P_2)$ bestimmt. Diese Methode sollte bei sehr schwachen Verzeichnungen und wenigen Konturen in schlechter Lage angewendet werden.

Über den Parameter PrincipalPointVar kann bei den Methoden *'adaptive'* und *'variable'* schließlich die Abweichung von c von der Bildmitte gesteuert werden. Falls die Abweichung gesteuert werden soll, muss PrincipalPointVar zwischen 1 und 100 liegen. Dabei gilt: Je höher der Wert, umso mehr darf c von der Bildmitte abweichen. Liegt der Wert bei 0, findet keine Steuerung statt, d.h. der Hauptpunkt wird alleine durch die Konturen bestimmt. Generell sollte die Steuerung über PrincipalPointVar bei schwachen Verzeichnungen und ähnlich orientierten Konturen verwendet werden, da sonst unter Umständen keine stabile Lösung gefunden wird.

Laufzeit

Die Laufzeit von radial_distortion_self_calibration ist für DistortionCenter = *'variable'* und PrincipalPointVar = *0* am geringsten. Die Laufzeit für DistortionCenter = *'variable'* und PrincipalPointVar > *0* steigt mit kleineren Werten von PrincipalPointVar signifikant an. Die Laufzeiten für DistortionCenter = *'adaptive'* und DistortionCenter = *'fixed'* sind auch signifikant höher als bei DistortionCenter = *'variable'* und PrincipalPointVar = *0*.

Achtung

Da das polynomielle Modell (`DistortionModel` = *'polynomial'*) mehr Parameter enthält als das Divisionsmodell (`DistortionModel` = *'division'*), kann die Kalibrierung über das polynomielle Modell eine etwas geringere Stabilität aufweisen als die Kalibrierung über das Divisionsmodell. Dies macht sich besonders bei der Bestimmung der tangentialen Verzeichnungskomponenten P_1, P_2 bemerkbar. Die Stabilität kann verbessert werden, indem man Konturen aus mehreren Bildern verwendet. Zusätzliche Stabilität erhält man wie oben bereits erwähnt für `DistortionCenter` = *'fixed'*, `DistortionCenter` = *'adaptive'* oder `PrincipalPointVar` > *0*.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont-array ⇝ *object*
  Konturen, die für die Kalibrierung zur Verfügung stehen.
- ▷ **`SelectedContours`** (output_object) ..... xld_cont-array ⇝ *object*
  Konturen, die zur Kalibrierung verwendet wurden
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Breite der Bilder, aus denen die Konturen extrahiert wurden.
  Defaultwert: 640
  Wertevorschläge: `Width` ∈ {640, 768}
  Restriktion: Width > 0
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Bilder, aus denen die Konturen extrahiert wurden.
  Defaultwert: 480
  Wertevorschläge: `Height` ∈ {480, 576}
  Restriktion: Height > 0
- ▷ **`InlierThreshold`** (input_control) ..... real ⇝ *real*
  Schwellenwert zur Klassifizierung von Ausreißern.
  Defaultwert: 0.05
  Wertevorschläge: `InlierThreshold` ∈ {0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1}
  Restriktion: InlierThreshold >= 0
- ▷ **`RandSeed`** (input_control) ..... integer ⇝ *integer*
  Initialisierungswert für den Zufallszahlengenerator.
  Defaultwert: 42
- ▷ **`DistortionModel`** (input_control) ..... string ⇝ *string*
  Legt das Verzeichnungsmodell fest.
  Defaultwert: 'division'
  Werteliste: `DistortionModel` ∈ {'division', 'polynomial'}
- ▷ **`DistortionCenter`** (input_control) ..... string ⇝ *string*
  Legt fest, wie das Verzeichnungszentrum bestimmt wird.
  Defaultwert: 'variable'
  Werteliste: `DistortionCenter` ∈ {'fixed', 'adaptive', 'variable'}
- ▷ **`PrincipalPointVar`** (input_control) ..... real ⇝ *real*
  Steuert die Abweichung des Verzeichnungszentrums von der Bildmitte; größere Werte erlauben größere Abweichungen; 0 schaltet den Strafterm ab.
  Defaultwert: 0.0
  Wertevorschläge: `PrincipalPointVar` ∈ {0.0, 5.0, 10.0, 20.0, 50.0, 100.0}
  Restriktion: PrincipalPointVar >= 0.0 && PrincipalPointVar <= 100.0
- ▷ **`CameraParam`** (output_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.

Beispiel

```
* Assume that GrayImage is one image in gray values with a
* resolution of 640 x 480 and a suitable number of contours.  Then
* the following example performs the calibration using these
* contours and corrects the image with the estimated distortion
* parameters.
edges_sub_pix (GrayImage, Edges, 'canny', 1.0, 20, 40)
segment_contours_xld (Edges, ContoursSplit, 'lines_circles', 5, 8, 4)
radial_distortion_self_calibration (ContoursSplit, SelectedContours, \
```

```
                                        640, 480, 0.08, 42, 'division', \
                                        'variable', 0, CameraParam)
get_domain (GrayImage, Domain)
change_radial_distortion_cam_par ('fullsize', CameraParam, 0, CamParamOut)
change_radial_distortion_image (GrayImage, Domain, ImageRectified, \
                                CameraParam, CamParamOut)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `radial_distortion_self_calibration` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`edges_sub_pix`, `segment_contours_xld`

Nachfolger

`change_radial_distortion_cam_par`, `change_radial_distortion_image`

Siehe auch

`camera_calibration`

Literatur

T. Thormälen, H. Broszio: „Automatic line-based estimation of radial lens distortion“; in: Integrated Computer-Aided Engineering; vol. 12; pp. 177-190; 2005.

Modul

Calibration

radiometric_self_calibration (Images : : ExposureRatios, Features, FunctionType, Smoothness, PolynomialDegree : InverseResponse)

Radiometrische Selbstkalibrierung einer Kamera.

`radiometric_self_calibration` führt eine radiometrische Selbstkalibrierung einer Kamera durch. Dazu müssen in `Images` mindestens zwei Bilder übergeben werden, die denselben Bildinhalt zeigen. Alle in `Images` übergebenen Bilder müssen mit unterschiedlichen Belichtungen aufgenommen worden sein. Typischerweise werden die unterschiedlichen Belichtungen durch Veränderung der Verschlusszeiten (Belichtungszeiten) an der Kamera erzeugt. Eine Änderung der Belichtung durch Verstellen der Blende ist nicht zu empfehlen, da hier die Belichtungsverhältnisse nicht genau genug bestimmt werden können. Das Verhältnis der Belichtungen aufeinanderfolgender Bilder wird in `ExposureRatios` übergeben. Ein Wert von *0.5* bedeutet z.B., dass das zweite Bild des Bildpaares mit der halben Belichtung des ersten Bildes aufgenommen wurde. Das Verhältnis der Belichtungen kann leicht aus den Verschlusszeiten berechnet werden, da die Belichtung proportional zur Verschlusszeit ist. Das Verhältnis der Belichtungen muss größer als 0 und kleiner als 1 sein. Das bedeutet, dass die Bilder nach abnehmender Belichtung sortiert sein müssen. `ExposureRatios` muss ein Element weniger enthalten, als die Anzahl der in `Images` übergebenen Bilder. Falls alle Verhältnisse der Belichtungen identisch sind, kann zur Vereinfachung auch ein einziger Wert in `ExposureRatios` übergeben werden.

Wie oben beschrieben, müssen die in `Images` übergebenen Bilder denselben Bildinhalt zeigen. Dazu ist es normalerweise notwendig, dass sich weder die Kamera noch die im Bild enthaltenen Objekte bewegen. Falls sich die Kamera um das optische Zentrum gedreht hat, sollten die Bilder mit `proj_match_points_ransac` und `projective_trans_image` auf eines der Bilder als Referenzbild ausgerichtet werden. Falls die zur radiometrischen Kalibrierung verwendeten Merkmale aus dem 2D-Grauwerthistogramm von aufeinanderfolgenden Bildpaaren aus `Images` bestimmt werden (`Features` = *'2d_histogram'*), ist es unerlässlich, dass die Bilder aufeinander ausgerichtet sind und dass sich die Objekte in den Bildern nicht bewegen. Im Modus `Features` = *'1d_histograms'* werden die zur radiometrischen Kalibrierung verwendeten Merkmale aus den 1D-Grauwerthistogrammen der Bildpaare berechnet. In diesem Modus kann eine Kalibrierung theoretisch auch

dann erfolgen, falls sich die 1D-Histogramme der Bilder durch eine Bewegung von Objekten im Bild nicht ändert. Dies kann z.B. dann der Fall sein, falls sich ein Objekt vor einem gleichmäßig texturierten Hintergrund bewegt. Aufgrund der höheren Genauigkeit ist aber der Modus Features = '2d_histogram' vorzuziehen. Der Modus Features = '1d_histograms' sollte nur verwendet werden, falls es unmöglich ist, den Kameraaufbau so zu konstruieren, dass sich weder die Kamera noch die Bildobjekte bewegen.

Des Weiteren ist darauf zu achten, dass der Grauwertbereich lückenlos überdeckt wird. Hierzu sollte ein geeigneter Bildinhalt gewählt werden. Ob Lücken im Grauwertbereich vorliegen, kann leicht anhand der 1D-Grauwerthistogramme der Bilder und der 2D-Grauwerthistogramme aufeinanderfolgender Bilder überprüft werden. In den 1D-Grauwerthistogrammen (siehe gray_histo_abs) sollten zwischen dem minimalen und maximalen Grauwert keine Bereiche mit einer Häufigkeit von 0 oder einer sehr kleinen Häufigkeit vorkommen. In den 2D-Grauwerthistogrammen (siehe histo_2dim) sollte nach der Segmentierung mit einem unteren Schwellenwert von 1 eine zusammenhängende Region in Form eines „Schlauches“ entstehen. Falls mehrere Zusammenhangskomponenten entstehen, sollte ein geeigneterer Bildinhalt gewählt werden. Falls der Bildinhalt so gewählt werden kann, dass der Grauwertbereich des Bildes (z.B. 0-255 bei Byte-Bildern) mit zwei Bildern mit unterschiedlichen Belichtungen ausgeschöpft werden kann, und keine Lücken in den Histogrammen vorhanden sind, sind diese zwei Bilder zur Kalibrierung ausreichend. Dies ist typischerweise jedoch nicht der Fall, so dass mit mehreren Bildern der Grauwertbereich überdeckt werden muss. Dazu sind, wie oben beschrieben, mehrere Bilder mit unterschiedlichen Belichtungen aufzunehmen, um den Grauwertbereich möglichst vollständig zu überdecken. Dabei sollte das erste Bild im Normalfall so belichtet werden, dass der maximale Grauwert entweder knapp unter der Sättigungsgrenze der Kamera liegt, oder dass das Bild deutlich übersteuert ist. Falls das erste Bild übersteuert aufgenommen wird, ist eine deutliche Übersteuerung notwendig, damit radiometric_self_calibration die übersteuerten Bereiche sicher erkennen kann. Falls die Kamera ein ungewöhnliches Sättigungsverhalten zeigt (z.B. eine Sättigungsgrenze, die signifikant unter dem maximalen Grauwert liegt), sollten in dem übersteuerten Bild die übersteuerten Bereiche per Hand mit reduce_domain ausmaskiert werden.

radiometric_self_calibration liefert die inverse Grauwert-Antwortfunktion der Kamera in InverseResponse zurück. Die inverse Antwortfunktion kann dazu verwendet werden, ein Bild mit linearer Grauwert-Antwortfunktion zu erzeugen, indem InverseResponse als LUT in lut_trans verwendet wird. Der Parameter FunctionType gibt an, welches Funktionsmodell für die Antwortfunktion verwendet werden soll. Für FunctionType = 'discrete' wird die Antwortfunktion durch eine diskrete Funktion mit der relevanten Anzahl von Grauwerten beschrieben (256 bei Byte-Bildern). Für FunctionType = 'polynomial' wird die Antwortfunktion durch ein Polynom vom Grad PolynomialDegree beschrieben. Die Berechnung der Antwortfunktion ist für FunctionType = 'discrete' langsamer. Da aber in den Bereichen, an denen keine Grauwertinformation vorliegt, ein Polynom trotz der unten beschriebenen Glattheitsbedingungen Oszillationen aufweisen kann, ist die diskrete Funktion dem Polynom im Normalfall vorzuziehen.

Die inverse Grauwert-Antwortfunktion wird als Tupel von ganzen Zahlen für FunctionType = 'discrete' und FunctionType = 'polynomial' zurückgeliefert. In manchen Anwendungen kann es wünschenswert sein, die inverse Grauwert-Antwortfunktion als Gleitkommazahlen zurückzuliefern, um die numerischen Fehler, die durch Runden entstehen, zu vermeiden. Falls z.B. die inverse Grauwert-Antwortfunktion invertiert werden muss, um die tatsächliche Antwortfunktion der Kamera zu berechnen, geht Information durch die Rundung verloren. In diesen Anwendungen kann FunctionType auf *'discrete_real'* oder *'polynomial_real'* gesetzt werden, was dazu führt, dass die inverse Grauwert-Antwortfunktion als ein Tupel von Gleitkommazahlen zurückgeliefert wird.

Der Parameter Smoothness definiert (zusätzlich zu den Bedingungen, die aus den Bildern über die Antwortfunktion der Kamera hergeleitet werden können) Bedingungen für die Glattheit der Antwortfunktion. Falls der Grauwertbereich wie oben beschrieben vollständig und lückenlos überdeckt werden kann und falls das Kamerarauschen gering ist, sollte der Standardwert von *1* nicht verändert werden. Ansonsten kann mit Werten > 1 eine stärkere Glättung und mit Werten < 1 eine schwächere Glättung der Antwortfunktion erreicht werden. Die Glättung ist insbesondere in Bereichen wichtig, in denen keine Grauwertinformation aus den Bildern hergeleitet werden kann, also insbesondere in Lücken im Histogramm und für Grauwerte, die kleiner sind als der minimale bzw. größer als der maximale Grauwert aller Bilder. Hier führen die Glattheitsbedingungen zu einer Interpolation bzw. Extrapolation der Antwortfunktion. Durch die Art der intern berechneten Bedingungsgleichungen bevorzugt FunctionType = 'discrete' eine Exponentialfunktion in undefinierten Grauwertbereichen, während FunctionType = 'polynomial' eine Gerade bevorzugt. Beachten Sie bitte, dass die Interpolation und Extrapolation in jedem Fall unsicherer ist, als den Grauwertbereich vollständig und lückenlos zu überdecken. Deswegen sollte in jedem Fall zuerst versucht werden, die Bilder optimal aufzunehmen, bevor anhand der Glattheitsbedingungen ein Auffüllen der Lücken vorgenommen wird. In jedem Fall sollte die Antwortfunktion nach dem Aufruf von radiometric_self_calibration auf Plausibilität überprüft werden. Insbesondere sollte überprüft werden, ob InverseResponse monoton ist. Falls dies nicht der Fall ist, sollte ein geeigneterer Bildinhalt verwendet werden, damit nicht interpoliert werden muss, oder Smoothness auf einen größeren Wert gesetzt werden.

Für FunctionType = *'polynomial'* kann es auch notwendig sein, PolynomialDegree zu verändern. Falls trotz dieser Änderungen immer noch ein wenig plausibles Ergebnis geliefert wird, sollte das Sättigungsverhalten der Kamera überprüft werden, z.B. anhand des 2D-Grauwerthistogramms, und die gesättigten Bereiche wie oben beschrieben per Hand ausmaskiert werden.

Bei der Bestimmung der inversen Grauwert-Antwortfunktion der Kamera kann die absolute Energie, die auf die Kamera fällt, nicht bestimmt werden. Das heißt, dass InverseResponse nur bis auf einen Skalierungsfaktor bestimmt werden kann. Daher wird als Nebenbedingung verwendet, dass der maximale Grauwert, der auftreten kann, am maximalen Eingabegrauwert angenommen werden soll, also z.B. für Byte-Bilder InverseResponse$[255] = 255$. Diese Nebenbedingung liefert im Normalfall ein intuitives Ergebnis. Falls jedoch ein mehrkanaliges Bild (typischerweise ein RGB-Bild) radiometrisch kalibriert werden soll (hierzu muss jeder Kanal einzeln kalibriert werden), kann es vorkommen, dass durch die obige Nebenbedingung ein unterschiedlicher Skalierungsfaktor für jeden Kanal berechnet wird. Das kann dazu führen, dass Grautöne nach der Korrektur nicht mehr grau erscheinen. In diesem Fall ist ein manueller Weißabgleich durchzuführen, indem ein homogenes graues Gebiet im Originalbild identifiziert wird, und aus den Grauwerten der korrigierten Kanäle Skalierungsfaktoren für zwei der drei Antwortkurven (bzw. allgemein für $n-1$ der n Kanäle) hergeleitet werden. Dabei sollte die nicht zu ändernde Antwortkurve so gewählt werden, dass alle Skalierungsfaktoren < 1 sind. Mit den so berechneten Skalierungsfaktoren sollten neue Antwortfunktionen durch Multiplikation aller Werte einer Antwortfunktion mit dem jeweiligen Skalierungsfaktor berechnet werden.

Parameter

- ▷ **Images** (input_object) ..... singlechannelimage-array ⇝ *object :* byte / uint2
  Eingabebilder.
- ▷ **ExposureRatios** (input_control) ..... real(-array) ⇝ *real*
  Verhältnis der Belichtungsenergien aufeinanderfolgender Bildpaare.
  Defaultwert: 0.5
  Wertevorschläge: ExposureRatios ∈ {0.25, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8}
  Restriktion: ExposureRatios > 0 && ExposureRatios < 1
- ▷ **Features** (input_control) ..... string ⇝ *string*
  Merkmale, die zur Berechnung der inversen Antwortfunktion der Kamera verwendet werden.
  Defaultwert: '2d_histogram'
  Werteliste: Features ∈ {'2d_histogram', '1d_histograms'}
- ▷ **FunctionType** (input_control) ..... string ⇝ *string*
  Typ der inversen Antwortfunktion der Kamera.
  Defaultwert: 'discrete'
  Werteliste: FunctionType ∈ {'discrete', 'polynomial', 'discrete_real', 'polynomial_real'}
- ▷ **Smoothness** (input_control) ..... real ⇝ *real*
  Glattheit der inversen Antwortfunktion der Kamera.
  Defaultwert: 1.0
  Wertevorschläge: Smoothness ∈ {0.3, 0.5, 0.7, 0.8, 1.0, 1.2, 1.5, 2.0, 3.0}
  Restriktion: Smoothness > 0
- ▷ **PolynomialDegree** (input_control) ..... integer ⇝ *integer*
  Grad des Polynoms falls FunctionType = *'polynomial'*.
  Defaultwert: 5
  Wertevorschläge: PolynomialDegree ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Restriktion: PolynomialDegree >= 1 && PolynomialDegree <= 20
- ▷ **InverseResponse** (output_control) ..... number-array ⇝ *integer* / real
  Inverse Antwortfunktion der Kamera.

Beispiel

```
open_framegrabber ('1394IIDC', 1, 1, 0, 0, 0, 0, 'default', -1, \
                   'default', -1, 'default', 'default', 'default', \
                   -1, -1, AcqHandle)
* Define appropriate shutter times.
Shutters := [1000,750,500,250,125]
Num := |Shutters|
* Grab and accumulate images with the different exposures.  In this
* loop, it must be ensured that the scene remains static.
```

```
gen_empty_obj (Images)
for I := 0 to Num-1 by 1
    set_framegrabber_param (AcqHandle, 'shutter', Shutters[I])
    grab_image (Image, AcqHandle)
    concat_obj (Images, Image, Images)
endfor
* Compute the exposure ratios from the shutter times.
ExposureRatios := real(Shutters[1:Num-1])/real(Shutters[0:Num-2])
radiometric_self_calibration (Images, ExposureRatios, '2d_histogram', \
                              'discrete', 1, 5, InverseResponse)
* Note that if the frame grabber supports hardware LUTs, we could
* also call set_framegrabber_lut here instead of lut_trans below.
* This would be more efficient.
while (1)
    grab_image_async (Image, AcqHandle, -1)
    lut_trans (Image, ImageLinear, InverseResponse)
    * Process radiometrically correct image.
    * [...]
endwhile
close_framegrabber (AcqHandle)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `radiometric_self_calibration` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_image`, `grab_image`, `grab_image_async`, `set_framegrabber_param`, `concat_obj`, `proj_match_points_ransac`, `proj_match_points_ransac_guided`, `projective_trans_image`

Nachfolger

`lut_trans`

Siehe auch

`histo_2dim`, `gray_histo`, `gray_histo_abs`, `reduce_domain`

Modul

Calibration

stationary_camera_self_calibration (: : NumImages, ImageWidth, ImageHeight, ReferenceImage, MappingSource, MappingDest, HomMatrices2D, Rows1, Cols1, Rows2, Cols2, NumCorrespondences, EstimationMethod, CameraModel, FixedCameraParams : CameraMatrices, Kappa, RotationMatrices, X, Y, Z, Error)

Selbstkalibrierung einer stationären projektiven Kamera.

`stationary_camera_self_calibration` führt eine Selbstkalibrierung einer stationären projektiven Kamera durch. Stationär bedeutet hierbei, dass sich die Kamera nur um das optische Zentrum drehen und dabei zoomen darf. Das optische Zentrum darf also nicht verschoben werden. Projektive Kamera bedeutet hierbei, dass als Kameramodell eine Lochkamera verwendet wird, deren Abbildung durch eine projektive 3D-2D-Abbildung beschrieben werden kann. Insbesondere können radiale Verzeichnungen nur modelliert werden, wenn die Brennweite konstant bleibt. Falls die Kamera signifikante radiale Verzeichnungen aufweist, sollten diese mit `change_radial_distortion_image` zumindest näherungsweise beseitigt werden.

Das zugrundeliegende Kameramodell kann wie folgt beschrieben werden:

$$\mathbf{x} = \mathtt{P}\mathbf{X} \ .$$

Hierbei ist $\mathbf{x}$ ein homogener 2D-Vektor, $\mathbf{X}$ ein homogener 3D-Vektor und $\mathtt{P}$ eine homogene 3×4 Projektionsmatrix. Die Projektionsmatrix $\mathtt{P}$ lässt sich wie folgt zerlegen:

$$\mathtt{P} = \mathtt{K}(\mathtt{R}|\mathbf{t}) \ .$$

Hierbei ist $\mathtt{R}$ eine 3×3 Rotationsmatrix und $\mathbf{t}$ ein inhomogener 3D-Vektor. Diese beiden Daten beschreiben die Lage der Kamera im Raum. Diese Konvention ist analog zu der bei camera_calibration verwendeten Konvention, d.h. für $\mathtt{R} = \mathtt{I}$ und $\mathbf{t} = 0$ zeigt die x-Achse nach rechts, die y-Achse nach unten und die z-Achse nach vorne von der Bildebene weg. $\mathtt{K}$ ist die Kalibriermatrix der Kamera (die Kameramatrix), die sich wie folgt beschreiben lässt:

$$\mathtt{K} = \begin{pmatrix} af & sf & u \\ 0 & f & v \\ 0 & 0 & 1 \end{pmatrix} \ .$$

Hierbei ist f die Brennweite der Kamera in Pixeln, a das Seitenverhältnis der Pixel, s ein Faktor, der die Schrägstellung der Bildachsen modelliert und (u, v) der Hauptpunkt der Kamera in Pixeln. In dieser Konvention entspricht die x-Achse der Spaltenachse und die y-Achse der Zeilenachse.

Da die Kamera stationär ist, kann man $\mathbf{t} = 0$ annehmen. Mit dieser Konvention sieht man leicht, dass die vierte Koordinate des homogenen 3D-Vektors $\mathbf{X}$ keinen Einfluss auf die Lage des projizierten 3D-Punktes hat. Somit kann man die vierte Koordinate auch auf 0 setzen und sieht, dass $\mathbf{X}$ als Punkt der unendlich fernen Ebene betrachtet werden kann, und somit eine Richtung im Raum repräsentiert. Mit dieser Konvention kann man die vierte Koordinate von $\mathbf{X}$ auch weglassen und $\mathbf{X}$ als inhomogenen 3D-Vektor betrachten, der als Richtungsvektor nur bis auf Skalierung bestimmt werden kann. Somit kann die obige Projektionsgleichung auch wie folgt geschrieben werden:

$$\mathbf{x} = \mathtt{KR}\mathbf{X} \ .$$

Wenn man zwei Bilder desselben Punktes mit einer stationären Kamera aufnimmt, gilt folglich

$$\begin{aligned} \mathbf{x}_1 &= \mathtt{K}_1\mathtt{R}_1\mathbf{X} \\ \mathbf{x}_2 &= \mathtt{K}_2\mathtt{R}_2\mathbf{X} \end{aligned}$$

und somit

$$\mathbf{x}_2 = \mathtt{K}_2\mathtt{R}_2\mathtt{R}_1^{-1}\mathtt{K}_1^{-1}\mathbf{x}_1 = \mathtt{K}_2\mathtt{R}_{12}\mathtt{K}_1^{-1}\mathbf{x}_1 = \mathtt{H}_{12}\mathbf{x}_1 \ .$$

Wenn sich die Kameraparameter bei den zwei Aufnahmen nicht ändern, gilt $K_1 = K_2$. Die beiden Bilder desselben 3D-Punktes sind folglich über eine projektive 2D-Abbildung miteinander verknüpft. Diese Abbildung kann mit proj_match_points_ransac bestimmt werden. Es ist zu beachten, dass die Reihenfolge der Koordinaten in den projektive 2D-Abbildungen in HALCON umgekehrt ist als in der obigen Konvention. Außerdem ist zu beachten, dass proj_match_points_ransac ein Koordinatensystem verwendet, bei dem der Ursprung eines Pixels in seiner linken oberen Ecke liegt, während stationary_camera_self_calibration ein Koordinatensystem verwendet, dass dem Koordinatensystem von camera_calibration entspricht, bei dem der Ursprung eines Pixels in seiner Mitte liegt. Für die mit proj_match_points_ransac bestimmten projektive 2D-Abbildungen müssen noch die Zeilen und Spalten vertauscht werden und eine Verschiebung um $(0.5, 0.5)$ angewendet werden, so dass in HALCON statt $\mathtt{H}_{12} = \mathtt{K}_2\mathtt{R}_{12}\mathtt{K}_1^{-1}$ gilt

$$\mathtt{H}_{12} = \begin{pmatrix} 0 & 1 & 0.5 \\ 1 & 0 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \mathtt{K}_2\mathtt{R}_{12}\mathtt{K}_1^{-1} \begin{pmatrix} 0 & 1 & -0.5 \\ 1 & 0 & -0.5 \\ 0 & 0 & 1 \end{pmatrix}$$

bzw.

$$\mathtt{K}_2\mathtt{R}_{12}\mathtt{K}_1^{-1} = \begin{pmatrix} 0 & 1 & -0.5 \\ 1 & 0 & -0.5 \\ 0 & 0 & 1 \end{pmatrix} \mathtt{H}_{12} \begin{pmatrix} 0 & 1 & 0.5 \\ 1 & 0 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} .$$

Aus der obigen Gleichung können Bedingungsgleichungen für die Kameraparameter auf zwei Arten hergeleitet werden. Zum einen kann durch Elimination der Rotation eine Gleichung hergeleitet werden, die die Kameramatrizen mit der projektive 2D-Abbildung zwischen zwei Bildern verknüpft. Sei $\mathtt{H}_{ij}$ die projektive Abbildung von Bild i nach Bild j. Dann gilt

$$\begin{aligned} \mathtt{K}_j\mathtt{K}_j^\top &= \mathtt{H}_{ij}\mathtt{K}_i\mathtt{K}_i^\top\mathtt{H}_{ij}^\top \\ \mathtt{K}_j^{-\top}\mathtt{K}_j^{-1} &= \mathtt{H}_{ij}^{-\top}\mathtt{K}_i^{-\top}\mathtt{K}_i^{-1}\mathtt{H}_{ij}^{-1} \end{aligned}$$

Aus der zweiten Gleichung können lineare Gleichungen zur Bestimmung der Kameraparameter gewonnen werden. Dieses Verfahren wird von `stationary_camera_self_calibration` für `EstimationMethod` = *'linear'* verwendet. Dabei werden für i alle in `MappingSource` verwendeten Bilder und für j alle in `MappingDest` verwendeten Bilder zur Aufstellung der Gleichungen benutzt. Nachdem die Kameraparameter bestimmt worden sind, kann die Rotation der Kamera in den einzelnen Bildern über die Gleichung $\mathtt{R}_{ij} = \mathtt{K}_j^{-1}\mathtt{H}_{ij}\mathtt{K}_i$ und Bildung der Abbildungskette vom Referenzbild `ReferenceImage` aus berechnet werden. Aus der ersten Gleichung kann ein nichtlineares Verfahren zur Bestimmung der Kameraparameter durch Minimierung des folgenden Fehlers hergeleitet werden:

$$E = \sum_{(i,j)\in\{(s,d)\}} \left\| \mathtt{K}_j\mathtt{K}_j^\top - \mathtt{H}_{ij}\mathtt{K}_i\mathtt{K}_i^\top\mathtt{H}_{ij}^\top \right\|_F^2$$

Hierbei ist $\{(s,d)\}$ wie beim linearen Verfahren die durch `MappingSource` und `MappingDest` gegebene Menge von überlappenden Bildern. Dieses Verfahren wird von `stationary_camera_self_calibration` für `EstimationMethod` = *'nonlinear'* verwendet. Dabei werden die Kameraparameter mit den Ergebnissen des linearen Verfahrens initialisiert. Diese beiden Verfahren sind sehr schnell und lieferen akzeptable Ergebnisse, wenn die projektiven 2D-Abbildungen $\mathtt{H}_{ij}$ hinreichend genau sind. Dazu ist es unerlässlich, dass die Bilder keine radialen Verzeichnungen aufweisen. Man sieht aber auch, dass die Bestimmung der Kameraparameter unabhängig von der Bestimmung der Rotationen erfolgt, und somit die möglichen Bedingungsgleichungen nicht vollständig ausgeschöpft werden. Insbesondere kann man sehen, dass nicht erzwungen wird, dass die Projektionen desselben Punktes nahe beieinander liegen. Deswegen bietet `stationary_camera_self_calibration` als drittes Verfahren eine Bündelausgleichung an (`EstimationMethod` = *'gold_standard'*). Dabei werden sowohl die Kameraparameter und Rotationen, als auch Richtungen im 3D (die oben angeführten Vektoren $\mathbf{X}$), die zur Minimierung des folgenden Fehlers führen, in einer gemeinsamen Optimierung bestimmt:

$$E = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} \left\| \mathbf{x}_{ij} - \mathtt{K}_i\mathtt{R}_i\mathbf{X}_j \right\|^2 + \frac{1}{\sigma^2}(u_i^2 + v_i^2) \right)$$

Hierbei werden nur Terme berücksichtigt, für die die rekonstruierte Richtung $\mathbf{X}_j$ im Bild i sichtbar ist. Als Startwerte für die Bündelausgleichung werden Parameter verwendet, die aus den Ergebnissen des nichtlinearen Verfahrens berechnet werden. Aufgrund der hohen Komplexität der Minimierung dauert die Bündelausgleichung erheblich länger als die beiden einfacheren Verfahren. Sie liefert aber auch signifikant bessere Ergebnisse, und sollte deshalb bevorzugt werden.

In jedem der drei Verfahren kann festgelegt werden, welche Kameraparameter bestimmt werden sollen. Die jeweils anderen Kameraparameter werden auf einem festen Wert gehalten. Die Festlegung erfolgt mit dem Parameter `CameraModel`, der ein Tupel von Werten enthält. `CameraModel` muss immer den Wert *'focus'* enthalten, der festlegt, dass die Brennweite f bestimmt wird. Wenn `CameraModel` den Wert *'principal_point'* enthält, wird der Hauptpunkt (u, v) der Kamera bestimmt, ansonsten wird er auf den Wert $(\mathtt{ImageWidth}/2, \mathtt{ImageHeight}/2)$ gesetzt. Wenn `CameraModel` den Wert *'aspect'* enthält, wird das Seitenverhältnis a der Pixel bestimmt, ansonsten wird es auf 1 gesetzt. Wenn `CameraModel` den Wert *'skew'* enthält, wird die Schrägstellung s der Bildachsen bestimmt, ansonsten wird sie auf 0 gesetzt. Es sind nur bestimmte Kombinationen dieser Parameter erlaubt: *'focus'*, *['focus', 'principal_point']*, *['focus', 'aspect']*, *['focus', 'principal_point', 'aspect']* und *['focus', 'principal_point', 'aspect', 'skew']*.

Zusätzlich zu den obigen Parametern ist es bei Verwendung von EstimationMethod = *'gold_standard'* möglich mit dem Parameter Kappa die Verzeichnung der Bilder zu schätzen. Hierfür kann in diesem Fall CameraModel um den Parameter *'kappa'* erweitert werden. Kappa entspricht dem Verzeichnungsparemeter κ des Divisionsmodells für Verzeichnungen (siehe camera_calibration).

Bei der Verwendung von EstimationMethod = *'gold_standard'* zur Bestimmung des Hauptpunkts ist es möglich, weit vom Bildmittelpunkt entfernt liegende Schätzungen zu bestrafen. Hierzu kann ein Sigma an den Parameter *'principal_point:0.5'* angehängt werden. Wird kein Sigma angegeben, so entfällt der Strafterm in der obigen Fehlerformel.

Der Parameter FixedCameraParams legt fest, ob sich die Kameraparameter pro Bild ändern können oder ob sie für alle Bilder als konstant angenommen werden sollen. Um eine Kamera zu kalibrieren und später mit der kalibrierten Kamera zu messen, ist nur FixedCameraParams = *'true'* sinnvoll. Der Modus FixedCameraParams = *'false'* ist hauptsächlich zur Berechnung von Mosaiken mit gen_spherical_mosaic sinnvoll, falls die Kamera bei der Aufnahme des Mosaiks gezoomt hat oder sich die Scharfstellung signifikant geändert hat. Falls ein Mosaik mit konstanten Kameraparametern berechnet werden soll, sollte natürlich auch FixedCameraParams = *'true'* verwendet werden. Es ist zu beachten, dass für FixedCameraParams = *'false'* in praktischen Anwendungen das Problem, speziell für Aufnahmen mit langen Brennweiten, sehr schlecht bestimmt ist. In diesen Fällen kann oft nur die Brennweite bestimmt werden. Deswegen ist es hier unter Umständen erforderlich, CameraModel = *'focus'* zu wählen oder die Lage des Hauptpunktes einzuschränken, indem ein kleines Sigma für den Strafterm des Hauptpunktes gewählt wird.

Die Anzahl der Bilder, die zur Kalibrierung verwendet werden, wird in NumImages übergeben. Aus der Anzahl der Bilder ergeben sich Einschränkungen für das Kameramodell. So können bei der Verwendung von nur zwei Bildern selbst bei der Annahme von konstanten Kameraparametern nicht alle Parameter bestimmt werden. Hier sollte sinnvollerweise die Schrägstellung der Bildachsen auf 0 gesetzt werden, indem *'skew' nicht* in CameraModel aufgenommen wird. Falls FixedCameraParams = *'false'* verwendet wird, können in keinem Fall alle Kameraparameter bestimmt werden. Hier sollte ebenfalls mindestens die Schrägstellung der Bildachsen auf 0 gesetzt werden. Weiterhin ist zu beachten, dass das Seitenverhältnis der Pixel nur korrekt bestimmt werden kann, wenn mindestens ein Bild um die optische Achse (die z-Achse des Kamerakoordinatensystems) gegenüber den anderen Bildern verdreht aufgenommen worden ist. Falls dies nicht der Fall ist, sollte die Bestimmung des Seitenverhältnisses unterdrückt werden, indem der Wert *'aspect' nicht* in CameraModel aufgenommen wird.

Wie oben beschrieben, ist es zur Kalibrierung notwendig, dass mit proj_match_points_ransac für jedes überlappende Bildpaar die projektive Abbildung zwischen den zwei beteiligten Bildern bestimmt wird. So sind z.B. für einen 2×2 Block von Bildern in folgender Anordnung

1	2
3	4

unter der Annahme, dass sich alle Bilder gegenseitig überlappen, folgende projektive Abbildungen zu bestimmen: 1↦2, 1↦3, 1↦4, 2↦3, 2↦4 und 3↦4. Die Indizes der Bilder, die die jeweilige Abbildung bestimmen, werden in MappingSource und MappingDest übergeben. Die Indizes der Bilder beginnen ab 1. In obigem Beispiel ist also MappingSource = *[1,1,1,2,2,3]* und MappingDest = *[2,3,4,3,4,4]* zu verwenden. Die Gesamtanzahl von Bildern, die zur Kalibrierung verwendet wird, wird in NumImages übergeben. Sie wird verwendet, um zu überprüfen, ob jedes Bild über eine Kette von Abbildungen erreicht werden kann. Der Index des Referenzbildes wird in ReferenceImage übergeben. Dieses Bild erhält in der Ausgabe als Rotation die Einheitsmatrix.

Die zu den Bildpaaren gehörigen 3×3 projektiven Transformationsmatrizen werden in HomMatrices2D übergeben. Zusätzlich müssen in Rows1, Cols1, Rows2 und Cols2 die Koordinaten der zugeordneten Punktpaare in den einzelnen Bildpaaren übergeben werden. Sie können aus der Ausgabe von proj_match_points_ransac mit tuple_select oder mit der HDevelop-Funktion subset bestimmt werden. Damit stationary_camera_self_calibration bestimmen kann, zu welcher Abbildung ein Punktpaar gehört, muss in NumCorrespondences für jedes Bildpaar die Anzahl der gefundenen Zuordnungen übergeben werden.

Die berechneten Kameramatrizen K_i werden in CameraMatrices als 3×3 Matrizen zurückgegeben. Für FixedCameraParams = *'false'* werden NumImages Matrizen zurückgegeben. Da für FixedCameraParams = *'true'* alle Kameramatrizen identisch sind, wird in diesem Fall nur eine Kameramatrix zurückgegeben. Die berechneten Rotationen R_i werden in RotationMatrices als 3×3 Matrizen zurückgegeben. RotationMatrices enthält NumImages Matrizen.

Falls EstimationMethod = *'gold_standard'* verwendet wird, werden in (X, Y, Z) die rekonstruierten Richtungen $\mathbf{X}_j$ zurückgegeben. Zusätzlich wird in Error der mittlere Projektionsfehler der rekonstruierten Richtungen

zurückgegeben. Dies kann verwendet werden, um zu überprüfen, ob die Optimierung zu sinnvollen Werten konvergiert ist.

Falls die berechneten Kameraparameter zur Projektion von 3D-Punkten oder 3D-Richtungen in das Bild i verwendet werden sollen, sollte die jeweilige Kameramatrix mit der Rotationsmatrix multipliziert werden (mit `hom_mat2d_compose`).

Parameter

- ▷ **NumImages** (input_control) . . . integer ⇝ *integer*
  Anzahl der verschiedenen Bilder, die zur Kalibrierung verwendet werden.
  Restriktion: NumImages >= 2
- ▷ **ImageWidth** (input_control) . . . extent.x ⇝ *integer*
  Breite der Bilder, aus denen die Punkte extrahiert wurden.
  Restriktion: ImageWidth > 0
- ▷ **ImageHeight** (input_control) . . . extent.y ⇝ *integer*
  Höhe der Bilder, aus denen die Punkte extrahiert wurden.
  Restriktion: ImageHeight > 0
- ▷ **ReferenceImage** (input_control) . . . integer ⇝ *integer*
  Index des Referenzbildes.
- ▷ **MappingSource** (input_control) . . . integer-array ⇝ *integer*
  Indizes der Ausgangsbilder der Transformationen.
- ▷ **MappingDest** (input_control) . . . integer-array ⇝ *integer*
  Indizes der Zielbilder der Transformationen.
- ▷ **HomMatrices2D** (input_control) . . . hom_mat2d-array ⇝ *real*
  Array von 3×3 projektiven Transformationsmatrizen.
- ▷ **Rows1** (input_control) . . . point.y-array ⇝ *real* / integer
  Zeilenkoordinaten korrespondierender Punkte in den jeweiligen Ausgangsbildern.
- ▷ **Cols1** (input_control) . . . point.x-array ⇝ *real* / integer
  Spaltenkoordinaten korrespondierender Punkte in den jeweiligen Ausgangsbildern.
- ▷ **Rows2** (input_control) . . . point.y-array ⇝ *real* / integer
  Zeilenkoordinaten korrespondierender Punkte in den jeweiligen Zielbildern.
- ▷ **Cols2** (input_control) . . . point.x-array ⇝ *real* / integer
  Spaltenkoordinaten korrespondierender Punkte in den jeweiligen Zielbildern.
- ▷ **NumCorrespondences** (input_control) . . . integer-array ⇝ *integer*
  Anzahl der Punktkorrespondenzen im jeweiligen Bildpaar.
- ▷ **EstimationMethod** (input_control) . . . string ⇝ *string*
  Schätzalgorithmus für die Kalibrierung.
  Defaultwert: 'gold_standard'
  Werteliste: `EstimationMethod` ∈ {'linear', 'nonlinear', 'gold_standard'}
- ▷ **CameraModel** (input_control) . . . string-array ⇝ *string*
  Verwendetes Kameramodell.
  Defaultwert: ['focus','principal_point']
  Werteliste: `CameraModel` ∈ {'focus', 'aspect', 'skew', 'principal_point', 'kappa'}
- ▷ **FixedCameraParams** (input_control) . . . string ⇝ *string*
  Sind die Kameraparameter für alle Bilder identisch?
  Defaultwert: 'true'
  Werteliste: `FixedCameraParams` ∈ {'true', 'false'}
- ▷ **CameraMatrices** (output_control) . . . hom_mat2d-array ⇝ *real*
  (Array von) 3×3 projektiven Kameramatrizen, die die internen Kameraparameter bestimmen.
- ▷ **Kappa** (output_control) . . . real(-array) ⇝ *real*
  Radiale Verzeichnung der Kamera.
- ▷ **RotationMatrices** (output_control) . . . hom_mat2d-array ⇝ *real*
  Array von 3×3 Transformationsmatrizen, die die Rotation der Kamera im jeweiligen Bild bestimmen.
- ▷ **X** (output_control) . . . point3d.x-array ⇝ *real*
  Richtungsparameter in X-Richtung aller Punkte, wenn `EstimationMethod` = *'gold_standard'* verwendet wird.

▷ **Y** (output_control) . . . . . point3d.y-array ⇝ *real*
Richtungsparameter in Y-Richtung aller Punkte, wenn `EstimationMethod` = *'gold_standard'* verwendet wird.

▷ **Z** (output_control) . . . . . point3d.z-array ⇝ *real*
Richtungsparameter in Z-Richtung aller Punkte, wenn `EstimationMethod` = *'gold_standard'* verwendet wird.

▷ **Error** (output_control) . . . . . real(-array) ⇝ *real*
Durchschnittlicher Fehler pro rekonstruiertem Punkt, wenn `EstimationMethod` = *'gold_standard'* verwendet wird.

Beispiel

```
* Assume that Images contains four images in the layout given in the
* above description.  Then the following example performs the camera
* self-calibration using these four images.
From := [1,1,1,2,2,3]
To := [2,3,4,3,4,4]
HomMatrices2D := []
Rows1 := []
Cols1 := []
Rows2 := []
Cols2 := []
NumMatches := []
for J := 0 to |From|-1 by 1
    select_obj (Images, ImageF, From[J])
    select_obj (Images, ImageT, To[J])
    points_foerstner (ImageF, 1, 2, 3, 100, 0.1, 'gauss', 'true', \
                      RowsF, ColsF, _, _, _, _, _, _, _, _)
    points_foerstner (ImageT, 1, 2, 3, 100, 0.1, 'gauss', 'true', \
                      RowsT, ColsT, _, _, _, _, _, _, _, _)
    proj_match_points_ransac (ImageF, ImageT, RowsF, ColsF, RowsT, ColsT, \
                               'ncc', 10, 0, 0, 480, 640, 0, 0.5, \
                               'gold_standard', 2, 42, HomMat2D, \
                               Points1, Points2)
    HomMatrices2D := [HomMatrices2D,HomMat2D]
    Rows1 := [Rows1,subset(RowsF,Points1)]
    Cols1 := [Cols1,subset(ColsF,Points1)]
    Rows2 := [Rows2,subset(RowsT,Points2)]
    Cols2 := [Cols2,subset(ColsT,Points2)]
    NumMatches := [NumMatches,|Points1|]
endfor
stationary_camera_self_calibration (4, 640, 480, 1, From, To, \
                                    HomMatrices2D, Rows1, Cols1, \
                                    Rows2, Cols2, NumMatches, \
                                    'gold_standard', \
                                    ['focus','principal_point'], \
                                    'true', CameraMatrix, Kappa, \
                                    RotationMatrices, X, Y, Z, Error)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `stationary_camera_self_calibration` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

proj_match_points_ransac, proj_match_points_ransac_guided

Nachfolger

gen_spherical_mosaic

Siehe auch

gen_projective_mosaic

Literatur

Lourdes Agapito, E. Hayman, I. Reid: „Self-Calibration of Rotating and Zooming Cameras“; International Journal of Computer Vision; vol. 45, no. 2; pp. 107–127; 2001.

Modul

Calibration

Kapitel 15

Klassifikation

15.1 Gaußsches Mischverteilungsmodell

add_class_train_data_gmm (: : GMMHandle, ClassTrainDataHandle :)

Hinzufügen von Trainingsdaten zu einem Gaußschen Mischverteilungsmodell (GMM).

add_class_train_data_gmm fügt die Trainingsdaten ClassTrainDataHandle zu einem Gaußschen Mischverteilungsmodell (GMM), gegeben durch GMMHandle, hinzu.

Parameter

▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
Handle eines GMMs.

▷ **ClassTrainDataHandle** (input_control) ..... class_train_data ⇝ *handle*
Handle der Trainingsdaten für den Klassifikator.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_class_train_data_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- GMMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_class_gmm, create_class_train_data

Nachfolger

get_sample_class_gmm

Alternativen

add_sample_class_gmm

Siehe auch

create_class_gmm

Modul

Foundation

add_sample_class_gmm (: : GMMHandle, Features, ClassID, Randomize :)

Hinzufügen eines Trainingsmusters zu den Trainingsdaten eines Gaußschen Mischverteilungsmodells.

add_sample_class_gmm fügt ein Trainingsmuster zu dem durch GMMHandle gegebenen Gaußsche Mischverteilungsmodell (GMM) hinzu. Das Trainingsmuster wird durch Features und ClassID beschrieben. Features ist der Merkmalsvektor des Musters und muss folglich die Länge NumDim besitzen, die in create_class_gmm angegeben worden ist. ClassID ist die Klasse des Musters, eine Zahl zwischen 0 und NumClasses-1 (ebenfalls festgelegt in create_class_gmm).

Im Spezialfall, in dem die Features-Vektoren vom Typ **Integer** sind, treten sie im Merkmalsraum in einem Gitter mit Schrittweite 1.0 auf. Wenn zum Beispiel RGB-Merkmale für Farbklassifikation verwendet werden, haben die Daten einen Integerwert von 0 bis 255 je Komponente. Normalerweise kann auch mehr als ein Merkmal denselben Punkt im Raum darstellen. Während des GMM-Trainings mit solchen Daten neigt der Algorithmus dazu, die modellierten Gaußschen Verteilungen zu linear abhängigen Linien oder Ebenen von Punkten, die parallel zu den Gitterrichtungen liegen, anzupassen. Falls die Anzahl von Zentren, die im Centers von train_class_gmm zurückgegeben wird, ungewöhnlich hoch ist, deutet das auf ein solches Verhalten des Algorithmus hin. Um dieses Problem umzugehen, kann der Parameter Randomize verwendet werden. Für Randomize $>$ 0.0 wird Gaußsches Rauschen mit Mittelwert 0 und Standardabweichung Randomize zu jeder Komponente des Trainingsvektors addiert und die Trainingsdaten werden im GMM gespeichert. Randomize-Werte kleiner als 1.0 verbessern die Situation nicht. Randomize-Werte deutlich größer als 2.0 beeinflussen die Form der Datenwolke zu viel. Für Integer-Merkmalsvektoren ist es empfehlenswert, Werte zwischen 1.5 und 2.0 zu verwenden. Falls die Daten durch Skalierung aus Integerdaten erzeugt wurden, kann dasselbe Problem auftreten. Hier muss Randomize entsprechend der Skalierung der Daten skaliert werden.

Bevor das GMM mit train_class_gmm trainiert werden kann, müssen *alle* Trainingsmuster mit add_sample_class_gmm zum GMM hinzugefügt werden.

Die Anzahl der abgespeicherten Trainingsmuster kann mit get_sample_num_class_gmm abgefragt werden. Einzelne Trainingsmuster können mit get_sample_class_gmm wieder ausgelesen werden.

Im Normalfall ist es günstig, die Trainingsmuster mit write_samples_class_gmm in eine Datei zu speichern, damit sie wiederverwendet werden können. Damit können bei Bedarf neue Trainingsmuster zum Datensatz hinzugefügt werden und ein *neu erzeugtes* GMM kann mit dem erweiterten Datensatz *neu* trainiert werden.

Parameter

- ▷ **GMMHandle** (input_control) . . . . . class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **Features** (input_control) . . . . . real-array ⇝ *real*
  Merkmalsvektor des zu speichernden Trainingsmusters.
- ▷ **ClassID** (input_control) . . . . . number ⇝ *integer*
  Klasse des zu speichernden Trainingsmusters.
- ▷ **Randomize** (input_control) . . . . . real ⇝ *real*
  Standardabweichung des Gaußschen Rauschens, das zu den Trainingsdaten hinzugefügt wird.
  Defaultwert: 0.0
  Wertevorschläge: Randomize ∈ {0.0, 1.5, 2.0}
  Restriktion: Randomize >= 0.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_sample_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- GMMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_gmm`

Nachfolger

`train_class_gmm`, `write_samples_class_gmm`

Alternativen

`read_samples_class_gmm`, `add_samples_image_class_gmm`

Siehe auch

`clear_samples_class_gmm`, `get_sample_num_class_gmm`, `get_sample_class_gmm`

Modul

Foundation

classify_class_gmm (: : GMMHandle, Features, Num : ClassID, ClassProb, Density, KSigmaProb)

Berechnen der Klasse eines Merkmalsvektors durch ein Gaußsches Mischverteilungsmodell.

`classify_class_gmm` berechnet mit dem durch `GMMHandle` gegebenen Gaußsche Mischverteilungsmodell (GMM) die `Num` besten Klassen des Merkmalsvektors `Features` und gibt die Klassen in `ClassID` und die zugehörigen Wahrscheinlichkeiten der Klassen in `ClassProb` zurück. Das GMM muss vor der Verwendung von `classify_class_gmm` mit `train_class_gmm` trainiert werden.

`classify_class_gmm` entspricht dem Aufruf von `evaluate_class_gmm` und der zusätzlichen Bestimmung der besten `Num` Klassen. Wie bei `evaluate_class_gmm` beschrieben, können die Ausgabewerte `ClassProb` des GMM als Wahrscheinlichkeiten des Auftretens der jeweiligen Klasse interpretiert werden. Im Normalfall, sollte es ausreichend sein, `Num` $= 1$ zu verwenden, um zu entscheiden, ob die Wahrscheinlichkeit der besten Klasse des Merkmalsvektors hoch genug ist. In manchen Anwendungen kann es aber auch interessant sein, die zweitbeste Klasse zu betrachten (`Num` $= 2$), insbesondere, wenn zu erwarten ist, dass sich die Klassen signifikant überlappen.

Die Parameter `Density` und `KSigmaProb` sind bei `evaluate_class_gmm` beschrieben.

Parameter

- ▷ **GMMHandle** (input_control) . . . . . class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **Features** (input_control) . . . . . real-array ⇝ *real*
  Merkmalsvektor.
- ▷ **Num** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der zu bestimmenden besten Klassen.
  Defaultwert: 1
  Wertevorschläge: `Num` ∈ {1, 2, 3, 4, 5}
- ▷ **ClassID** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ergebnis der Klassifikation des Merkmalsvektors durch das GMM.
- ▷ **ClassProb** (output_control) . . . . . real-array ⇝ *real*
  Die a-posteriori-Klassenwahrscheinlichkeit.
- ▷ **Density** (output_control) . . . . . real-array ⇝ *real*
  Wahrscheinlichkeitsdichte des Merkmalsvektors.
- ▷ **KSigmaProb** (output_control) . . . . . real-array ⇝ *real*
  Normalisierte k-Sigma-Wahrscheinlichkeit für den Merkmalsvektor.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `classify_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_gmm`, `read_class_gmm`

Alternativen

`evaluate_class_gmm`

Siehe auch

`create_class_gmm`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Mario A.T. Figueiredo: „Unsupervised Learning of Finite Mixture Models“; IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 3; March 2002.

Modul

Foundation

clear_class_gmm (: : GMMHandle :)

Löschen eines Gaußschen Mischverteilungsmodells.

`clear_class_gmm` löscht das Gaußsche Mischverteilungsmodell (GMM), das durch `GMMHandle` angegeben wird, und gibt sämtlichen für das GMM benötigten Speicher frei. Das GMM kann nach dem Aufruf von `clear_class_gmm` nicht mehr verwendet werden. Der Handle `GMMHandle` ist nach dem Aufruf ungültig.

Parameter

▷ **GMMHandle** (input_control) ..... class_gmm(-array) ⇝ *handle*
Handle des GMM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `clear_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- GMMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`classify_class_gmm`, `evaluate_class_gmm`

Siehe auch

`create_class_gmm`, `read_class_gmm`, `write_class_gmm`, `train_class_gmm`

Modul

Foundation

clear_samples_class_gmm (: : GMMHandle :)

Löschen aller Trainingsdaten eines Gaußschen Mischverteilungsmodells.

clear_samples_class_gmm löscht alle in dem Gaußschen Mischverteilungsmodell (GMM) GMMHandle abgespeicherten Trainingsmuster. Eine Verwendung von clear_samples_class_gmm ist nur dann sinnvoll, wenn das GMM in demselben Prozess trainiert wird, in dem das GMM auch zur Evaluierung mit evaluate_class_gmm oder zur Klassifikation mit classify_class_gmm verwendet wird. In diesem Fall kann der durch die Trainingsmuster belegte Speicher mit clear_samples_class_gmm wieder freigegeben werden, und somit Speicherplatz gespart werden. In der üblichen Verwendungsart, in der das GMM offline trainiert wird und mit write_class_gmm gespeichert wird, ist die Verwendung von clear_samples_class_gmm normalerweise überflüssig, da write_class_gmm die Trainingsmuster nicht abspeichert, und somit im Online-Prozess, der das GMM mit read_class_gmm einliest, auch kein Speicher für die Trainingsmuster benötigt wird.

Parameter

▷ **GMMHandle** (input_control) ..... class_gmm(-array) ⇝ *handle*
Handle des GMM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert clear_samples_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- GMMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

train_class_gmm, write_samples_class_gmm

Siehe auch

create_class_gmm, clear_class_gmm, add_sample_class_gmm, read_samples_class_gmm

Modul

Foundation

create_class_gmm (: : NumDim, NumClasses, NumCenters, CovarType, Preprocessing, NumComponents, RandSeed : GMMHandle)

Erzeugen eines Gaußschen Mischverteilungsmodells zur Klassifikation

create_class_gmm erzeugt ein Gaußsches Mischverteilungsmodell (GMM), das zur Klassifikation verwendet werden kann. NumDim gibt die Anzahl der Dimensionen des Merkmalsraums an, NumClasses gibt die Anzahl der zu trainierenden Klassen an. Ein GMM besteht aus einer Anzahl NumCenters an *Gaußschen Zentren* je Klasse. NumCenters kann dabei nicht nur die exakt zu verwendende Anzahl an Zentren spezifizieren, sondern in Abhängigkeit der Anzahl der übergebenen Parameter auch eine Ober- und Untergrenze:

Genau ein Parameter: Der Parameter entspricht exakt der zu verwendenden Anzahl an Zentren für alle Klassen.

Genau zwei Parameter: Der erste Parameter entspricht der minimalen Anzahl an Zentren, der zweite der maximalen Anzahl für alle Klassen.

Genau $2 \cdot NumClasses$ **Parameter:** Abwechselnd jeder erste Parameter entspricht der minimalen Anzahl Zentren je Klasse und jeder zweite Parameter der maximalen Anzahl an Zentren je Klasse.

Werden Ober- und Untergrenzen für die Anzahl der Zentren spezifiziert, dann wird die optimale Anzahl an Zentren aus dem Minimale-Nachrichtenlänge-Kriterium (Minimum Message Length Criterion, MML) bestimmt. Im Allgemeinen ist es empfehlenswert, für das spätere Training mit (zu) vielen Zentren als Maximum und der erwarteten Anzahl Zentren als Minimum zu beginnen.

Ein Zentrum wird jeweils von den Parametern Mittelpunkt $\mathbf{m}_j$, Kovarianzmatrix $\mathbf{C}_j$ und Mischungskoeffizient (mixing coefficient) P_j beschrieben. Diese Parameter werden aus den Trainingsdaten mittels des Expectation-Maximization (EM) Algorithmus bestimmt. Ein GMM kann eine beliebige Wahrscheinlichkeitsdichtefunktion annähern, vorausgesetzt es werden genügend Zentren verwendet. Die Kovarianzmatrizen $\mathbf{C}_j$ haben eine Größe von `NumDim` · `NumDim` (`NumComponents` · `NumComponents`, falls eine Vorverarbeitung verwendet wird) und sind symmetrisch. Weitere Bedingungen werden mit `CovarType` festgelegt:

Für `CovarType` = *'spherical'* ist $\mathbf{C}_j$ ein skalares Vielfaches der Einheitsmatrix $\mathbf{C}_j = s_j^2\mathbf{I}$. Die Zentrum-Dichtefunktion $p(\mathbf{x}|j)$ wird damit

$$p(\mathbf{x}|j) = \frac{1}{(2\pi s_j^2)^{d/2}} \exp(-\frac{\|\mathbf{x}-\mathbf{m}_j\|^2}{2s_j^2})$$

Für `CovarType` = *'diag'* ist $\mathbf{C}_j$ eine Diagonalmatrix $\mathbf{C}_j = diag(s_{j,1}^2, ..., s_{j,d}^2)$. Die zugehörige Zentrum-Dichtefunktion $p(\mathbf{x}|j)$ ist

$$p(\mathbf{x}|j) = \frac{1}{(2\pi \prod_{i=1}^{d} s_{j,i}^2)^{d/2}} \exp(-\sum_{i=1}^{d} \frac{(x_i - m_{j,i})^2}{2s_{j,i}^2})$$

Für `CovarType` = *'full'* ist $\mathbf{C}_j$ eine positiv definite Matrix. Die zugehörige Zentrum-Dichtefunktion $p(\mathbf{x}|j)$ ist

$$p(\mathbf{x}|j) = \frac{1}{(2\pi)^{d/2}|\mathbf{C}_j|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}-\mathbf{m}_j)^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m}_j))$$

Die Komplexität der Berechnungen steigt von `CovarType` = *'spherical'* über `CovarType` = *'diag'* bis `CovarType` = *'full'* an. Gleichzeitig nimmt jedoch die Flexibilität der Zentrum zu. *'spherical'* benötigt deshalb im Allgemeinen höhere Werte für `NumCenters` als *'full'*.

Die allgemeine Vorgehensweise, um GMM zu benutzen, ist wie folgt: Zuerst wird mit `create_class_gmm` ein GMM erzeugt. Anschließend werden mit `add_sample_class_gmm` Trainingsvektoren hinzugefügt, diese können anschließend mit `write_samples_class_gmm` in einer Datei gespeichert werden. Mit `train_class_gmm` werden die o.a. Zentrenparameter bestimmt. Die können weiterhin mit `write_class_gmm` für spätere Klassifikationen gespeichert werden.

Aus den Mischungskoeffizienten P_j und den Zentren-Dichtefunktionen $p(\mathbf{x}|j)$ wird die Wahrscheinlichkeitsdichtefunktion $p(\mathbf{x})$ berechnet:

$$p(x) = \sum_{j=1}^{n_{comp}} P(j)p(\mathbf{x}|j)$$

Eine Funktion der Wahrscheinlichkeitsdichtefunktion $p(\mathbf{x})$ kann mittels `evaluate_class_gmm` für einen Merkmalsvektor $\mathbf{x}$ bestimmt werden. `classify_class_gmm` hingegen sortiert die $p(\mathbf{x})$ und gibt damit die wahrscheinlichsten Klassen des Merkmalsvektors an.

Die Parameter `Preprocessing` und `NumComponents` verwendet man, um die Trainingsdaten vorzuverarbeiten. Dabei können die Dimensionen der Daten reduziert werden. Diese Parameter sind in `create_class_mlp` beschrieben.

`create_class_gmm` initialisiert die Koordinaten der Zentren mit Zufallszahlen. Damit die Ergebnisse des Trainierens des GMM mit `train_class_gmm` reproduzierbar werden, wird in `RandSeed` der Initialisierungswert des Zufallszahlengenerators angegeben.

Parameter

▷ **NumDim** (input_control) ..... integer ⇝ *integer*
Anzahl der Dimensionen des Merkmalsraums.
Defaultwert: 3
Wertevorschläge: `NumDim` ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Restriktion: NumDim >= 1

▷ **NumClasses** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Klassen des GMM.
Defaultwert: 5
Wertevorschläge: NumClasses ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Restriktion: NumClasses >= 1

▷ **NumCenters** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der Zentren pro Klasse.
Defaultwert: 1
Wertevorschläge: NumCenters ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30}
Restriktion: NumClasses >= 1

▷ **CovarType** (input_control) . . . . . string ⇝ *string*
Art der Kovarianzmatrizen.
Defaultwert: 'spherical'
Werteliste: CovarType ∈ {'spherical', 'diag', 'full'}

▷ **Preprocessing** (input_control) . . . . . string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
Defaultwert: 'normalization'
Werteliste: Preprocessing ∈ {'none', 'normalization', 'principal_components', 'canonical_variates'}

▷ **NumComponents** (input_control) . . . . . integer ⇝ *integer*
Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale (ignoriert bei Preprocessing = *'none'* und Preprocessing = *'normalization'*).
Defaultwert: 10
Wertevorschläge: NumComponents ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Restriktion: NumComponents >= 1

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Initialisierungswert des Zufallszahlengenerators, der zur Initialisierung des GMM mit zufälligen Werten verwendet wird.
Defaultwert: 42

▷ **GMMHandle** (output_control) . . . . . class_gmm ⇝ *handle*
Handle des GMM.

Beispiel

```
* Classification with Gaussian Mixture Models
create_class_gmm (NumDim , NumClasses, [1,5], 'full', 'none',\
                  NumComponents, 42, GMMHandle)
* Add the training data
for J := 0 to NumData-1 by 1
    * Features := [...]
    * ClassID := [...]
    add_sample_class_gmm (GMMHandle, Features, ClassID, Randomize)
endfor
* Train the GMM
train_class_gmm (GMMHandle, 100, 0.001, 'training', 0.0001, Centers, Iter)
* Classify unknown data in 'Features'
classify_class_gmm (GMMHandle, Features, 1, ID, Prob, Density, KSigmaProb)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`add_sample_class_gmm`, `add_samples_image_class_gmm`

Alternativen

`create_class_mlp`, `create_class_svm`

Siehe auch

`clear_class_gmm`, `train_class_gmm`, `classify_class_gmm`, `evaluate_class_gmm`, `classify_image_class_gmm`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Mario A.T. Figueiredo: „Unsupervised Learning of Finite Mixture Models“; IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 3; March 2002.

Modul

Foundation

deserialize_class_gmm (: : SerializedItemHandle : GMMHandle)

Deserialisieren eines Gaußschen Mischverteilungsmodell (GMM).

`deserialize_class_gmm` deserialisiert ein Gaußsches Mischverteilungsmodell (GMM) (einschließlich eventuell abgespeicherter Trainingsmuster), welches mit `serialize_class_gmm` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte GMM wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten GMM mit dem Handle `GMMHandle` gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **GMMHandle** (output_control) ..... class_gmm ⇝ *handle*
  Handle des GMM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_class_gmm`

Nachfolger

`classify_class_gmm`, `evaluate_class_gmm`, `create_class_lut_gmm`

Siehe auch

`create_class_gmm`, `write_class_gmm`, `serialize_class_gmm`

Modul

Foundation

evaluate_class_gmm (: : GMMHandle, Features : ClassProb, Density, KSigmaProb)

Berechnen der Bewertung eines Merkmalsvektors durch ein Gaußsches Mischverteilungsmodell.

evaluate_class_gmm berechnet drei verschiedene Wahrscheinlichkeitswerte für einen Merkmalsvektor Features durch das Gaußsche Mischverteilungsmodell (GMM) GMMHandle.

Die a-posteriori-Wahrscheinlichkeit einer Klasse i für ein Merkmal Features(x) wird über

$$p(i|\mathbf{x}) = \sum_{j=1}^{n_{comp}} P(j)p(\mathbf{x}|j)$$

berechnet und in ClassProb pro Klasse zurückgegeben. Die Berechnungsformeln für die Aktivitäten/Zentren-Dichtefunktion p(x|j) sind bei create_class_gmm angegeben.

Die Dichtefunktion von einem Features Vektor wird als Summe der Klassenwahrscheinlichkeiten berechnet

$$p(\mathbf{x}) = \sum_{i=1}^{n_{classes}} Pr(i)p(i|\mathbf{x})$$

und in Density zurückgegeben. Hierbei sind $Pr(i)$ die a-priori-Wahrscheinlichkeiten der Klassen, so wie sie von train_class_gmm berechnet werden. Density kann zur sog. Novelty-Detection verwendet werden, d.h. zur Rückweisung von Merkmalsvektoren, die keiner der trainierten Klassen entsprechen. Da Density jedoch von der Skalierung der Merkmalswerte abhängt und als Dichtewert auch nicht zwischen 0 und 1 liegen muss, kann die Novelty-Detection im Normalfall über KSigmaProb (s.u.) einfacher durchgeführt werden.

Ein *k-Sigma-Fehler-Ellipsoid* ist der geometrische Ort von Punkten $\mathbf{x}$, für die

$$(\mathbf{x} - \mu)^T C^{-1} (\mathbf{x} - \mu) = k^2$$

gilt. Im eindimensionalen Fall ist er das Intervall $[\mu - k\sigma, \mu + k\sigma]$. Für beliebige 1D-Normalverteilungen gilt, dass für $k = 1$ ca. 68% der Werte der Zufallsvariablen innerhalb dieses Intervalls auftreten, für $k = 2$ ca. 95%, für $k = 3$ ca. 99% usw. Diese Wahrscheinlichkeit wird k-Sigma-Wahrscheinlichkeit genannt. Sie wird mit $P[k]$ bezeichnet. $P[k]$ kann numerisch berechnet werden, wobei zu beachten ist, dass für gleiche Werte von k, $P^{(N)}[k] > P^{(N+1)}[k]$ (hier sind N und (N+1) die Dimensionen des Merkmalsvektors). Für die GMM wird die k-Sigma-Wahrscheinlichkeit wie folgt berechnet:

$$P_{GMM}[\mathbf{x}] = \sum_{j=1}^{n_{comp}} P(j)P_j[k_j], \quad \text{wobei} \quad k_j^2 = (\mathbf{x} - \mu_j)^T C_j^{-1} (\mathbf{x} - \mu_j)$$

$P_{GMM}[k]$ wird schließlich noch mit den a-priori-Wahrscheinlichkeiten gewichtet und dann normalisiert. Der größte Wert aller Klassen wird in KSigmaProb zurückgegeben:

$$\text{KSigmaProb} = \frac{1}{\Pr_{max}} \max\left(Pr(i)P_{GMM}[\mathbf{x}]\right)$$

KSigmaProb kann zur Novelty-Detection verwendet werden, da dieser Wert angibt, wie gut ein Merkmal in die Werteverteilung der Klasse passt, der es zugeordnet wurde. Normalerweise werden Merkmale, die einen KSigmaProb-Wert kleiner als 0.0001 haben, zurückgewiesen. Der Schwellenwert, der in classify_image_class_gmm über den Parameter RejectionThreshold definiert wird, bezieht sich auf die entsprechenden KSigmaProb-Werte.

Das GMM muss vor der Verwendung von evaluate_class_gmm mit train_class_gmm trainiert werden.

Die Position des Maximums von ClassProb wird typischerweise als die Klasse des Merkmalsvektors interpretiert und der zugehörige Wert als die Wahrscheinlichkeit der Klasse. In diesem Fall sollte statt evaluate_class_gmm classify_class_gmm verwendet werden, da classify_class_gmm direkt die Klasse und Wahrscheinlichkeit zurückliefert.

Parameter

▷ **GMMHandle** (input_control) . . . . . class_gmm ⇝ *handle*
Handle des GMM.

▷ **Features** (input_control) . . . . . real-array ⇝ *real*
Merkmalsvektor.

▷ **ClassProb** (output_control) . . . . . real-array ⇝ *real*
Die a-posteriori-Klassenwahrscheinlichkeit.

▷ **Density** (output_control) . . . . . real ⇝ *real*
Wahrscheinlichkeitsdichte des Merkmalsvektors.

▷ **KSigmaProb** (output_control) . . . . . real ⇝ *real*
Normalisierte k-Sigma-Wahrscheinlichkeit für den Merkmalsvektor.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `evaluate_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_gmm`, `read_class_gmm`

Alternativen

`classify_class_gmm`

Siehe auch

`create_class_gmm`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Mario A.T. Figueiredo: „Unsupervised Learning of Finite Mixture Models“; IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 3; March 2002.

Modul

Foundation

get_class_train_data_gmm (: : GMMHandle : ClassTrainDataHandle)

Liest die Trainingsdaten eines Gaußschen Mischverteilungsmodell (GMM) aus.

`get_class_train_data_gmm` liest die Trainingsdaten eines Gaußschen Mischverteilungsmodell (GMM) aus und liefert diese als Handle in `ClassTrainDataHandle` zurück.

Parameter

▷ **GMMHandle** (input_control) . . . . . class_gmm ⇝ *handle*
Handle eines GMMs.

▷ **ClassTrainDataHandle** (output_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten des GMMs.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_class_train_data_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

add_sample_class_gmm, read_samples_class_gmm

Nachfolger

add_class_train_data_svm, add_class_train_data_svm, add_class_train_data_knn

Siehe auch

create_class_train_data

Modul

Foundation

get_params_class_gmm (: : GMMHandle : NumDim, NumClasses, MinCenters, MaxCenters, CovarType)

Auslesen der Parameter eines Gaußschen Mischverteilungsmodells.

get_params_class_gmm gibt die Parameter des durch GMMHandle angegebenen Gaußschen Mischverteilungsmodells (GMM) zurück, die bei der Erzeugung mit create_class_gmm verwendet wurden. Dies ist insbesondere nützlich, wenn das GMM mit read_class_gmm eingelesen wurde. Die Ausgabedaten von get_params_class_gmm können z.B. verwendet werden, um zu überprüfen, ob die Merkmalsvektoren und ggf. die Zieldaten, die verwendet werden sollen, zu dem GMM passen. Zur Beschreibung der Parameter siehe create_class_gmm.

Parameter

- ▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **NumDim** (output_control) ..... integer ⇝ *integer*
  Anzahl der Dimensionen des Merkmalsraums.
- ▷ **NumClasses** (output_control) ..... integer ⇝ *integer*
  Anzahl der Klassen des GMM.
- ▷ **MinCenters** (output_control) ..... integer-array ⇝ *integer*
  Minimale Anzahl von Zentren pro GMM-Klasse.
- ▷ **MaxCenters** (output_control) ..... integer-array ⇝ *integer*
  Maximale Anzahl von Zentren pro GMM-Klasse.
- ▷ **CovarType** (output_control) ..... string ⇝ *string*
  Art der Kovarianzmatrizen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_params_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_class_gmm, read_class_gmm

Nachfolger

add_sample_class_gmm, train_class_gmm

Siehe auch

evaluate_class_gmm, classify_class_gmm

Modul

Foundation

get_prep_info_class_gmm (: : GMMHandle, Preprocessing : InformationCont, CumInformationCont)

Berechnen des Informationsgehaltes der vorverarbeiteten Merkmalsvektoren eines GMM.

get_prep_info_class_gmm berechnet den Informationsgehalt der mit der durch Preprocessing gegebenen Vorverarbeitung transformierten Komponenten der Trainingsvektoren. Preprocessing kann auf *'principal_components'* oder *'canonical_variates'* gesetzt werden. Die zugrundeliegenden Vorverarbeitungen sind bei create_class_gmm beschrieben. Der Informationsgehalt wird aus die Variation der transformierten Komponenten der Trainingsvektoren berechnet, d.h. er wird rein aufgrund der Trainingsdaten unabhängig von einer Fehlerrate bei einer Klassifikation der Trainingsdaten berechnet. Der Informationsgehalt wird für alle relevanten Komponenten der transformierten Merkmalsvektoren (NumComponents für *'principal_components'* und *'canonical_variates'*, siehe create_class_gmm) in InformationCont zurückgegeben. Der Informationsgehalt wird als eine Zahl zwischen 0 und 1 dargestellt. Ein prozentualer Informationsgehalt kann leicht durch Multiplikation mit 100 berechnet werden. Der kumulierte Informationsgehalt der ersten n Komponenten wird in der n-ten Komponente von CumInformationCont zurückgegeben, d.h. CumInformationCont enthält die Summen der ersten n Elemente von InformationCont. Um get_prep_info_class_gmm verwenden zu können, müssen mit add_sample_class_gmm oder read_samples_class_gmm genügend viele Trainingsmuster zu dem durch GMMHandle gegebenen GMM hinzugefügt werden.

InformationCont und CumInformationCont können dazu verwendet werden, um zu entscheiden, wie viele Komponenten der transformierten Merkmalsvektoren relevante Information enthalten. Ein oft verwendetes Kriterium ist z.B. zu verlangen, dass die transformierten Daten x% (z.B. 90%) der Daten enthalten sollten. Dies kann leicht anhand des ersten Wertes von CumInformationCont, der über x% liegt, bestimmt werden. Der so erhaltene Wert kann bei einem neuerlichen Aufruf von create_class_gmm als NumComponents verwendet werden. Da zum Aufruf von get_prep_info_class_gmm schon ein GMM mit create_class_gmm erzeugt werden muss, also auch ein initialer Wert von NumComponents bei create_class_gmm angegeben werden muss, aber bei Verwendung von get_prep_info_class_gmm typischerweise noch nicht bekannt ist, wie viele Komponenten relevant sind, empfiehlt sich folgendes zweistufiges Vorgehen, um NumComponents zu bestimmen: In einem ersten Schritt wird ein GMM mit der maximalen Anzahl von NumComponents (NumComponents für *'principal_components'* und *'canonical_variates'*) erzeugt. Dann werden die Trainingsmuster zu dem GMM hinzugefügt und mit write_samples_class_gmm in Datei gespeichert. Anschließend wird mit get_prep_info_class_gmm der Informationsgehalt der Komponenten und somit NumComponents bestimmt. Danach wird ein neues GMM mit der gewünschten Anzahl Komponenten erzeugt, und die abgespeicherten Trainingsdaten mit read_samples_class_gmm wieder eingelesen. Hierauf wird das GMM mit train_class_gmm trainiert.

Parameter

- ▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **Preprocessing** (input_control) ..... string ⇝ *string*
  Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
  Defaultwert: 'principal_components'
  Werteliste: Preprocessing ∈ {'principal_components', 'canonical_variates'}
- ▷ **InformationCont** (output_control) ..... real-array ⇝ *real*
  Relativer Informationsgehalt der transformierten Merkmalsvektoren.
- ▷ **CumInformationCont** (output_control) ..... real-array ⇝ *real*
  Kumulierter Informationsgehalt der transformierten Merkmalsvektoren.

Beispiel

```
* Create the initial GMM
create_class_gmm (NumDim, NumClasses, NumCenters, 'full',\
                  'principal_components', NumComponents, 42, GMMHandle)
* Generate and add the training data
for J := 0 to NumData-1 by 1
    * Generate training features and classes
    * Data = [...]
    * ClassID = [...]
    add_sample_class_gmm (GMMHandle, Data, ClassID, Randomize)
```

```
endfor
write_samples_class_gmm (GMMHandle, 'samples.gtf')
* Compute the information content of the transformed features
get_prep_info_class_gmm (GMMHandle, 'principal_components',\
                         InformationCont, CumInformationCont)
* Determine Comp by inspecting InformationCont and CumInformationCont
* NumComponents = [...]
* Create the actual GMM
create_class_gmm (NumDim, NumClasses, NumCenters, 'full',\
                  'principal_components', NumComponents, 42, GMMHandle)
* Train the GMM
read_samples_class_gmm (GMMHandle, 'samples.gtf')
train_class_gmm (GMMHandle, 200, 0.0001, 0.0001, Regularize, Centers, Iter)
write_class_gmm (GMMHandle, 'classifier.gmm')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_prep_info_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls `get_prep_info_class_gmm` den Fehler 9211 (Matrix ist nicht positiv definit) bei `Preprocessing` = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_class_gmm`, `read_samples_class_gmm`

Nachfolger

`clear_class_gmm`, `create_class_gmm`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Andrew Webb: „Statistical Pattern Recognition“; Arnold, London; 1999.

Modul

Foundation

get_sample_class_gmm (: : GMMHandle, NumSample : Features, ClassID)

Auslesen eines Trainingsmusters aus den Trainingsdaten eines Gaußschen Mischverteilungsmodells (GMM).

`get_sample_class_gmm` liest ein Trainingsmuster, das mit `add_sample_class_gmm` oder `add_samples_image_class_gmm` abgespeichert wurde, aus dem Gaußschen Mischverteilungsmodell (GMM) `GMMHandle` aus. Der Index des auszulesenden Musters wird mit `NumSample` festgelegt. Er wird ab 0 gezählt, d.h. `NumSample` muss zwischen 0 und `NumSamples` − 1 liegen, wobei `NumSamples` mit `get_sample_num_class_gmm` bestimmt werden kann. Das Trainingsmuster wird in `Features` und `ClassID` zurückgegeben. Dabei ist `Features` ein Merkmalsvektor der Länge `NumDim` und `ClassID` dessen Klasse (siehe `add_sample_class_gmm` und `create_class_gmm`).

`get_sample_class_gmm` kann z.B. dazu verwendet werden, die Trainingsdaten mit `classify_class_gmm` zu reklassifizieren, um festzustellen, welche der Trainingsmuster eventuell falsch klassifiziert werden.

Parameter

- ▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **NumSample** (input_control) ..... integer ⇝ *integer*
  Nummer des gespeicherten Trainingsmusters.
- ▷ **Features** (output_control) ..... real-array ⇝ *real*
  Merkmalsvektor des Trainingsmusters.
- ▷ **ClassID** (output_control) ..... number ⇝ *integer*
  Klasse des Trainingsmusters.

Beispiel

```
create_class_gmm (2, 2, [1,10], 'spherical', 'none', 2, 42, GMMHandle)
read_samples_class_gmm (GMMHandle, 'samples.gsf')
train_class_gmm (GMMHandle, 100, 1e-4, 'training', 1e-4, Centers, Iter)
* Reclassify the training samples
get_sample_num_class_gmm (GMMHandle, NumSamples)
for I := 0 to NumSamples-1 by 1
  get_sample_class_gmm (GMMHandle, I, Features, Class)
  classify_class_gmm (GMMHandle, Features, 2, ClassID, ClassProb,\
                      Density, KSigmaProb)
  if (not (Class == ClassProb[0]))
    * classified incorrectly
  endif
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_sample_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_class_gmm`, `add_samples_image_class_gmm`, `read_samples_class_gmm`, `get_sample_num_class_gmm`

Nachfolger

`classify_class_gmm`, `evaluate_class_gmm`

Siehe auch

`create_class_gmm`

Modul

Foundation

get_sample_num_class_gmm (: : GMMHandle : NumSamples)

Auslesen der Anzahl der Trainingsmuster in den Trainingsdaten eines Gaußschen Mischverteilungsmodell (GMM).

`get_sample_num_class_gmm` liefert in `NumSamples` die Anzahl der in dem durch `GMMHandle` gegebenen Gaußsche Mischverteilungsmodell (GMM) abgespeicherten Trainingsmuster zurück. `get_sample_num_class_gmm` sollte aufgerufen werden, bevor mit `get_sample_class_gmm` die einzelnen Trainingsmuster ausgelesen werden, z.B. um eine Reklassifikation der Trainingsdaten durchzuführen (siehe `get_sample_class_gmm`).

Parameter

▷ **GMMHandle** (input_control) . . . . . class_gmm ⇝ *handle*
Handle des GMM.

▷ **NumSamples** (output_control) . . . . . integer ⇝ *integer*
Anzahl der gespeicherten Trainingsmuster.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_sample_num_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_class_gmm`, `add_samples_image_class_gmm`, `read_samples_class_gmm`

Nachfolger

`get_sample_class_gmm`

Siehe auch

`create_class_gmm`

Modul

Foundation

read_class_gmm (: : FileName : GMMHandle)

Lesen eines Gaußschen Mischverteilungsmodells aus einer Datei.

`read_class_gmm` liest ein mit `write_class_gmm` abgespeichertes Gaußsche Mischverteilungsmodell (GMM) ein. Da das Trainieren eines GMM relativ lange dauern kann, wird das GMM typischerweise in einem Offline-Prozess trainiert und mit `write_class_gmm` gespeichert. Im Online-Prozess wird das GMM mit `read_class_gmm` gelesen und danach zur Evaluierung mit `evaluate_class_gmm` oder zur Klassifikation mit `classify_class_gmm` verwendet. Die Dateiendung für den GMM-Klassifikator ist in HALCON 'ggc'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.ggc`

▷ **GMMHandle** (output_control) . . . . . class_gmm ⇝ *handle*
Handle des GMM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`classify_class_gmm`, `evaluate_class_gmm`, `create_class_lut_gmm`

Siehe auch

create_class_gmm, write_class_gmm

Modul

Foundation

read_samples_class_gmm (: : GMMHandle, FileName :)

Lesen von Trainingsdaten eines Gaußschen Mischverteilungsmodells aus einer Datei.

read_samples_class_gmm liest Trainingsmuster aus der durch FileName gegebenen Datei aus und fügt sie zu den schon in dem Gaußschen Mischverteilungsmodell (GMM) GMMHandle vorhandenen Trainingsmustern hinzu. Das GMM muss zuvor mit create_class_gmm erzeugt werden. Wie bei train_class_gmm und write_samples_class_gmm beschrieben, können read_samples_class_gmm, add_sample_class_gmm und write_samples_class_gmm dazu verwendet werden, eine Datenbank aus Trainingsmustern aufzubauen, um somit die Leistung des GMM durch erneutes Trainieren zu verbessern.

Es ist zu beachten, dass die Trainingsdaten die korrekte Dimensionalität aufweisen. Die in FileName gespeicherten Merkmalsvektoren müssen die Längen NumDim besitzen, die bei create_class_gmm angegeben worden ist, ebenso müssen in create_class_gmm genügend Klassen angelegt worden sein. Falls dies nicht der Fall ist, wird eine Fehlermeldung zurückgegeben.

Es können Dateien von Trainingsdaten gelesen werden die mit write_samples_class_svm oder write_samples_class_mlp erstellt wurden..

Parameter

- ▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **FileName** (input_control) ..... filename.read ⇝ *string*
  Name der Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_samples_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- GMMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_class_gmm

Nachfolger

train_class_gmm

Alternativen

add_sample_class_gmm

Siehe auch

write_samples_class_gmm, write_samples_class_mlp, clear_samples_class_gmm

Modul

Foundation

select_feature_set_gmm (: : ClassTrainDataHandle, SelectionMethod, GenParamName, GenParamValue : GMMHandle, SelectedFeatureIndices, Score)

Wählt aus einer Liste an Merkmalen eine optimale Untermenge für ein bestimmtes Klassifikationsproblem aus.

select_feature_set_gmm wählt die optimalen Merkmale für ein bestimmtes Klassifikationsproblem aus. Die Daten für das Klassifikationsproblem werden mit ClassTrainDataHandle angegeben. Als Klassifikator wird ein Gaußsches Mischverteilungsmodell (GMM) verwendet. Eine nähere Beschreibung des GMMs kann bei create_class_gmm gefunden werden.

In GMMHandle wird ein Klassifikator zurückgegeben, der mit der resultierenden Merkmalsauswahl parametrisiert und trainiert wurde. Die Merkmalsauswahl wird in SelectedFeatureIndices zurückgegeben.

Der Operator kann für folgende Probleme nützlich sein: Es gibt eine Liste von Merkmalen von denen einige wenige für die Lösung eines Problems ausreichen. So kann mit Hilfe dieses Operators eine fundierte Entscheidung getroffen werden welche dieser Merkmale relevant für das gegebene Problem sind. Außerdem können für bestimmte Verfahren der Merkmalsextraktion verschiedene Parametersätze verglichen werden.

Um das Klassifikationsproblem zu definieren, werden dem Handle ClassTrainDataHandle Merkmalsvektoren hinzugefügt, die aus einzelnen Untergruppen bestehen. Diese Untergruppen müssen mit Hilfe des Operators set_feature_lengths_class_train_data festgelegt werden. Diese Untergruppen werden dann von dem in select_feature_set_gmm ausgeführten Selektionsprozess entweder ausgewählt oder abgelehnt, je nachdem ob sie zur Lösung des Klassifikationsproblems beitragen.

Die so definierten Merkmale werden dann in SelectedFeatureIndices mit ihrem Index referenziert, außer es wurden zusätzlich Namen beim Aufruf von set_feature_lengths_class_train_data gesetzt. Ist das der Fall, werden statt der Indizes die Namen der ausgewählten Merkmale zurückgegeben. Wurde set_feature_lengths_class_train_data zuvor nicht aufgerufen, werden die einzelnen Spalten als Merkmale interpretiert.

Für den Selektionsprozess können in SelectionMethod zwei verschiedene Methoden ausgewählt werden: entweder die Greedy-Variante *'greedy'* (das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt) oder die dynamisch oszillierende Suche *'greedy_oscillating'* (Das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt. Danach wird getestet ob eines der hinzugefügten Merkmale entbehrlich ist.). Während *'greedy'* schneller terminiert, kann *'greedy_oscillating'* bessere Ergebnisse erzielen wenn die Dimensionen der Merkmale sehr klein ist oder redundante Daten vorliegen.

Als Optimierungskriterium dient die Klassifikationsrate, die mit einem zweifachen Kreuzvalidierungsverfahren ermittelt wird. Die beste erreichte Klassifikationsrate wird in Score zurückgegeben.

Es gibt folgende generischen Parameter die über GenParamName und GenParamValue gesetzt werden können, deren genaue Bedeutung in create_class_gmm und train_class_gmm nachgelesen werden kann:

'min_centers': Minimale Anzahl an Cluster die in den Daten gebildet werden um sie zu repräsentieren. Mehr Cluster können die Klassifikation verbessern, gegeben die Anzahl der Trainingsdaten ist ausreichend hoch und aussagekräftig.
Mögliche Werte: '1', '2'
Standardwert: '1'

'max_center': Maximale Anzahl an Cluster die in den Daten gebildet werden um sie zu repräsentieren. Mehr Cluster können die Klassifikation verbessern, gegeben die Anzahl der Trainingsdaten ist ausreichend hoch und aussagekräftig.
Mögliche Werte: '1', '5', '10'
Standardwert: '1'

'covar_type': Art der Kovarianz, die zur Repräsentation der Größe eines Clusters eingesetzt wird.
Mögliche Werte: 'spherical', 'diag', 'full'
Standardwert: 'spherical'

'random_seed': Random seed.
Standardwert: '42'

'threshold': Schwellenwert für das Abbruchkriterium des Trainings.
Standardwert: '0.001'

'regularize': Regularisierende Komponente.
Standardwert: '0.0001'

'randomize': Randomisierung der Eingabevektoren.
Standardwert: '0'

'class_priors': Modus zur Bestimmung der a-priori-Wahrscheinlichkeiten der Klassen.
Mögliche Werte: 'training', 'uniform'
Standardwert: 'training'

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lange sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator `select_feature_set_gmm` zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

- ▷ **ClassTrainDataHandle** (input_control) . . . . . class_train_data ⇝ *handle*
  Handle der Trainingsdaten. Sollte zuvor in Untermerkmale aufgeteilt worden sein.
- ▷ **SelectionMethod** (input_control) . . . . . string ⇝ *string*
  Methode zur Auswahl der besten Merkmale.
  Defaultwert: 'greedy'
  Werteliste: `SelectionMethod` ∈ {'greedy', 'greedy_oscillating'}
- ▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
  Namen der generischen Parameter zum Steuern des Klassifikators.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'min_centers', 'max_center', 'covar_type', 'random_seed', 'threshold', 'regularize', 'randomize', 'class_priors'}
- ▷ **GenParamValue** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Werte der generischen Parameter zum Steuern des Klassifikators.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {1, 2, 3, 'spherical', 'diag', 'full', 42, 0.001, 0.0001, 0}
- ▷ **GMMHandle** (output_control) . . . . . class_gmm ⇝ *handle*
  Handle des trainierten Klassifikators.
- ▷ **SelectedFeatureIndices** (output_control) . . . . . string-array ⇝ *string*
  Die Indizes oder Namen der ausgewählten Merkmale.
- ▷ **Score** (output_control) . . . . . real-array ⇝ *real*
  Die Klassifikationsrate die mit dem ausgewählten Merkmalssatz erreicht wurde.

Beispiel

```
* Find out which of the two features distinguishes two Classes
NameFeature1 := 'Good Feature'
NameFeature2 := 'Bad Feature'
LengthFeature1 := 3
LengthFeature2 := 2
* Create training data
create_class_train_data (LengthFeature1+LengthFeature2,\
  ClassTrainDataHandle)
* Define the features which are in the training data
set_feature_lengths_class_train_data (ClassTrainDataHandle, [LengthFeature1,\
  LengthFeature2], [NameFeature1, NameFeature2])
* Add training data
*                                                    |Feat1|  |Feat2|
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  2,1  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  2,1  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  3,4  ], 0)
```

```
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  3,4  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [0,0,1,  5,6  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,3,2,  5,6  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [0,0,1,  5,6  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,3,2,  5,6  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [0,0,1,  5,6  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,3,2,  5,6  ], 1)
* Add more data
* ...
* Select the better feature with a GMM
select_feature_set_gmm (ClassTrainDataHandle, 'greedy', [], [], GMMHandle,\
  SelectedFeatureGMM, Score)
* Use the classifier
* ...
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `select_feature_set_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`create_class_train_data`, `add_sample_class_train_data`, `set_feature_lengths_class_train_data`

Nachfolger

`classify_class_gmm`

Alternativen

`select_feature_set_mlp`, `select_feature_set_knn`, `select_feature_set_svm`

Siehe auch

`create_class_gmm`, `gray_features`, `region_features`

Modul

Foundation

serialize_class_gmm (: : GMMHandle : SerializedItemHandle)

Serialisieren eines Gaußschen Mischverteilungsmodell (GMM).

`serialize_class_svm` serialisiert ein Gaußsches Mischverteilungsmodell (GMM) einschließlich eventuell abgespeicherter Trainingsmuster (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_class_gmm` und `write_samples_class_gmm` in eine Datei geschrieben werden. Das GMM wird in dem Handle `GMMHandle` übergeben. Das serialisierte GMM wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_class_gmm` wieder deserialisiert werden.

Parameter

▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
Handle des GMM.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_gmm`

Nachfolger

`clear_class_gmm`, `fwrite_serialized_item`, `send_serialized_item`, `deserialize_class_gmm`

Siehe auch

`create_class_gmm`, `read_class_gmm`, `write_samples_class_gmm`, `deserialize_class_gmm`

Modul

Foundation

train_class_gmm (: : GMMHandle, MaxIter, Threshold, ClassPriors, Regularize : Centers, Iter)

Trainieren eines Gaußschen Mischverteilungsmodells.

`train_class_gmm` trainiert das durch `GMMHandle` angegebene Gaußsche Mischverteilungsmodell (GMM). Bevor das GMM trainiert werden kann, müssen mit `add_sample_class_gmm`, `add_samples_image_class_gmm` oder `read_samples_class_gmm` alle im Training zu verwendenden Trainingsmuster in dem GMM abgespeichert werden. Es können auch nach dem Training noch neue zusätzliche Trainingsmuster hinzugefügt und trainiert werden.

Beim Training wird der Fehler, den das GMM auf den gespeicherten Trainingsdaten erzielt, durch das expectation maximization (EM) Verfahren minimiert.

`MaxIter` spezifiziert die maximale Anzahl von Iterationen pro Klasse für den EM-Algorithmus. In der Praxis sollten Werte zwischen *20* und *200* ausreichend für die meisten Probleme sein. `Threshold` spezifiziert eine Schwelle für die relative Änderung des Fehlers. Sollte bis zum Erreichen von `MaxIter` Iterationen die relative Änderung des Fehlers nicht unter `Threshold` liegen, so wird der Algorithmus für diese Klasse abgebrochen. Da der Algorithmus mit der maximal spezifizierten Anzahl Zentren (Parameter `NumCenters` in `create_class_gmm`) begonnen wird ist im Falle des vorzeitigen Abbruchs von einer nicht optimalen Anzahl Zentren und einem nicht optimalen Fehler für diese Klasse auszugehen. In diesem Fall kann ein neues Training mit veränderten Parametern (z.B. einem anderen Wert für `RandSeed` in `create_class_gmm`) versucht werden.

`ClassPriors` bestimmt die Berechnungsweise von a-priori-Wahrscheinlichkeiten der Klassen. Falls *'training'* eingegeben ist, wird die Wahrscheinlichkeit einer Klasse von dem Anteil der entsprechenden Trainingsmuster im Vergleich zu der gesamten Anzahl von Mustern berechnet. Falls *'uniform'* eingegeben ist, werden alle a-priori-Wahrscheinlichkeiten zu $1/$`NumClasses` gesetzt.

`Regularize` regularisiert (beinahe) *singuläre* Kovarianzmatrizen während des Trainings. Singuläre Kovarianzmatrizen treten nur bei ausschließlich linear abhängigen Trainingsvektoren auf. In diesem Fall wird `Regularize` wird zur Hauptdiagonale der Matrix addiert, so dass die Matrix nicht singulär wird. Es ist empfehlenswert einen Wert von *1e-4* zu übergeben. Wenn `Regularize` auf *0.0* gesetzt wird, wird keine Regularisierung der Matrix durchgeführt.

Die Mittelpunkte der Zentren werden anfangs auf zufällige Werte gesetzt. In Einzelfällen kann es dazu kommen, dass mit den durch `RandSeed` in `create_class_gmm` festgelegten Werten ein relativ hoher Fehler als optimaler Wert bestimmt wird, d.h., dass die Optimierung in einem lokalen Minimum steckenbleibt. Falls vermutet werden kann, dass dies passiert ist, sollte ein neues GMM mit einem anderen Wert für `RandSeed` erzeugt werden, um zu überprüfen, ob ein signifikant kleinerer Fehler erzeugt werden kann.

Es ist zu beachten, dass je nach Anzahl der Zentren, des Typs der Kovarianzmatrizen und der Anzahl der Trainingsmuster das Trainieren zwischen wenigen Sekunden und einigen Stunden dauern kann.

Als Ausgabe liefert `train_class_gmm` in `Centers` die Anzahl Zentren pro Klasse zurück, die von dem EM-Algorithmus als am besten geeignet festgestellt wurde. Diese Rückgabe kann als Referenz für zukünftig zu erstellende GMMs in `NumCenters` (in `create_class_gmm`) dienen. Wenn die Anzahl der gefundenen Zentren beim GMM-Training mit Integer-Daten unerwartet hoch ist, kann dass durch Anpassung des `Randomize`-Parameters in `add_sample_class_gmm` korrigiert werden. In `Iter` wird die Anzahl der durchgeführten Iterationen pro Klasse zurückgeliefert. Sollte ein Wert gleich `MaxIter` sein, so wurde der Algorithmus abgebrochen (s.o.).

Parameter

- ▷ **`GMMHandle`** (input_control) ..... class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **`MaxIter`** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von Iterationen des Expectation Maximization Algorithmus
  Defaultwert: 100
  Wertevorschläge: `MaxIter` ∈ {10, 20, 30, 50, 100, 200}
- ▷ **`Threshold`** (input_control) ..... real ⇝ *real*
  Schwellenwert für die relative Änderung des Fehlers damit der expectation maximization Algorithmus endet.
  Defaultwert: 0.001
  Wertevorschläge: `Threshold` ∈ {0.001, 0.0001}
  Restriktion: Threshold >= 0.0 && Threshold <= 1.0
- ▷ **`ClassPriors`** (input_control) ..... string ⇝ *string*
  Modus zur Bestimmung der a-priori-Wahrscheinlichkeiten der Klassen
  Defaultwert: 'training'
  Werteliste: `ClassPriors` ∈ {'training', 'uniform'}
- ▷ **`Regularize`** (input_control) ..... real ⇝ *real*
  Regularisierungswert zur Vermeidung der Singularität der Kovarianzmatrizen.
  Defaultwert: 0.0001
  Restriktion: Regularize >= 0.0 && Regularize < 1.0
- ▷ **`Centers`** (output_control) ..... integer-array ⇝ *integer*
  Anzahl der gefundenen Zentren pro Klasse
- ▷ **`Iter`** (output_control) ..... integer-array ⇝ *integer*
  Anzahl der durchgeführten Iterationen pro Klasse

Beispiel

```
create_class_gmm (NumDim, NumClasses, [1,5], 'full', 'none', 0, 42,\
                  GMMHandle)
* Add the training data
read_samples_class_gmm (GMMHandle, 'samples.gsf')
* Train the GMM
train_class_gmm (GMMHandle, 100, 1e-4, 'training', 1e-4, Centers, Iter)
* Write the Gaussian Mixture Model to file
write_class_gmm (GMMHandle, 'gmmclassifier.gmm')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `train_class_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- GMMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

add_sample_class_gmm, read_samples_class_gmm

Nachfolger

evaluate_class_gmm, classify_class_gmm, write_class_gmm, create_class_lut_gmm

Alternativen

read_class_gmm

Siehe auch

create_class_gmm

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Mario A.T. Figueiredo: „Unsupervised Learning of Finite Mixture Models“; IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 3; March 2002.

Modul

Foundation

write_class_gmm (: : GMMHandle, FileName :)

Abspeichern eines Gaußschen Mischverteilungsmodells in einer Datei.

write_class_gmm speichert das Gaußsche Mischverteilungsmodell (GMM) GMMHandle in der Datei FileName ab. Die Dateiendung für für den GMM-Klassifikator ist in HALCON 'ggc'. write_class_gmm wird typischerweise aufgerufen, nachdem das GMM mit train_class_gmm trainiert wurde. Das GMM kann mit read_class_gmm wieder eingelesen werden. write_class_gmm speichert *nicht* die im GMM eventuell abgespeicherten Trainingsmuster ab. Dazu ist write_samples_class_gmm zu verwenden.

Parameter

- ▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **FileName** (input_control) ..... filename.write ⇝ *string*
  Name der Datei.
  Dateiendung: .ggc

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_class_gmm

Nachfolger

clear_class_gmm

Siehe auch

create_class_gmm, read_class_gmm, write_samples_class_gmm

Modul

Foundation

write_samples_class_gmm (: : GMMHandle, FileName :)

Abspeichern der Trainingsdaten eines Gaußschen Mischverteilungsmodells in einer Datei.

write_samples_class_gmm speichert die in dem Gaußschen Mischverteilungsmodell GMM) GMMHandle abgespeicherten Trainingsmuster in der Datei FileName ab. write_samples_class_gmm kann dazu verwendet werden, eine Datenbank mit Trainingsmustern aufzubauen, und somit durch erneutes Trainieren mit einer erweiterten Datenbank die Leistung des GMM zu verbessern (siehe train_class_gmm).

Die Datei FileName wird von write_samples_class_gmm überschrieben. Eine Erweiterung der Datenbank der Trainingsmuster ist aber einfach möglich, da read_samples_class_gmm und add_sample_class_gmm die Trainingsmuster zu den bereits im Speicher des GMM gehaltenen Trainingsmustern hinzufügen.

Die erstellte Datei kann mit read_samples_class_mlp ausgelesen werden falls später der Klassifikator eines mehrschichtigen Perzeptrons (MLP) verwendet werden soll. Die Klasse eines Trainingsmusters im GMM entspricht beim MLP einer Komponente des Zielvektors mit Wert 1.0.

Parameter

▷ **GMMHandle** (input_control) . . . . . class_gmm ⇝ *handle*
Handle des GMM.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_samples_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_gmm

Nachfolger

clear_samples_class_gmm

Siehe auch

create_class_gmm, read_samples_class_gmm, read_samples_class_mlp, write_samples_class_mlp

Modul

Foundation

15.2 K-Nächste-Nachbarn

add_class_train_data_knn (: : KNNHandle, ClassTrainDataHandle :)

Hinzufügen von Trainingsdaten zu einem k-Nearest-Neighbor-Klassifikator (k-NN).

add_class_train_data_knn fügt die Trainingsdaten ClassTrainDataHandle zu einem k-Nearest-Neighbor-Klassifikator (k-NN), gegeben durch KNNHandle, hinzu.

Parameter

▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
Handle des k-NN-Klassifikators.

▷ **ClassTrainDataHandle** (input_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten für den Klassifikator.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_class_train_data_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- KNNHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_knn`, `create_class_train_data`

Nachfolger

`get_sample_class_knn`

Alternativen

`add_sample_class_knn`

Siehe auch

`create_class_knn`

Modul

Foundation

add_sample_class_knn (: : KNNHandle, Features, ClassID :)

Fügt ein Trainingsmuster zu einem k-NN-Klassifikator hinzu.

`add_sample_class_knn` fügt ein Trainingsmuster zu dem durch `KNNHandle` gegebenen k-Nearest-Neighbor-Klassifikators (k-NN) hinzu. Die Länge des angegebenen Trainingsmusters muss mit der in `create_class_knn` angegebenen Dimension `NumDim` übereinstimmen.

Das Trainingsmuster wird durch `Features` und `ClassID` beschrieben. `Features` ist ein Merkmalsvektor des Musters und muss folglich die Länge `NumDim` besitzen, die in `create_class_knn` angegeben worden ist. `ClassID` ist die Klasse des Musters, eine ganze Zahl, größer oder gleich 0. Bei der Verwendung von nur einer Klasse muss die Zahl 0 verwendet werden. Ebenso sollte bei der geplanten Verwendung des Operators `classify_image_class_knn` darauf geachtet werden, dass alle Klassenlabel von 0 bis zur Anzahl der verwendeten Klassen vergeben werden, da sonst für nicht benutzte Label leere Regionen erzeugt werden.

Es ist erlaubt weitere Trainingsdaten nach einem bereits erfolgten Training hinzuzufügen. Die Veränderungen werden erst wirksam, wenn `train_class_knn` erneut aufgerufen wird.

Falls der k-NN Klassifikator mit automatischer Merkmals-Normalisierung trainiert wurde interpretiert `add_sample_class_knn` das übergebene Muster als nicht normalisiert und intern wird eine Normalisierung durchgeführt, die durch den letzten Aufruf von `train_class_knn` definiert ist. Die Referenz-Dokumentation des `train_class_knn` Operators enthält weitere Informationen zur automatischen Merkmals-Normalisierung.

Parameter

- ▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
  Handle des k-NN-Klassifikators.
- ▷ **Features** (input_control) . . . . . number(-array) ⇝ *real*
  Liste der hinzuzufügenden Merkmalsvektoren.
- ▷ **ClassID** (input_control) . . . . . integer(-array) ⇝ *integer*
  Klassenlabel der Merkmalsvektoren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_sample_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- KNNHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`train_class_knn`, `read_class_knn`

Siehe auch

`create_class_knn`, `read_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

`classify_class_knn ( : : KNNHandle, Features : Result, Rating )`

Sucht nach den nächsten Nachbarn eines gegebenen Merkmalsvektors.

`classify_class_knn` sucht die nächsten *'k'* Nachbarn zu dem gegebenen Merkmalsvektor in `Features`. Die Nachbarschaft wird mit Hilfe der L2-Norm zwischen dem gegebenen Merkmalsvektors und den Trainingsbeispielen ermittelt.

Mithilfe von `set_params_class_knn` können sowohl die Anzahl *'k'* der verwendeten Nachbarn, als auch weitere Parameter, die das Verhalten von `classify_class_knn` steuern, eingestellt werden. Mit der Auswahl der Methode *'method'* und *'max_num_classes'* in `set_params_class_knn` können verschiedene Ergebnistypen für `classify_class_knn` ausgewählt werden:

'classes_distance': Ist diese Option gewählt, werden die Klassen der nächsten *'k'* Vektoren in `Result` zurückgegeben und deren Distanz zum Merkmalsvektor in `Rating`. Es wird allerdings nur das nächste Beispiel für eine Klasse verwendet, und es werden maximal *'max_num_classes'* Werte zurückgegeben. In einem k-NN besteht keine effiziente Möglichkeit exakt *'max_num_classes'* zu finden.

'classes_frequency': Ist diese Option gewählt, werden die häufigsten Klassen unter den nächsten *'k'* Vektoren zurückgegeben. Die relative Häufigkeit wird in `Rating` zurückgegeben. Es werden maximal *'max_num_classes'* Werte zurückgegeben.

'classes_weighted_frequency': Ist diese Option gewählt, werden die häufigsten Klassen unter den nächsten *'k'* Vektoren zurückgegeben allerdings sortiert nach der Häufigkeit gewichtet mit der jeweiligen Distanz zum Merkmalsvektor. Die gewichtete Häufigkeit wird in `Rating` zurückgegeben als Zahl zwischen 0.0 und 1.0. Es werden maximal *'max_num_classes'* Werte zurückgegeben.

'neighbors_distance': Ist diese Option gewählt, werden die nächsten *'k'* Nachbarn in `Result` und deren Distanzen in `Rating` zurückgegeben. Der Wert von *'max_num_classes'* wird hierbei ignoriert.

Im Standardfall gibt `classify_class_knn` die Klasse und Distanz des nächsten Beispiels zurück (entspricht *'classes_distance'*).

Parameter

▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
Handle des k-NN-Klassifikators.

▷ **Features** (input_control) . . . . . number-array ⇝ *real*
Merkmalsvektor der klassifiziert werden soll.

▷ **Result** (output_control) . . . . . number-array ⇝ *integer*
Ergebnisse der Klassifikation, entweder als Klassen oder Indizes.

▷ **Rating** (output_control) . . . . . number-array ⇝ *real*
Bewertung der Ergebnisse, entweder als Distanzen oder als relative oder gewichtete Häufigkeiten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `classify_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_knn`, `read_class_knn`, `set_params_class_knn`

Nachfolger

`clear_class_knn`

Siehe auch

`create_class_knn`, `read_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

clear_class_knn (: : KNNHandle :)

Löscht einen k-NN-Klassifikator.

`clear_class_knn` löscht den k-Nearest-Neighbor-Klassifikator (k-NN) gegeben durch `KNNHandle`. Das Handle ist nach dem Aufruf ungültig.

Parameter

▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
Handle des k-NN-Klassifikator.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `clear_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- KNNHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`train_class_knn`, `read_class_knn`

Siehe auch

`create_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

create_class_knn (: : NumDim : KNNHandle)

Erzeugen eines neuen k-Nearest-Neighbor-Klassifikators (k-NN).

`create_class_knn` erzeugt einen neuen k-Nearest-Neighbor-Klassifikator (k-NN). Der k-NN dient entweder zur Klassifikation oder um direkt nächste Nachbarn in einem Satz von Vektoren der Dimension `NumDim` zu finden.

Die meisten der im Kapitel Klassifikation/K-Nearest-Neighbor beschriebenen Operatoren beziehen sich auf einen derart initialisierten Klassifikator, der in `KNNHandle` zurückgegeben wird.

Der k-Nearest-Neighbor-Klassifikator ermittelt eine Klasse indem er die Klasse der k nächstgelegene Trainingsbeispiele ermittelt. Die Suche nach dem nächstgelegene Trainingsbeispiel wird approximiert durchgeführt, wodurch das Verfahren in logarithmischer Zeit zu der Anzahl an Trainingsbeispielen und deren Dimension ablaufen kann.

Die Dimension des Merkmalsvektors in `NumDim` ist der einzige Parameter der vorab festgelegt werden muss.

Parameter

▷ **NumDim** (input_control) ..... number-array ⇝ *integer*
Dimensionen des verwendeten Merkmalsvektors.
Defaultwert: 10

▷ **KNNHandle** (output_control) ..... class_knn ⇝ *handle*
Handle des k-NN-Klassifikators.

Ergebnis

`create_class_knn` liefert den Wert TRUE, falls der Parameter korrekt ist.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`add_sample_class_knn`, `train_class_knn`

Alternativen

`create_class_svm`, `create_class_mlp`

Siehe auch

`select_feature_set_knn`, `read_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

deserialize_class_knn (: : SerializedItemHandle : KNNHandle)

Deserialisieren eines serialisierten k-NN-Klassifikators.

deserialize_class_knn deserialisiert einen k-Nearest-Neighbor-Klassifikator (k-NN) (einschließlich eventuell abgespeicherter Trainingsmuster), welcher mit serialize_class_knn serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Der serialisierte k-NN-Klassifikator wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten k-NN-Klassifikator mit dem Handle KNNHandle gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **KNNHandle** (output_control) . . . . . class_knn ⇝ *handle*
Handle eines k-NN-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_class_knn

Nachfolger

classify_class_knn

Alternativen

serialize_class_knn

Siehe auch

create_class_knn

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

get_class_train_data_knn (: : KNNHandle : ClassTrainDataHandle)

Liest die Trainingsdaten eines k-NN-Klassifikators aus.

get_class_train_data_knn liest die Trainingsdaten eines k-NN Klassifikators aus und liefert diese als Handle in ClassTrainDataHandle zurück.

Parameter

▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
Handle eines k-NN-Klassifikators.

▷ **ClassTrainDataHandle** (output_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten des k-NN-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_class_train_data_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`add_sample_class_knn`

Nachfolger

`add_class_train_data_svm`, `add_class_train_data_gmm`, `add_class_train_data_knn`

Siehe auch

`create_class_train_data`

Modul

Foundation

`get_params_class_knn` (: : KNNHandle, GenParamName : GenParamValue)

Liest die Parameter eines k-NN-Klassifikators aus.

`get_params_class_knn` liest die Parameter des k-NN Klassifikators `KNNHandle` aus. Mögliche Werte für `GenParamName` sind die folgenden:

'method': Gibt die momentan gewählte Methode zur Ergebnisermittlung bei `classify_class_knn` zurück. Mögliche Rückgabewerte sind *'classes_distance'*, *'classes_frequency'*, *'classes_weighted_frequencies'* und *'neighbors_distance'*.

'k': Die Anzahl an Nachbarn die verwendet wird um das das Ergebnis von `classify_class_knn` zu bestimmen.

'max_num_classes': Die maximale Anzahl an Klassen die von `classify_class_knn` zurückgegeben werden soll. Dieser Wert wird bei der Auswahl der Methode *'neighbors_distance'* ignoriert.

'num_checks': Limitiert die Anzahl an Versuchen ein vorläufiges Ergebnis der k-NN-Suche zu verbessern.

'epsilon': Ein Parameter um die Genauigkeit der Suche zu verringern.

Parameter

▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
Handle eines k-NN-Klassifikators.

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der generischen Parameter des k-NN-Klassifikators.
Defaultwert: ['method','k']
Werteliste: `GenParamName` ∈ {'method', 'num_checks', 'epsilon', 'k'}

▷ **GenParamValue** (output_control) . . . . . number-array ⇝ *integer* / real / string
Werte der selektierten Parameter.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_params_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_class_knn, read_class_knn

Nachfolger

classify_class_knn

Siehe auch

create_class_knn, read_class_knn

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

get_sample_class_knn (: : KNNHandle, IndexSample : Features, ClassID)

Auslesen eines Trainingsmusters aus den Trainingsdaten eines k-NN-Klassifikators.

get_sample_class_knn liest ein Trainingsmuster, das mit add_sample_class_knn hinzugefügt wurde, aus dem k-Nearest-Neighbor-Klassifikator (k-NN) KNNHandle aus. Der Index des auszulesenden Musters wird mit IndexSample festgelegt. Er wird ab 0 gezählt, d.h. IndexSample muss zwischen 0 und NumSamples −1 liegen, wobei NumSamples mit get_sample_num_class_knn bestimmt werden kann. Das Trainingsmuster wird in Features und ClassID zurückgegeben. Dabei ist Features ein Merkmalsvektor der Länge NumDim (siehe create_class_knn) und ClassID die Klasse des Merkmalsvektors.

Parameter

- ▷ **KNNHandle** (input_control) ..... class_knn ⇝ *handle*
  Handle eines k-NN-Klassifikators.
- ▷ **IndexSample** (input_control) ..... integer ⇝ *integer*
  Index des Trainingsmusters.
- ▷ **Features** (output_control) ..... real-array ⇝ *real*
  Merkmalsvektor des Trainingsmusters.
- ▷ **ClassID** (output_control) ..... integer-array ⇝ *integer*
  Klasse des Trainingsmusters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sample_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_train_data

Siehe auch

create_class_knn

Modul

Foundation

get_sample_num_class_knn (: : KNNHandle : NumSamples)

Auslesen der Anzahl der Trainingsmuster in den Trainingsdaten eines k-NN-Klassifikators.

get_sample_num_class_knn liefert in NumSamples die Anzahl der verfügbaren Trainingsmuster in dem durch KNNHandle gegebenen k-Nearest-Neighbor-Klassifikators (k-NN) zurück. get_sample_num_class_knn sollte aufgerufen werden, bevor mit get_sample_class_knn die einzelnen Trainingsmuster ausgelesen werden, z.B. um eine Reklassifikation der Trainingsdaten durchzuführen.

Parameter

▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
Handle eines k-NN-Klassifikators.

▷ **NumSamples** (output_control) . . . . . integer ⇝ *integer*
Anzahl der gespeicherten Trainingsmuster.

Ergebnis

Falls KNNHandle korrekt ist, liefert get_sample_num_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_knn

Nachfolger

get_sample_class_knn

Siehe auch

create_class_knn

Modul

Foundation

read_class_knn (: : FileName : KNNHandle)

Liest einen k-NN-Klassifikator aus einer Datei.

read_class_knn liest einen k-Nearest-Neighbor-Klassifikator (k-NN) aus der Datei FileName und gibt ein Handle des k-NN-Klassifikators in KNNHandle zurück. Die Dateiendung für den k-NN-Klassifikator ist in HALCON 'gnc'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: .gnc

▷ **KNNHandle** (output_control) . . . . . class_knn ⇝ *handle*
Handle eines k-NN-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

classify_class_knn

Siehe auch

create_class_knn

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

select_feature_set_knn (: : ClassTrainDataHandle, SelectionMethod, GenParamName, GenParamValue : KNNHandle, SelectedFeatureIndices, Score)

Wählt aus einer Liste an Merkmalen eine optimale Untermenge für ein bestimmtes Klassifikationsproblem aus.

select_feature_set_knn wählt die optimalen Merkmale für ein bestimmtes Klassifikationsproblem aus. Die Trainingsdaten für das Klassifikationsproblem werden mit ClassTrainDataHandle angegeben. Als Klassifikator wird ein k-Nearest-Neighbor-Klassifikator (k-NN) verwendet. Einige Details über den k-NN-Klassifikator können in der Beschreibung von create_class_knn nachgelesen werden.

In KNNHandle wird ein Klassifikator zurückgegeben, der mit der resultierenden Merkmalsauswahl parametrisiert und trainiert wurde. Die Merkmalsauswahl wird in SelectedFeatureIndices zurückgegeben.

Der Operator kann für folgende Probleme nützlich sein: Es gibt eine Liste von Merkmalen von denen einige wenige für die Lösung eines Problems ausreichen. So kann mit Hilfe dieses Operators eine fundierte Entscheidung getroffen werden welche dieser Merkmale relevant für das gegebene Problem sind. Außerdem können für bestimmte Verfahren der Merkmalsextraktion verschiedene Parametersätze verglichen werden.

Um das Klassifikationsproblem zu definieren, werden dem Handle ClassTrainDataHandle Merkmalsvektoren hinzugefügt, die aus einzelnen Untergruppen bestehen. Diese Untergruppen müssen mit Hilfe des Operators set_feature_lengths_class_train_data festgelegt werden. Diese Untergruppen werden dann von dem in select_feature_set_knn ausgeführten Selektionsprozess entweder ausgewählt oder abgelehnt, je nachdem ob sie zur Lösung des Klassifikationsproblems beitragen.

Die so definierten Merkmale werden dann in SelectedFeatureIndices mit ihrem Index referenziert, außer es wurden zusätzlich Namen beim Aufruf von set_feature_lengths_class_train_data gesetzt. Ist das der Fall, werden statt der Indizes die Namen der ausgewählten Merkmale zurückgegeben. Wurde set_feature_lengths_class_train_data zuvor nicht aufgerufen, werden die einzelnen Spalten als Merkmale interpretiert.

Für den Selektionsprozess können in SelectionMethod zwei verschiedene Methoden ausgewählt werden: entweder die Greedy-Variante *'greedy'* (das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt) oder die dynamisch oszillierende Suche *'greedy_oscillating'* (Das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt. Danach wird getestet ob eines der hinzugefügten Merkmale entbehrlich ist.). Während *'greedy'* schneller terminiert, kann *'greedy_oscillating'* bessere Ergebnisse erzielen wenn die Dimensionen der Merkmale sehr klein ist oder redundante Daten vorliegen.

Als Optimierungskriterium dient die Klassifikationsrate, die mit einem zweifachen Kreuzvalidierungsverfahren ermittelt wird. Die beste erreichte Klassifikationsrate wird in Score zurückgegeben.

Für diesen Klassifikator können noch folgenden Parameter mit GenParamName und GenParamValue eingestellt werden:

'num_neighbors': Anzahl an Nachbarn die mindestens exakt ermittelt werden um den nächsten Nachbarn zu finden. Dieser Parameter sollte für hochdimensionale Eingaberäume erhöht werden.

Mögliche Werte: '1', '2', '5', '10'

Standardwert: '1'

'num_trees': Anzahl an Suchbäumen im k-NN-Klassifikator

Mögliche Werte: '1', '4', '10'

Standardwert: '4'

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lange sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator `select_feature_set_knn` zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

- ▷ **`ClassTrainDataHandle`** (input_control) . . . . . class_train_data ⇝ *handle*
  Handle der Trainingsdaten. Sollte zuvor in Untermerkmale aufgeteilt worden sein.
- ▷ **`SelectionMethod`** (input_control) . . . . . string ⇝ *string*
  Methode zur Auswahl der besten Merkmale.
  Defaultwert: 'greedy'
  Werteliste: `SelectionMethod` ∈ {'greedy', 'greedy_oscillating'}
- ▷ **`GenParamName`** (input_control) . . . . . string(-array) ⇝ *string*
  Namen der generischen Parameter zum Steuern des Klassifikators und der Merkmalsauswahl.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'num_neighbors', 'num_trees'}
- ▷ **`GenParamValue`** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Werte der generischen Parameter zum Steuern des Klassifikators und der Merkmalsauswahl.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {1, 2, 3}
- ▷ **`KNNHandle`** (output_control) . . . . . class_knn ⇝ *handle*
  Ein trainierter k-NN-Klassifikator.
- ▷ **`SelectedFeatureIndices`** (output_control) . . . . . string-array ⇝ *string*
  Die Indizes oder Namen der ausgewählten Merkmale.
- ▷ **`Score`** (output_control) . . . . . real-array ⇝ *real*
  Die Klassifikationsrate, die mit dem ausgewählten Merkmalssatz erreicht wurde.

Beispiel

```
* Find out which of the two features distinguishes two Classes
NameFeature1 := 'Good Feature'
NameFeature2 := 'Bad Feature'
LengthFeature1 := 3
LengthFeature2 := 2
* Create training data
create_class_train_data (LengthFeature1+LengthFeature2,\
  ClassTrainDataHandle)
* Define the features which are in the training data
set_feature_lengths_class_train_data (ClassTrainDataHandle, [LengthFeature1,\
  LengthFeature2], [NameFeature1, NameFeature2])
* Add training data
*                                                      |Feat1| |Feat2|
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  2,1  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  2,1  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  3,4  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  3,4  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [0,0,1,  5,6  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,3,2,  5,6  ], 1)
* Add more data
* ...
* Select the better feature with the k-NN classifier
select_feature_set_knn (ClassTrainDataHandle, 'greedy', [], [], KNNHandle,\
  SelectedFeatureKNN, Score)
* Use the classifier
* ...
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `select_feature_set_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`create_class_train_data`, `add_sample_class_train_data`, `set_feature_lengths_class_train_data`

Nachfolger

`classify_class_knn`

Alternativen

`select_feature_set_mlp`, `select_feature_set_svm`, `select_feature_set_gmm`

Siehe auch

`select_feature_set_trainf_knn`, `gray_features`, `region_features`

Modul

Foundation

serialize_class_knn (: : KNNHandle : SerializedItemHandle)

Serialisieren eines k-NN-Klassifikators.

`serialize_class_knn` serialisiert einen k-Nearest-Neighbor-Klassifikator (k-NN) einschließlich abgespeicherter Trainingsmuster (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_class_knn` in eine Datei geschrieben werden. Der k-NN-Klassifikator wird in dem Handle `KNNHandle` übergeben. Der serialisierte k-NN-Klassifikator wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_class_knn` wieder deserialisiert werden.

Parameter

▷ **KNNHandle** (input_control) ..... class_knn ⇝ *handle*
Handle eines k-NN-Klassifikators.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_knn`, `read_class_knn`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_class_knn`

Siehe auch

`create_class_knn`, `read_class_knn`, `deserialize_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

set_params_class_knn (: : KNNHandle, GenParamName, GenParamValue :)

Setzt die Klassifikationsparameter eines k-NN-Klassifikators.

`set_params_class_knn` setzt die Parameter der Klassifikation für den k-NN-Klassifikators `KNNHandle`. Das Setzen der Parameter beeinflusst die Ergebnisse und Verhalten von `classify_class_knn`.

Mithilfe von `set_params_class_knn` können sowohl die Anzahl *'k'* der verwendeten Nachbarn, als auch weitere Parameter, mit Hilfe von `GenParamName` und `GenParamValue` eingestellt werden. Das Erhöhen des Wertes für *'k'* erhöht die Genauigkeit der Ergebnisse, führt allerdings zu längeren Laufzeiten. Mit der Auswahl der Methode *'method'* und *'max_num_classes'* können verschiedene Ergebnistypen für `classify_class_knn` ausgewählt werden:

'classes_distance': Ist diese Option gewählt, werden die Klassen der nächsten *'k'* Vektoren zurückgegeben und deren Distanz zum Merkmalsvektor. Es wird allerdings nur das nächste Beispiel für eine Klasse verwendet, und es werden maximal *'max_num_classes'* Werte zurückgegeben. In einem k-NN besteht keine effiziente Möglichkeit exakt *'max_num_classes'* Klassen zu finden.

'classes_frequency': Ist diese Option gewählt, werden die häufigsten Klassen unter den nächsten *'k'* Vektoren zurückgegeben. Die jeweilige relative Häufigkeit der Klassen wird als Bewertung mit zurückgegeben. Die Anzahl der Rückgabewerte ist maximal *'max_num_classes'*.

'classes_weighted_frequency': Ist diese Option gewählt, werden die häufigsten Klassen unter den nächsten *'k'* Vektoren zurückgegeben allerdings sortiert nach der Häufigkeit gewichtet mit der jeweiligen Distanz zum Merkmalsvektor. Die jeweilige gewichtete Häufigkeit wird als Bewertung als Zahl zwischen 0.0 und 1.0 mit zurückgegeben. Die Anzahl der Rückgabewerte ist maximal *'max_num_classes'*.

'neighbors_distance': Ist diese Option gewählt, werden die nächsten *'k'* Nachbarn und deren Distanzen zurückgegeben. Der Wert von *'max_num_classes'* wird hierbei ignoriert.

Der Standardrückgabemethode ist *'classes_distance'*.

Zusätzlich können noch die Anzahl an Versuchen das Ergebnis zu verbessern mit *'num_checks'* eingestellt werden: Der Parameter muss positiv sein und der Standardwert ist 32. Eine Erhöhung führt zu genaueren Ergebnissen bei höherer Laufzeit. Das Setzen des Wertes auf 0 führt zu einer exakten Suche.

Zusätzlich kann noch mit *'epsilon'* ein Abbruchkriterium aktiviert werden, indem der Wert über den Standardwert 0.0 angehoben wird. Das führt unter Umständen zu schnelleren Ergebnissen, bei höherem Risiko, nicht den exakten nächsten Nachbarn zu finden.

Parameter

- ▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
  Handle eines k-NN-Klassifikators.
- ▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
  Namen der generischen Parameter des k-NN-Klassifikators.
  Defaultwert: ['method','k','max_num_classes']
  Werteliste: `GenParamName` ∈ {'method', 'num_checks', 'epsilon', 'k', 'max_num_classes'}
- ▷ **GenParamValue** (input_control) . . . . . number-array ⇝ *integer* / real / string
  Werte der generischen Parameter des k-NN-Klassifikators.
  Defaultwert: ['classes_distance',5,1]
  Wertevorschläge: `GenParamValue` ∈ {'classes_distance', 'classes_frequency', 'classes_weighted_frequencies', 'neighbors_distance', 32, 0.0, 0.02, 0, 1, 2, 3, 4, 5, 6}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_params_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- KNNHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`train_class_knn`, `read_class_knn`

Nachfolger

`classify_class_knn`

Siehe auch

`create_class_knn`, `read_class_knn`, `get_params_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

`train_class_knn` `( : : KNNHandle, GenParamName, GenParamValue : )`

Erstellt die Suchbäume eines k-NN-Klassifikators.

`train_class_knn` erstellt die Suchbäume eines k-Nearest-Neighbor-Klassifikators.

Die Anzahl der Suchbäume kann in den generischen Parametern `GenParamName` und `GenParamValue` über *'num_trees'* festgelegt werden. Der Standardwert ist hierbei 4, welcher üblicherweise zu ausreichend guten Ergebnissen führt. Eine höhere Anzahl der Bäume führt zu längeren Laufzeiten und zu höherem Speicherverbrauch, aber zu tendenziell genaueren Ergebnissen, speziell bei höheren Dimensionen der Merkmalsvektoren.

Nach einem erfolgten Training können weitere Daten mit dem Operator `add_sample_class_knn` hinzugefügt werden. Die neu hinzugefügten Daten beeinflussen die Klassifikation erst, wenn `train_class_knn` erneut aufgerufen wurde.

Die automatische Merkmals-Normalisierung kann aktiviert werden, indem der Eingabeparameter `GenParamName` den Wert *'normalization'* und `GenParamValue` an gleicher Stelle *'true'* enthält. Die Merkmalsvektoren werden in jeder Dimension einzeln normalisiert. Für jede Dimension wird der Mittelwert und die Standardabweichung anhand der übermittelten Trainingsdaten ermittelt. Alle Merkmalsvektoren werden dann dadurch normalisiert, dass in jeder Dimension der Mittelwert abgezogen und durch die Standardabweichung geteilt wird. Solch eine Normalisierung bewirkt, dass die Merkmalsvektoren einen Mittelwert von Null und eine Varianz von 1 nach der Normalisierung besitzen. Falls die Standardabweichung in einer Dimension den Wert Null aufweisen sollte, so wird nur der Mittelwert abgezogen und nicht durch Null dividiert. Man beachte, dass eine Standardabweichung von Null in einer Dimension bedeutet, dass diese Dimension zu dem Klassifikator keine relevante Information beisteuert und man diese Dimension weglassen könnte. Die Automatische Merkmals-Normalisierung bewirkt, dass die gespeicherten Trainingsdaten verändert werden. Diese Veränderungen lassen sich aber ohne Verluste rückgängig machen, indem der Operator `train_class_knn` erneut aufgerufen wird, wobei 'normalization' auf 'false' gesetzt werden muss. Falls Normalisierung eingesetzt wird, interpretiert der Operator `classify_class_knn` die ihm übergebenen Merkmalsvektoren als nicht normalisiert und es wird intern eine Normalisierung durchgeführt, die durch den letzten Aufruf von `train_class_knn` definiert ist.

Parameter

▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
Handle des k-NN-Klassifikators.

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der generischen Parameter, die für das Training des k-NN-Klassifikators angepasst werden können.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'num_trees', 'normalization'}

▷ **GenParamValue** (input_control) . . . . . number-array ⇝ *integer* / string / real
Werte der generischen Parameter, die für das Training des k-NN-Klassifikators angepasst werden können.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {4, 'false', 'true'}

Ergebnis

Sind die Eingabeparameter korrekt, dann liefert `train_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- KNNHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`add_sample_class_knn`, `read_class_knn`

Alternativen

`select_feature_set_knn`

Siehe auch

`create_class_knn`, `read_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

write_class_knn (: : KNNHandle, FileName :)

Schreibt einen trainierten k-NN-Klassifikator in eine Datei.

`write_class_knn` schreibt den k-Nearest-Neighbor-Klassifikator (k-NN) gegeben durch `KNNHandle` in die Datei `FileName`. Der Klassifikator kann mit `read_class_knn` wieder gelesen werden. Da die Samples inhärenter Bestandteil eines k-NN-Klassifikators sind, werden sie durch den Operator `write_class_knn` mit abgespeichert. Im Gegensatz zu anderen Klassifikatoren wie z.B. SVM gibt es daher keinen Operator, um die Samples separat zu speichern. Die Samples können mit dem Operator `get_sample_class_knn` aus dem k-NN-Klassifikator extrahiert werden. Die Dateiendung für den k-NN-Klassifikator ist in HALCON 'gnc'.

Parameter

▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
Handle eines k-NN-Klassifikators.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name einer Datei.
Dateiendung: `.gnc`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_knn`, `read_class_knn`

Siehe auch

`create_class_knn`, `read_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

15.3 Look-Up-Tabelle

`clear_class_lut` `( : : ClassLUTHandle : )`

Löschen eines Look-Up-Table Klassifikators.

`clear_class_lut` löscht die Look-Up-Table (LUT), die durch `ClassLUTHandle` angegeben wird, und gibt sämtlichen für die LUT benötigten Speicher frei. Der LUT Klassifikator kann nach dem Aufruf von `clear_class_lut` nicht mehr verwendet werden. Der Handle `ClassLUTHandle` ist nach dem Aufruf ungültig.

Parameter

▷ **`ClassLUTHandle`** (input_control) ..... class_lut(-array) ⇝ *handle*
Handle des LUT Klassifikators.

Ergebnis

Falls `ClassLUTHandle` korrekt ist, dann liefert `clear_class_lut` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`classify_image_class_lut`

Siehe auch

`create_class_lut_mlp`, `create_class_lut_svm`, `create_class_lut_gmm`

Modul

Foundation

`create_class_lut_gmm` `( : : GMMHandle, GenParamName, GenParamValue : ClassLUTHandle )`

Erzeugen einer Look-Up-Table anhand eines Gaußschen Mischverteilungsmodells zur Klassifizierung von Byte-Bildern.

create_class_lut_gmm erzeugt eine Look-Up Table (LUT) ClassLUTHandle anhand eines trainierten Gaußschen Mischverteilungsmodells (GMM) GMMHandle zur Klassifizierung von mehrkanaligen Byte-Bildern. Mit diesem GMM-basierten LUT Klassifikator kann der Operator classify_image_class_gmm, der für die spätere Klassifikation benötigt wird, durch den Operator classify_image_class_lut ersetzt werden. Die Klassifikation wird dadurch deutlich beschleunigt, weil die entsprechende Klasse nicht in jedem Bildpunkt extra berechnet werden muss, da alle möglichen Antworten des GMM in der LUT gespeichert sind. Für die Erzeugung der LUT sind die im zuvor aufgerufenen Operator create_class_gmm definierten Parameter NumDim, Preprocessing und NumComponents von Bedeutung. In NumDim ist festgelegt, wie viele Kanäle die zu klassifizierenden Bilder haben dürfen. Über das Preprocessing (siehe create_class_gmm) kann diese Anzahl auf NumComponents transformiert werden. NumComponents beschreibt schließlich die Größe des Merkmalsvektors, der vom Klassifikator classify_class_gmm intern verarbeitet wird. Aufgrund von Performance und Speicherverwaltung ist die LUT auf maximal *3* Dimensionen beschränkt. Da sie den Operator classify_class_gmm ersetzt, muss NumComponents $\leq$ *3* gelten. Findet keine Transformation mit Reduktion in den Bildkanälen statt (NumDim = NumComponents) werden beim Erzeugen der LUT alle möglichen Pixelwerte, die in einem Byte-Bild vorkommen können, mit classify_class_gmm klassifiziert und die resultierenden Klassen in der LUT gespeichert. Findet eine Transformation mit Reduktion der Bildkanäle statt (NumDim >NumComponents) werden die Vorverarbeitungsparameter des GMM in einer separaten Struktur der LUT gespeichert. Beim Erzeugen der LUT werden alle transformierten Pixelwerte mit classify_class_gmm klassifiziert und die resultierenden Klassen in der LUT gespeichert. Aufgrund der diskreten Struktur der LUT kann es dabei zu einem Genauigkeitsverlust gegenüber der Klassifizierung mit classify_image_class_gmm kommen. Über die Parameter *'bit_depth'* und *'class_selection'* kann die Klassifizierungsgenauigkeit, der Speicherbedarf und die Laufzeit zur Erzeugung der LUT angepasst werden.

Die folgenden Parameter des GMM-basierten LUT Klassifikators können mit GenParamName und GenParamValue gesetzt werden:

'bit_depth': Anzahl der verwendeten Bits der Bildpixel. *'bit_depth'* steuert den Speicherbedarf der LUT und ist durch die Bittiefe des Bildes beschränkt (*'bit_depth'* $\leq$ *8*). Ist die Bittiefe der LUT kleiner (*'bit_depth'* $<$ *8*), werden die Klassen von mehreren Pixelkombinationen auf den selben Eintrag der LUT abgebildet, was die Klassifizierungsgenauigkeit verringern kann. Einer dieser Cluster enthält $2^{NumComponents \cdot (8-bit_depth)}$ Pixelkombinationen, wobei NumComponents die in create_class_gmm spezifizierte Dimension der LUT beschreibt. Für *'bit_depth'* $=$ *7*, NumComponents $=$ *3* werden beispielsweise die Klassen von *8* Pixelkombinationen auf den selben Eintrag der LUT abgebildet. Die LUT benötigt maximal $2^{NumComponents \cdot bit_depth+2}$ Bytes vom Speicher. Für NumComponents $=$ *3*, *'bit_depth'* $=$ *8* und NumClasses $<$ *16* (angegeben in create_class_gmm) werden beispielsweise durch eine interne Speicheroptimierung nur 8 MB Speicher für die LUT benötigt. Gilt NumClasses $=$ *1*, benötigt die LUT bei voller Bittiefe nur 2 MB Speicher. Die Laufzeit der Klassifizierung in classify_image_class_lut wird minimal, falls die LUT in den Cache passt. **Defaultwert** ist *8*, typische Werte sind [*6*,*7*,*8*]. Beschränkungen: *'bit_depth'* $\geq$ *1*, *'bit_depth'* $\leq$ *8*.

'class_selection': Methode zur Klassenauswahl für die LUT. Kann angepasst werden um die spätere Klassifizierungsgenauigkeit und die Laufzeit zur Erzeugung der LUT zu steuern. Der Wert für *'class_selection'* wird ignoriert, falls die Bittiefe der LUT maximal ist, also *'bit_depth'* $=$ *8* gilt. Ist die Bittiefe der LUT kleiner (*'bit_depth'* $<$ *8*), werden die Klassen von mehreren Pixelkombinationen auf den selben Eintrag der LUT abgebildet. Einer dieser Cluster enthält $2^{NumComponents \cdot (8-bit_depth)}$ Pixelkombinationen, wobei NumComponents die in create_class_gmm spezifizierte Dimension der LUT beschreibt. Gilt *'class_selection'* $=$ *'best'*, wird die Klasse gespeichert, die am häufigsten im Cluster vorkommt. Für *'class_selection'* $=$ *'fast'* wird nur der Pixel im Cluster klassifiziert, der den kleinsten Wert (komponentenweise) hat, was jedoch die Genauigkeit bei der späteren Klassifizierung über die LUT verringern kann. Allerdings lässt sich dadurch die Laufzeit zur Erzeugung der LUT reduzieren. Sie verhält sich proportional zum maximal benötigten Speicherplatz der LUT, der mit $2^{NumComponents \cdot bit_depth+2}$ definiert ist. **Defaultwert** ist *'fast'*, mögliche Werte sind [*'fast'*, *'best'*].

'rejection_threshold': Schwellenwert zur Zurückweisung unsicher klassifizierter Punkte des GMM. Der Parameter stellt einen Schwellenwert auf dem von der Klassifikation zurückgelieferten K-Sigma-Wahrscheinlichkeitsmaß dar (siehe classify_class_gmm und evaluate_class_gmm). Alle Pixel mit einer Wahrscheinlichkeit unterhalb von *'rejection_threshold'* werden keiner Klasse zugeordnet. **Defaultwert** ist *0.0001*. Beschränkungen: *'rejection_threshold'* $\geq$ *0*, *'rejection_threshold'* $\leq$ *1*.

Parameter

- ▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
  Handle des GMM.
- ▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
  Namen der generischen Parameter, die für die Erzeugung des LUT Klassifikators angepasst werden können.
  Defaultwert: []
  Wertevorschläge: `GenParamName` ∈ {'bit_depth', 'class_selection', 'rejection_threshold'}
- ▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *string* / integer / real
  Werte der generischen Parameter, die für die Erzeugung des LUT Klassifikators angepasst werden können.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {8, 7, 6, 'fast', 'best'}
- ▷ **ClassLUTHandle** (output_control) ..... class_lut ⇝ *handle*
  Handle des LUT Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_class_lut_gmm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`train_class_gmm`, `read_class_gmm`

Nachfolger

`classify_image_class_lut`

Alternativen

`create_class_lut_knn`, `create_class_lut_mlp`, `create_class_lut_svm`

Siehe auch

`classify_image_class_lut`, `clear_class_lut`

Modul

Foundation

create_class_lut_knn (: : KNNHandle, GenParamName,
GenParamValue : ClassLUTHandle)

Erzeugen einer Look-Up Tabelle anhand eines k-Nearest-Neighbor-Klassifikators zur Klassifizierung von Byte-Bildern.

`create_class_lut_knn` erzeugt eine Look-Up Tabelle (LUT) `ClassLUTHandle` anhand eines trainierten k-Nearest-Neighbor-Klassifikators (k-NN) `KNNHandle` zur Klassifizierung von mehrkanaligen Byte-Bildern. Mit diesem k-NN-basierten LUT Klassifikator kann der Operator `classify_image_class_knn`, der für die spätere Klassifikation benötigt wird, durch den Operator `classify_image_class_lut` ersetzt werden. Die Klassifikation wird dadurch deutlich beschleunigt, weil die entsprechende Klasse nicht in jedem Bildpunkt extra berechnet werden muss, da alle möglichen Antworten des k-NN in der LUT gespeichert sind. Für die Erzeugung der LUT sind der im zuvor aufgerufenen Operator `create_class_knn` definierte Parameter `NumDim`, von Bedeutung. In `NumDim` ist festgelegt, wie viele Kanäle die zu klassifizierenden Bilder haben dürfen. Beim Erzeugen der LUT werden alle Pixelwerte mit `classify_class_knn` klassifiziert und die resultierenden Klassen in der LUT gespeichert. Aufgrund der diskreten Struktur der LUT kann es dabei zu einem Genauigkeitsverlust gegenüber der Klassifizierung mit `classify_image_class_knn` kommen. Über den Parameter *'bit_depth'* kann die Klassifizierungsgenauigkeit, der Speicherbedarf und die Laufzeit zur Erzeugung der LUT angepasst werden.

Die folgenden Parameter des k-NN-basierten LUT Klassifikators können mit GenParamName und GenParamValue gesetzt werden:

'bit_depth': Anzahl der verwendeten Bits der Bildpixel. *'bit_depth'* steuert den Speicherbedarf der LUT und ist durch die Bittiefe des Bildes beschränkt (*'bit_depth'* $\leq$ *8*). Ist die Bittiefe der LUT kleiner (*'bit_depth'* $<$ *8*), werden die Klassen von mehreren Pixelkombinationen auf den selben Eintrag der LUT abgebildet, was die Klassifizierungsgenauigkeit verringern kann. Einer dieser Cluster enthält $2^{NumDim \cdot (8-bit_depth)}$ Pixelkombinationen, wobei NumDim die in create_class_knn spezifizierte Dimension der LUT beschreibt. Für *'bit_depth'* $=$ *7*, NumDim $=$ *3* werden beispielsweise die Klassen von *8* Pixelkombinationen auf den selben Eintrag der LUT abgebildet. Die LUT benötigt maximal $2^{NumDim \cdot bit_depth+2}$ Bytes vom Speicher. Für NumDim $=$ *3*, *'bit_depth'* $=$ *8* werden beispielsweise durch eine interne Speicheroptimierung ca. 8 MB Speicher für die LUT benötigt. Die Laufzeit der Klassifizierung in classify_image_class_lut wird minimal, falls die LUT in den Cache passt. **Defaultwert** ist *8*, typische Werte sind [*6*,*7*,*8*]. Beschränkungen: *'bit_depth'* $\geq$ *1*, *'bit_depth'* $\leq$ *8*.

'rejection_threshold': Schwellenwert zur Zurückweisung unsicher klassifizierter Punkte des kNN. Der Parameter stellt einen Schwellenwert auf dem von der Klassifikation zurückgelieferten Distanzwert dar (siehe classify_class_knn). Alle Pixel mit einer Distanz oberhalb von *'rejection_threshold'* werden keiner Klasse zugeordnet. **Defaultwert** ist *5*. Beschränkungen: *'rejection_threshold'* $\geq$ *0*.

Parameter

- ▷ **KNNHandle** (input_control) . . . . . class_knn ⇝ *handle*
  Handle des k-NN-Klassifikators.
- ▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
  Namen der generischen Parameter, die für die Erzeugung des LUT Klassifikators angepasst werden können.
  Defaultwert: []
  Wertevorschläge: GenParamName ∈ {'bit_depth', 'rejection_threshold'}
- ▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *string* / integer / real
  Werte der generischen Parameter, die für die Erzeugung des LUT Klassifikators angepasst werden können.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {8, 7, 6, 0.5, 5, 10, 50}
- ▷ **ClassLUTHandle** (output_control) . . . . . class_lut ⇝ *handle*
  Handle des LUT Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_class_lut_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

train_class_knn, read_class_knn

Nachfolger

classify_image_class_lut

Alternativen

create_class_lut_svm, create_class_lut_gmm, create_class_lut_mlp

Siehe auch

classify_image_class_lut, clear_class_lut

Modul

Foundation

create_class_lut_mlp (: : MLPHandle, GenParamName, GenParamValue : ClassLUTHandle)

Erzeugen einer Look-Up-Table anhand eines mehrschichtigen Perzeptrons zur Klassifizierung von Byte-Bildern.

create_class_lut_mlp erzeugt eine Look-Up Table (LUT) ClassLUTHandle anhand eines trainierten mehrschichtigen Perzeptrons (MLP) MLPHandle zur Klassifizierung von mehrkanaligen Byte-Bildern. Mit diesem MLP-basierten LUT Klassifikator kann der Operator classify_image_class_mlp, der für die spätere Klassifikation benötigt wird, durch den Operator classify_image_class_lut ersetzt werden. Die Klassifikation wird dadurch deutlich beschleunigt, weil die entsprechende Klasse nicht in jedem Bildpunkt extra berechnet werden muss, da alle möglichen Antworten des MLP in der LUT gespeichert sind. Für die Erzeugung der LUT sind die im zuvor aufgerufenen Operator create_class_mlp definierten Parameter NumInput, Preprocessing und NumComponents von Bedeutung. In NumInput ist festgelegt, wie viele Kanäle die zu klassifizierenden Bilder haben dürfen. Über das Preprocessing (siehe create_class_mlp) kann diese Anzahl auf NumComponents transformiert werden. NumComponents beschreibt schließlich die Größe des Merkmalsvektors, der vom Klassifikator classify_class_mlp intern verarbeitet wird. Aufgrund von Performance und Speicherverwaltung ist die LUT auf maximal *3* Dimensionen beschränkt. Da sie den Operator classify_class_mlp ersetzt, muss NumComponents $\leq$ *3* gelten. Findet keine Transformation mit Reduktion in den Bildkanälen statt (NumInput = NumComponents) werden beim Erzeugen der LUT alle möglichen Pixelwerte, die in einem Byte-Bild vorkommen können, mit classify_class_mlp klassifiziert und die resultierenden Klassen in der LUT gespeichert. Findet eine Transformation mit Reduktion der Bildkanäle statt (NumInput >NumComponents) werden die Vorverarbeitungsparameter des MLP in einer separaten Struktur der LUT gespeichert. Beim Erzeugen der LUT werden alle transformierten Pixelwerte mit classify_class_mlp klassifiziert und die resultierenden Klassen in der LUT gespeichert. Aufgrund der diskreten Struktur der LUT kann es dabei zu einem Genauigkeitsverlust gegenüber der Klassifizierung mit classify_image_class_mlp kommen. Über die Parameter *'bit_depth'* und *'class_selection'* kann die Klassifizierungsgenauigkeit, der Speicherbedarf und die Laufzeit zur Erzeugung der LUT angepasst werden.

Die folgenden Parameter des MLP-basierten LUT Klassifikators können mit GenParamName und GenParamValue gesetzt werden:

'bit_depth': Anzahl der verwendeten Bits der Bildpixel. *'bit_depth'* steuert den Speicherbedarf der LUT und ist durch die Bittiefe des Bildes beschränkt (*'bit_depth'* $\leq$ *8*). Ist die Bittiefe der LUT kleiner (*'bit_depth'* $<$ *8*), werden die Klassen von mehreren Pixelkombinationen auf den selben Eintrag der LUT abgebildet, was die Klassifizierungsgenauigkeit verringern kann. Einer dieser Cluster enthält $2^{NumComponents \cdot (8-bit_depth)}$ Pixelkombinationen, wobei NumComponents die in create_class_mlp spezifizierte Dimension der LUT beschreibt. Für *'bit_depth'* = *7*, NumComponents = *3* werden beispielsweise die Klassen von *8* Pixelkombinationen auf den selben Eintrag der LUT abgebildet. Die LUT benötigt maximal $2^{NumComponents \cdot bit_depth+2}$ Bytes vom Speicher. Für NumComponents = *3*, *'bit_depth'* = *8* und NumOutput $<$ *16* (angegeben in create_class_mlp) werden beispielsweise durch eine interne Speicheroptimierung nur 8 MB Speicher für die LUT benötigt. Gilt NumOutput = *1*, benötigt die LUT bei voller Bittiefe nur 2 MB Speicher. Die Laufzeit der Klassifizierung in classify_image_class_lut wird minimal, falls die LUT in den Cache passt. **Defaultwert** ist *8*, typische Werte sind [*6*,*7*,*8*]. Beschränkungen: *'bit_depth'* $\geq$ *1*, *'bit_depth'* $\leq$ *8*.

'class_selection': Methode zur Klassenauswahl für die LUT. Kann angepasst werden um die spätere Klassifizierungsgenauigkeit und die Laufzeit zur Erzeugung der LUT zu steuern. Der Wert für *'class_selection'* wird ignoriert, falls die Bittiefe der LUT maximal ist, also *'bit_depth'* = *8* gilt. Ist die Bittiefe der LUT kleiner (*'bit_depth'* $<$ *8*), werden die Klassen von mehreren Pixelkombinationen auf den selben Eintrag der LUT abgebildet. Einer dieser Cluster enthält $2^{NumComponents \cdot (8-bit_depth)}$ Pixelkombinationen, wobei NumComponents die in create_class_mlp spezifizierte Dimension der LUT beschreibt. Gilt *'class_selection'* = *'best'*, wird die Klasse gespeichert, die am häufigsten im Cluster vorkommt. Für *'class_selection'* = *'fast'* wird nur der Pixel im Cluster klassifiziert, der den kleinsten Wert (komponentenweise) hat, was jedoch die Genauigkeit bei der späteren Klassifizierung über die LUT verringern kann. Allerdings lässt sich dadurch die Laufzeit zur Erzeugung der LUT reduzieren. Sie verhält sich proportional zum maximal benötigten Speicherplatz der LUT, der mit $2^{NumComponents \cdot bit_depth+2}$ definiert ist. **Defaultwert** ist *'fast'*, mögliche Werte sind [*'fast'*, *'best'*].

'rejection_threshold': Schwellenwert zur Zurückweisung unsicher klassifizierter Punkte des MLP. Der Parameter stellt einen Schwellenwert auf dem von der Klassifikation zurückgelieferten Wahrscheinlichkeitsmaß dar (siehe classify_class_mlp und evaluate_class_mlp). Alle Pixel mit einer Wahrscheinlichkeit

unterhalb von *'rejection_threshold'* werden keiner Klasse zugeordnet. **Defaultwert** ist *0.5*. Beschränkungen: *'rejection_threshold'* $\geq$ *0*, *'rejection_threshold'* $\leq$ *1*.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter, die für die Erzeugung des LUT Klassifikators angepasst werden können.
Defaultwert: []
Wertevorschläge: `GenParamName` ∈ {'bit_depth', 'class_selection', 'rejection_threshold'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *string* / integer / real
Werte der generischen Parameter, die für die Erzeugung des LUT Klassifikators angepasst werden können.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {8, 7, 6, 'fast', 'best'}

▷ **ClassLUTHandle** (output_control) . . . . . class_lut ⇝ *handle*
Handle des LUT Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_class_lut_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`train_class_mlp`, `read_class_mlp`

Nachfolger

`classify_image_class_lut`

Alternativen

`create_class_lut_gmm`, `create_class_lut_knn`, `create_class_lut_svm`

Siehe auch

`classify_image_class_lut`, `clear_class_lut`

Modul

Foundation

create_class_lut_svm (: : SVMHandle, GenParamName, GenParamValue : ClassLUTHandle)

Erzeugen einer Look-Up-Table anhand einer Support-Vektor-Maschine zur Klassifizierung von Byte-Bildern.

`create_class_lut_svm` erzeugt eine Look-Up Table (LUT) `ClassLUTHandle` anhand einer trainierten Support-Vector-Maschine (SVM) `SVMHandle` zur Klassifizierung von mehrkanaligen Byte-Bildern. Mit diesem SVM-basierten LUT Klassifikator kann der Operator `classify_image_class_svm`, der für die spätere Klassifikation benötigt wird, durch den Operator `classify_image_class_lut` ersetzt werden. Die Klassifikation wird dadurch deutlich beschleunigt, weil die entsprechende Klasse nicht in jedem Bildpunkt extra berechnet werden muss, da alle möglichen Antworten der SVM in der LUT gespeichert sind. Für die Erzeugung der LUT sind die im zuvor aufgerufenen Operator `create_class_svm` definierten Parameter `NumFeatures`, `Preprocessing` und `NumComponents` von Bedeutung. In `NumFeatures` ist festgelegt, wie viele Kanäle die zu klassifizierenden Bilder haben dürfen. Über das `Preprocessing` (siehe `create_class_svm`) kann diese Anzahl auf `NumComponents` transformiert werden. `NumComponents` beschreibt schließlich die Größe

des Merkmalsvektors, der vom Klassifikator classify_class_svm intern verarbeitet wird. Aufgrund von Performance und Speicherverwaltung ist die LUT auf maximal *3* Dimensionen beschränkt. Da sie den Operator classify_class_svm ersetzt, muss NumComponents $\leq$ *3* gelten. Findet keine Transformation mit Reduktion in den Bildkanälen statt (NumFeatures = NumComponents) werden beim Erzeugen der LUT alle möglichen Pixelwerte, die in einem Byte-Bild vorkommen können, mit classify_class_svm klassifiziert und die resultierenden Klassen in der LUT gespeichert. Findet eine Transformation mit Reduktion der Bildkanäle statt (NumFeatures >NumComponents) werden die Vorverarbeitungsparameter der SVM in einer separaten Struktur der LUT gespeichert. Beim Erzeugen der LUT werden alle transformierten Pixelwerte mit classify_class_svm klassifiziert und die resultierenden Klassen in der LUT gespeichert. Aufgrund der diskreten Struktur der LUT kann es dabei zu einem Genauigkeitsverlust gegenüber der Klassifizierung mit classify_image_class_svm kommen. Über die Parameter *'bit_depth'* und *'class_selection'* kann die Klassifizierungsgenauigkeit, der Speicherbedarf und die Laufzeit zur Erzeugung der LUT angepasst werden.

Die folgenden Parameter des SVM-basierten LUT Klassifikators können mit GenParamName und GenParamValue gesetzt werden:

'bit_depth': Anzahl der verwendeten Bits der Bildpixel. *'bit_depth'* steuert den Speicherbedarf der LUT und ist durch die Bittiefe des Bildes beschränkt (*'bit_depth'* $\leq$ *8*). Ist die Bittiefe der LUT kleiner (*'bit_depth'* $<$ *8*), werden die Klassen von mehreren Pixelkombinationen auf den selben Eintrag der LUT abgebildet, was die Klassifizierungsgenauigkeit verringern kann. Einer dieser Cluster enthält $2^{NumComponents \cdot (8-bit_depth)}$ Pixelkombinationen, wobei NumComponents die in create_class_svm spezifizierte Dimension der LUT beschreibt. Für *'bit_depth'* = *7*, NumComponents = *3* werden beispielsweise die Klassen von *8* Pixelkombinationen auf den selben Eintrag der LUT abgebildet. Die LUT benötigt maximal $2^{NumComponents \cdot bit_depth+2}$ Bytes vom Speicher. Für NumComponents = *3*, *'bit_depth'* = *8* und NumClasses $<$ *16* (angegeben in create_class_svm) werden beispielsweise durch eine interne Speicheroptimierung nur 8 MB Speicher für die LUT benötigt. Gilt NumClasses = *1*, benötigt die LUT bei voller Bittiefe nur 2 MB Speicher. Die Laufzeit der Klassifizierung in classify_image_class_lut wird minimal, falls die LUT in den Cache passt. **Defaultwert** ist *8*, typische Werte sind [*6*,*7*,*8*]. Beschränkungen: *'bit_depth'* $\geq$ *1*, *'bit_depth'* $\leq$ *8*.

'class_selection': Methode zur Klassenauswahl für die LUT. Kann angepasst werden um die spätere Klassifizierungsgenauigkeit und die Laufzeit zur Erzeugung der LUT zu steuern. Der Wert für *'class_selection'* wird ignoriert, falls die Bittiefe der LUT maximal ist, also *'bit_depth'* = *8* gilt. Ist die Bittiefe der LUT kleiner (*'bit_depth'* $<$ *8*), werden die Klassen von mehreren Pixelkombinationen auf den selben Eintrag der LUT abgebildet. Einer dieser Cluster enthält $2^{NumComponents \cdot (8-bit_depth)}$ Pixelkombinationen, wobei NumComponents die in create_class_gmm spezifizierte Dimension der LUT beschreibt. Gilt *'class_selection'* = *'best'*, wird die Klasse gespeichert, die am häufigsten im Cluster vorkommt. Für *'class_selection'* = *'fast'* wird nur der Pixel im Cluster klassifiziert, der den kleinsten Wert (komponentenweise) hat, was jedoch die Genauigkeit bei der späteren Klassifizierung über die LUT verringern kann. Allerdings lässt sich dadurch die Laufzeit zur Erzeugung der LUT reduzieren. Sie verhält sich proportional zum maximal benötigten Speicherplatz der LUT, der mit $2^{NumComponents \cdot bit_depth+2}$ definiert ist. **Defaultwert** ist *'fast'*, mögliche Werte sind [*'fast'*, *'best'*].

Parameter

▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
Handle der SVM.

▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
Namen der generischen Parameter, die für die Erzeugung des LUT Klassifikators angepasst werden können.
Defaultwert: []
Wertevorschläge: GenParamName ∈ {'bit_depth', 'class_selection'}

▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *string* / integer
Werte der generischen Parameter, die für die Erzeugung des LUT Klassifikators angepasst werden können.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {8, 7, 6, 'fast', 'best'}

▷ **ClassLUTHandle** (output_control) ..... class_lut ⇝ *handle*
Handle des LUT Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_class_lut_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`train_class_svm`, `read_class_svm`

Nachfolger

`classify_image_class_lut`

Alternativen

`create_class_lut_gmm`, `create_class_lut_knn`, `create_class_lut_mlp`

Siehe auch

`classify_image_class_lut`, `clear_class_lut`

Modul

Foundation

15.4 Neuronale Netze

`add_class_train_data_mlp` (: : `MLPHandle`, `ClassTrainDataHandle` :)

Hinzufügen von Trainingsdaten zu einem mehrschichtigen Perzeptron (MLP).

`add_class_train_data_mlp` fügt die Trainingsdaten `ClassTrainDataHandle` zu einem mehrschichtigen Perzeptron (MLP), gegeben durch `MLPHandle`, hinzu.

Parameter

▷ **`MLPHandle`** (input_control) . . . . . class_mlp ⇝ *handle*
Handle eines MLPs.

▷ **`ClassTrainDataHandle`** (input_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten für den Klassifikator.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_class_train_data_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_mlp`, `create_class_train_data`

Nachfolger

`get_sample_class_mlp`

Alternativen

add_sample_class_mlp

Siehe auch

create_class_mlp

Modul

Foundation

add_sample_class_mlp (: : MLPHandle, Features, Target :)

Hinzufügen eines Trainingsmusters zu den Trainingsdaten eines mehrschichtigen Perzeptrons.

add_sample_class_mlp fügt ein Trainingsmuster zu dem durch MLPHandle gegebenen mehrschichtigen Perzeptron (MLP) hinzu. Das Trainingsmuster wird durch Features und Target beschrieben. Features ist der Merkmalsvektor des Musters und muss folglich die Länge NumInput besitzen, die in create_class_mlp angegeben worden ist. Target ist der Zielvektor des Musters, der für alle drei Aktivierungsfunktionen in der Ausgabeschicht des MLP (siehe create_class_mlp) die Länge NumOutput besitzen muss und aus reellen Zahlen bestehen muss (Ausnahme: s.u.). Falls das MLP zur Regression (Funktionsapproximation) verwendet wird (OutputFunction = *'linear'*), ist Target der Funktionswert der Funktion an der Koordinate Features. In diesem Fall kann Target beliebige reelle Zahlen beinhalten. Für OutputFunction = *'logistic'* können in Target nur die Werte *0.0* und *1.0* angegeben werden. Ein Wert von *1.0* spezifiziert das Vorhandensein des jeweiligen Attributes, während *0.0* die Abwesenheit des Attributes spezifiziert. Da in diesem Fall die Attribute unabhängig voneinander anwesend oder abwesend sein können, können beliebige Kombinationen von *0.0* und *1.0* übergeben werden. Für OutputFunction = *'softmax'* dürfen die Werte von Target auch nur *0.0* und *1.0* enthalten. Im Gegensatz zu OutputFunction = *'logistic'* muss der Wert *1.0* an genau einer Stelle des Tupels Target vorkommen. Die Stelle des Tupels bezeichnet die Klasse des Musters. Zur Vereinfachung kann für OutputFunction = *'softmax'* auch ein einzelner ganzzahliger Wert übergeben werden, der direkt die Klasse des Musters bezeichnet. Hierbei werden die Klassen von 0 ab gezählt, d.h. die Klasse muss ein Wert zwischen 0 und NumOutput − 1 sein. Intern wird die Klasse in einen Zielvektor der Länge NumOutput konvertiert.

Bevor das MLP mit train_class_mlp trainiert werden kann, müssen *alle* Trainingsmuster mit add_sample_class_mlp zum MLP hinzugefügt werden.

Die Anzahl der abgespeicherten Trainingsmuster kann mit get_sample_num_class_mlp abgefragt werden. Einzelne Trainingsmuster können mit get_sample_class_mlp wieder ausgelesen werden.

Im Normalfall ist es günstig, die Trainingsmuster mit write_samples_class_mlp in eine Datei zu speichern, damit sie wiederverwendet werden können, und damit bei Bedarf neue Trainingsmuster zum Datensatz hinzugefügt werden können, und ein *neu erzeugtes* MLP mit dem erweiterten Datensatz *neu* trainiert werden kann.

Parameter

- ▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
  Handle des MLP.
- ▷ **Features** (input_control) . . . . . real-array ⇝ *real*
  Merkmalsvektor des zu speichernden Trainingsmusters.
- ▷ **Target** (input_control) . . . . . number(-array) ⇝ *integer* / real
  Klasse oder Zielvektor des zu speichernden Trainingsmusters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_sample_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_mlp`

Nachfolger

`train_class_mlp`, `write_samples_class_mlp`

Alternativen

`read_samples_class_mlp`

Siehe auch

`clear_samples_class_mlp`, `get_sample_num_class_mlp`, `get_sample_class_mlp`

Modul

Foundation

`classify_class_mlp` (: : `MLPHandle`, `Features`, `Num` : `Class`, `Confidence`)

Berechnen der Klasse eines Merkmalsvektors durch ein mehrschichtiges Perzeptron.

`classify_class_mlp` berechnet mit dem durch `MLPHandle` gegebenen mehrschichtigen Perzeptron (MLP) die `Num` besten Klassen des Merkmalsvektors `Features` und gibt die Klassen in `Class` und die zugehörigen Konfidenzen (Wahrscheinlichkeiten) der Klassen in `Confidence` zurück. Das MLP muss vor der Verwendung von `classify_class_mlp` mit `train_class_mlp` trainiert werden.

`classify_class_mlp` kann nur aufgerufen werden, falls das MLP als Klassifikator mit `OutputFunction` = *'softmax'* verwendet wird (siehe `create_class_mlp`). Ansonsten wird eine Fehlermeldung zurückgeliefert. `classify_class_mlp` entspricht dem Aufruf von `evaluate_class_mlp` und der zusätzlichen Bestimmung der besten `Num` Klassen. Wie bei `evaluate_class_mlp` beschrieben, können die Ausgabewerte des MLP als Wahrscheinlichkeiten des Auftretens der jeweiligen Klasse interpretiert werden. Hier wird aber die a-posteriori-Wahrscheinlichkeit `ClassProb` weiter als `ClassProb` $= p(i|x)/p(x)$ normalisiert, wobei $p(i|x)$ und $p(x)$ wie bei `evaluate_class_gmm` definiert sind. Im Normalfall, sollte es ausreichend sein, `Num` = *1* zu verwenden, um zu entscheiden, ob die Wahrscheinlichkeit der besten Klasse des Merkmalsvektors hoch genug ist. In manchen Anwendungen kann es aber auch interessant sein, die zweitbeste Klasse zu betrachten (`Num` = *2*), insbesondere, wenn zu erwarten ist, dass sich die Klassen signifikant überlappen.

Parameter

- ▷ **`MLPHandle`** (input_control) ..... class_mlp ⇝ *handle*
  Handle des MLP.
- ▷ **`Features`** (input_control) ..... real-array ⇝ *real*
  Merkmalsvektor.
- ▷ **`Num`** (input_control) ..... integer-array ⇝ *integer*
  Anzahl der zu bestimmenden besten Klassen.
  Defaultwert: 1
  Wertevorschläge: `Num` ∈ {1, 2, 3, 4, 5}
- ▷ **`Class`** (output_control) ..... integer(-array) ⇝ *integer*
  Ergebnis der Klassifikation des Merkmalsvektors durch das MLP.
- ▷ **`Confidence`** (output_control) ..... real(-array) ⇝ *real*
  Konfidenz(en) der Klasse(n) des Merkmalsvektors.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `classify_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_class_mlp, read_class_mlp

Alternativen

apply_dl_classifier, evaluate_class_mlp

Siehe auch

create_class_mlp

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Andrew Webb: „Statistical Pattern Recognition“; Arnold, London; 1999.

Modul

Foundation

clear_class_mlp (: : MLPHandle :)

Löschen eines mehrschichtigen Perzeptrons.

clear_class_mlp löscht das mehrschichtige Perzeptron (MLP), das durch MLPHandle angegeben wird, und gibt sämtlichen für das MLP benötigten Speicher frei. Das MLP kann nach dem Aufruf von clear_class_mlp nicht mehr verwendet werden. Der Handle MLPHandle ist nach dem Aufruf ungültig.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp(-array) ⇝ *handle*
Handle des MLP.

Ergebnis

Falls MLPHandle korrekt ist, dann liefert clear_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

classify_class_mlp, evaluate_class_mlp

Siehe auch

create_class_mlp, read_class_mlp, write_class_mlp, train_class_mlp

Modul

Foundation

clear_samples_class_mlp (: : MLPHandle :)

Löschen aller Trainingsdaten eines mehrschichtigen Perzeptrons.

clear_samples_class_mlp löscht alle mit add_sample_class_mlp oder read_samples_class_mlp zu dem mehrschichtigen Perzeptron (MLP) MLPHandle hinzugefügten Trainingsmuster. Eine Verwendung von clear_samples_class_mlp ist nur dann sinnvoll, wenn das MLP in demselben Prozess trainiert wird, in dem das MLP auch zur Evaluierung mit evaluate_class_mlp

oder zur Klassifikation mit classify_class_mlp verwendet wird. In diesem Fall kann der durch die Trainingsmuster belegte Speicher mit clear_samples_class_mlp wieder freigegeben werden, und somit Speicherplatz gespart werden. In der üblichen Verwendungsart, in der das MLP offline trainiert wird und mit write_class_mlp gespeichert wird, ist die Verwendung von clear_samples_class_mlp normalerweise überflüssig, da write_class_mlp die Trainingsmuster nicht abspeichert, und somit im Online-Prozess, der das MLP mit read_class_mlp einliest, auch kein Speicher für die Trainingsmuster benötigt wird.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp(-array) ⇝ *handle*
Handle des MLP.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert clear_samples_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

train_class_mlp, write_samples_class_mlp

Siehe auch

create_class_mlp, clear_class_mlp, add_sample_class_mlp, read_samples_class_mlp

Modul

Foundation

```
create_class_mlp ( : : NumInput, NumHidden, NumOutput,
    OutputFunction, Preprocessing, NumComponents,
    RandSeed : MLPHandle )
```

Erzeugen eines mehrschichtigen Perzeptrons zur Klassifikation oder Regression.

create_class_mlp erzeugt ein neuronales Netz in Form eines mehrschichtigen Perzeptrons (engl.: multilayer perceptron, MLP), welches zur Klassifikation oder Regression (Funktionsapproximation) verwendet werden kann, je nachdem, wie OutputFunction gesetzt wird. Das MLP besitzt drei Schichten: eine Eingabeschicht mit NumInput Eingabevariablen (Einheiten, Neuronen), eine versteckte Schicht mit NumHidden Einheiten und eine Ausgabeschicht mit NumOutput Ausgabevariablen. Das MLP führt mit den Eingabedaten x_i, dem so genannten Merkmalsvektor, folgende Berechnungsschritte aus, um die Aktivierungen z_j der versteckten Schicht zu berechnen:

$$\begin{aligned} a_j^{(1)} &= \sum_{i=1}^{n_i} w_{ji}^{(1)} x_i + b_j^{(1)}, \qquad j = 1, \ldots, n_h \\ z_j &= \tanh\left(a_j^{(1)}\right), \qquad j = 1, \ldots, n_h \end{aligned}$$

Die Matrix $w_{ji}^{(1)}$ und der Vektor $b_j^{(1)}$ sind dabei die Gewichte der Eingabeschicht (ersten Schicht) des MLP. Die Aktivierungen z_j der ersten Schicht werden in der versteckten Schicht (zweiten Schicht) zunächst genauso durch Linearkombinationen transformiert, wie in der ersten Schicht:

$$a_k^{(2)} = \sum_{j=1}^{n_h} w_{kj}^{(2)} z_j + b_k^{(2)}, \qquad k = 1, \ldots, n_o$$

Dabei sind die Matrix $w_{kj}^{(2)}$ und der Vektor $b_k^{(2)}$ die Gewichte der zweiten Schicht des MLP.

Durch Setzen von `OutputFunction` kann die Aktivierungsfunktion der Ausgabeschicht festgelegt werden. Für `OutputFunction` = *'linear'* werden die Daten einfach kopiert:

$$y_k = a_k^{(2)}, \qquad k = 1, \ldots, n_o$$

Diese Art der Aktivierungsfunktion sollte bei Regressionsaufgaben (Funktionsapproximation) verwendet werden. Diese Aktivierungsfunktion eignet sich nicht für Klassifikationsaufgaben.

Für `OutputFunction` = *'logistic'* werden die Aktivierungen wie folgt berechnet:

$$y_k = \frac{1}{1 + \exp\left(-a_k^{(2)}\right)}, \qquad k = 1, \ldots, n_o$$

Diese Art der Aktivierungsfunktion sollte bei Klassifikationsaufgaben verwendet werden, bei denen mehrere (`NumOutput`) unabhängige logische Attribute als Ausgabe auftreten können. Dies kommt nur bei sehr wenigen Klassifikationsaufgaben vor.

Für `OutputFunction` = *'softmax'* werden die Aktivierungen wie folgt berechnet:

$$y_k = \frac{\exp\left(a_k^{(2)}\right)}{\sum_{l=1}^{n_o} \exp\left(a_l^{(2)}\right)}, \qquad k = 1, \ldots, n_o$$

Diese Art der Aktivierungsfunktion sollte bei üblichen Klassifikationsaufgaben verwendet werden, bei denen mehrere (`NumOutput`), sich gegenseitig ausschließende Klassen auftreten. Insbesondere muss `OutputFunction` = *'softmax'* bei der Klassifikation von Pixeldaten mit `classify_image_class_mlp` verwendet werden.

Mit den Parametern `Preprocessing` und `NumComponents` kann eine Vorverarbeitung der Merkmalsvektoren festgelegt werden. Für `Preprocessing` = *'none'* werden die Merkmalsvektoren ohne Änderung an das MLP übergeben. `NumComponents` wird hier ignoriert.

Für alle anderen Werte von `Preprocessing` wird aus dem Trainingsdatensatz eine Transformation der Merkmale berechnet, die sowohl beim Training als auch bei der späteren Klassifikation oder Auswertung dazu verwendet wird, die Merkmalsvektoren zu transformieren.

Für `Preprocessing` = *'normalization'* werden die Merkmalsvektoren normalisiert, indem der Mittelwert der Trainingsvektoren von den Merkmalsvektoren abgezogen wird und die dadurch entstehenden Merkmalsvektoren durch die Standardabweichung der jeweiligen Komponente der Trainingsvektoren geteilt wird. Die transformierten Merkmalsvektoren haben also einen Mittelwert von 0 und eine Standardabweichung von 1 in jeder Komponente. Die Normalisierung ändert die Länge des Merkmalsvektors nicht. `NumComponents` wird hier ignoriert. Diese Transformation empfiehlt sich bei allen Daten, in denen die Mittelwerte und Standardabweichungen der Merkmalsvektoren sich stark von 0 bzw. 1 unterscheiden oder bei denen die Komponenten der Daten nicht in denselben Einheiten gemessen werden (z.B., falls einige der Daten Grauwertmerkmale und andere Regionenmerkmale sind, oder falls z.B. Regionenmerkmale wie `'circularity'` (Einheit: Skalar) und `'area'` (Einheit: Quadratpixel) gemischt werden). In diesem Fall benötigt das Trainieren des MLP typischerweise weniger Iterationen als ohne Normalisierung.

Für `Preprocessing` = *'principal_components'* werden die Merkmalsvektoren einer Hauptachsentransformation (principal component analysis) unterzogen. Die Hauptachsentransformation normalisiert zunächst die Merkmalsvektoren (s.o.). Danach wird eine orthogonale Transformation (eine Rotation im Merkmalsraum) berechnet, die die Trainingsvektoren dekorreliert. Nach der Transformation ist der Mittelwert der Trainingsvektoren 0 und die Kovarianzmatrix der Trainingsvektoren ist eine Diagonalmatrix. Die Transformation wird so bestimmt, dass die transformierten Merkmale, die die größte Variationsbreite aufweisen, als erstes in dem transformierten Merkmalsvektor stehen. Dadurch kann erreicht werden, dass die letzten Komponenten des Merkmalsvektors, die typischerweise stark vom Rauschen beeinflusst werden, ohne großen Informationsverlust weggelassen werden können. Mit

NumComponents wird festgelegt, wie viele der transformierten Komponenten verwendet werden sollen. Es können bis zu NumInput Komponenten selektiert werden. Mit Hilfe von get_prep_info_class_mlp kann der Informationsgehalt der einzelnen transformierten Komponenten bestimmt werden, und somit NumComponents einfacher bestimmt werden. Wie die Normalisierung empfiehlt sich diese Transformation, wenn die Merkmalsvektoren Mittelwerte und Standardabweichungen haben, die sich stark von 0 bzw. 1 unterscheiden oder bei denen die Komponenten der Merkmalsvektoren nicht in denselben Einheiten gemessen werden, und wenn zusätzlich zu erwarten ist, dass die Merkmale stark korreliert sind.

Im Gegensatz zu den obigen drei Transformationen, welche für beliebige Typen des MLP verwendet werden können, kann die durch Preprocessing = *'canonical_variates'* (kanonische Merkmale) spezifizierte Transformation nur verwendet werden, falls das MLP als Klassifikator mit OutputFunction = *'softmax'* verwendet wird. In diesem Fall wird eine Transformation bestimmt, die die Trainingsvektoren zuerst normalisiert und dann im Mittel über alle Klassen dekorreliert. Gleichzeitig werden in den transformierten Trainingsvektoren die Mittelwerte der einzelnen Klassen möglichst weit separiert. Wie bei Preprocessing = *'principal_components'* werden die transformierten Komponenten nach Informationsgehalt sortiert, so dass transformierte Merkmale mit wenig Informationsgehalt weggelassen werden können. Bei kanonischen Merkmalen können höchstens $\min(\texttt{NumOutput} - 1, \texttt{NumInput})$ Merkmale selektiert werden. Auch hier kann mit get_prep_info_class_mlp der Informationsgehalt der einzelnen transformierten Komponenten bestimmt werden. Wie die Hauptachsentransformation können die kanonischen Merkmale dazu verwendet werden, die Datenmenge ohne großen Informationsverlust zu verringern, wobei zusätzlich noch die Trennbarkeit der Daten nach der Datenreduktion optimiert wird.

Für die letzten zwei Transformationsarten (*'principal_components'* und *'canonical_variates'*) bestimmt NumComponents die Anzahl der Einheiten in der Eingabeschicht, während NumInput die Dimensionalität der Eingabedaten (Länge des untransformierten Merkmalsvektors) bestimmt. Dadurch wird die Anzahl der Eingabevariablen des MLP geringer, und daher kann typischerweise auch die Anzahl der versteckten Einheiten des MLP geringer gewählt werden, wodurch sich normalerweise die Trainingszeit und die Evaluierungs- und Klassifikationszeit verringert.

Normalerweise sollte NumHidden in der Größenordnung von NumInput und NumOutput gewählt werden. In vielen Fällen führen auch viel kleinere Werte von NumHidden schon zu sehr guten Klassifikationsergebnissen. Wenn NumHidden zu groß gewählt wird, besteht die Gefahr, dass das MLP sich an die Trainingsdaten überanpasst, was zu schlechten Generalisierungseigenschaften führt, d.h. das MLP lernt die Trainingsdaten sehr gut, liefert aber auf unbekannten Daten keine besonders guten Ergebnisse.

create_class_mlp initialisiert die oben beschriebenen Gewichte mit Zufallszahlen. Damit die Ergebnisse des Trainierens des MLP mit train_class_mlp reproduzierbar werden, wird in RandSeed der Initialisierungswert des Zufallszahlengenerators angegeben. Falls das Training einen relativ hohen Fehler zurückliefert, kann eventuell durch eine andere Wahl von RandSeed und durch erneutes Trainieren ein kleinerer Trainingsfehler erreicht werden.

Nachdem das MLP erzeugt wurde, werden typischerweise mit add_sample_class_mlp oder read_samples_class_mlp Trainingsdaten zum Netz hinzugefügt und das MLP mit train_class_mlp trainiert. Daraufhin kann das MLP mit write_class_mlp abgespeichert werden. Alternativ können auch sofort nach dem Training Daten mit evaluate_class_mlp bewertet werden oder (für OutputFunction = *'softmax'*) mit classify_class_mlp klassifiziert werden.

Das Training des MLPs führt im Normalfall zu relativ scharfen Grenzen zwischen den unterschiedlichen Klassen, d.h. die Konfidenz für eine Klasse fällt von nahe 1 (innerhalb der Region einer Klasse) auf nahe 0 (innerhalb der Region einer anderen Klasse) innerhalb eines schmalen „Bandes“ im Merkmalsraum. Falls die Klassen sich nicht überlappen, erfolgt dieser Übergang an einem geeigneten Ort zwischen den Klassen; falls sich die Klassen überlappen, erfolgt dieser Übergang an einem geeigneten Ort im Überlappungsbereich der Klassen. Dieser scharfe Übergang ist in vielen Anwendungen erwünscht. In einigen Anwendungen ist hingegen ein glatterer Übergang zwischen verschiedenen Klassen wünschenswert (d.h. ein Übergang in einem breiteren „Band“ im Merkmalsraum), um ein gewisses Niveau an Unsicherheit in der Region zwischen den Klassen auszudrücken. Weiterhin kann es, wie oben beschrieben, wünschenswert sein, eine Überanpassung des MLPs an die Trainingsdaten zu verhindern. Zu diesem Zweck kann das MLP mit set_regularization_params_class_mlp regularisiert werden.

Ein wie oben definiertes MLP besitzt keine inhärente Fähigkeit zur Neuheitserkennung, d.h. es klassifiziert zufällige Merkmalsvektoren in eine der Klassen mit einer Konfidenz nahe 1 (außer der zufällige Merkmalsvektor liegt zufälligerweise ein einer Region des Merkmalsraums, in dem sich die Trainingsdaten von zwei Klassen überlappen). In einigen Anwendungen ist es jedoch wünschenswert, Merkmalsvektoren zurückzuweisen, die nicht nahe an irgendeiner Klasse liegen, wobei „Nähe“ über die Distanz des Merkmalsvektors zu der Menge der Merkmalsvektoren in den Trainingsdaten definiert ist. Um das MLP mit der Fähigkeit zur Neuheitserkennung auszustatten, d.h. um

Merkmalsvektoren zurückweisen zu können, die zu keiner Klasse gehören, kann eine explizite Rückweisungsklasse erzeugt werden, indem NumOutput auf die Anzahl der tatsächlichen Klassen plus 1 gesetzt wird. Danach kann set_rejection_params_class_mlp verwendet werden, um train_class_mlp so zu konfigurieren, dass automatisch Trainingsdaten für diese Rüchweisungsklasse erzeugt werden.

Die Kombination der Regularisierung mit der automatischen Erzeugung einer Rückweisungsklasse ist in vielen Anwendungen nützlich, da sie einen glatten Übergang zwischen den tatsächlichen Klassen sowie von den tatsächlichen Klassen zur Rückweisungsklasse erzeugt. Dies spiegelt die Anforderungen dieser Anwendungen wieder, dass nur Merkmalsvektoren innerhalb des Bereichs des Merkmalsraums, der den Trainingsdaten entspricht, eine Konfidenz nahe 1 erhalten sollen, während zufällige Merkmalsvektoren, die zu keiner Klasse gehören, eine Konfidenz nahe 0 erhalten sollen, und dass Übergänge zwischen den Klassen glatt sein sollen, was eine zunehmende Unsicherheit widerspiegelt, je weiter ein Merkmalsvektor von der jeweiligen Klasse entfernt liegt. Insbesondere haben OCR-Anwendungen manchmal diese Anforderung (siehe create_ocr_class_mlp).

Ein Vergleich zwischen MLP und Support-Vektor-Maschinen (SVM) (siehe create_class_svm) zeigt typischerweise, dass SVMs generell schneller trainiert werden, insbesondere bei großen Trainingsdatensätzen, und eine leicht verbesserte Erkennungsrate haben. Das MLP weist schnellere Klassifikationszeiten auf und sollte daher in zeitkritischen Anwendungen verwendet werden. Es ist zu beachten, dass der Vergleich von optimal abgestimmten Parametern ausgeht.

Parameter

▷ **NumInput** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Eingabevariablen (Merkmale) des MLP.
Defaultwert: 20
Wertevorschläge: NumInput ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Restriktion: NumInput >= 1

▷ **NumHidden** (input_control) . . . . . integer ⇝ *integer*
Anzahl der versteckten Einheiten des MLP.
Defaultwert: 10
Wertevorschläge: NumHidden ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 150}
Restriktion: NumHidden >= 1

▷ **NumOutput** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Ausgabevariablen (Klassen) des MLP.
Defaultwert: 5
Wertevorschläge: NumOutput ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 150}
Restriktion: NumOutput >= 1

▷ **OutputFunction** (input_control) . . . . . string ⇝ *string*
Art der Aktivierungsfunktion in der Ausgabeschicht des MLP.
Defaultwert: 'softmax'
Werteliste: OutputFunction ∈ {'linear', 'logistic', 'softmax'}

▷ **Preprocessing** (input_control) . . . . . string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
Defaultwert: 'normalization'
Werteliste: Preprocessing ∈ {'none', 'normalization', 'principal_components', 'canonical_variates'}

▷ **NumComponents** (input_control) . . . . . integer ⇝ *integer*
Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale (ignoriert bei Preprocessing = *'none'* und Preprocessing = *'normalization'*).
Defaultwert: 10
Wertevorschläge: NumComponents ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Restriktion: NumComponents >= 1

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Initialisierungswert des Zufallszahlengenerators, der zur Initialisierung des MLP mit zufälligen Werten verwendet wird.
Defaultwert: 42

▷ **MLPHandle** (output_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

Beispiel

```
* Use the MLP for regression (function approximation)
```

```
create_class_mlp (1, NumHidden, 1, 'linear', 'none', 1, 42, MLPHandle)
* Generate the training data
* D = [...]
* T = [...]
* Add the training data
for J := 0 to NumData-1 by 1
    add_sample_class_mlp (MLPHandle, D[J], T[J])
endfor
* Train the MLP
train_class_mlp (MLPHandle, 200, 0.001, 0.001, Error, ErrorLog)
* Generate test data
* X = [...]
* Compute the output of the MLP on the test data
for J := 0 to N-1 by 1
    evaluate_class_mlp (MLPHandle, X[J], Y)
endfor

* Use the MLP for classification
create_class_mlp (NumIn, NumHidden, NumOut, 'softmax', \
                  'normalization', NumIn, 42, MLPHandle)
* Generate and add the training data
for J := 0 to NumData-1 by 1
    * Generate training features and classes
    * Data = [...]
    * Class = [...]
    add_sample_class_mlp (MLPHandle, Data, Class)
endfor
* Train the MLP
train_class_mlp (MLPHandle, 100, 1, 0.01, Error, ErrorLog)
* Use the MLP to classify unknown data
for J := 0 to N-1 by 1
    * Extract features
    * Features = [...]
    classify_class_mlp (MLPHandle, Features, 1, Class, Confidence)
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`add_sample_class_mlp`, `set_regularization_params_class_mlp`, `set_rejection_params_class_mlp`

Alternativen

`read_dl_classifier`, `create_class_svm`, `create_class_gmm`

Siehe auch

`clear_class_mlp`, `train_class_mlp`, `classify_class_mlp`, `evaluate_class_mlp`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Andrew Webb: „Statistical Pattern Recognition“; Arnold, London; 1999.

Modul

Foundation

deserialize_class_mlp (: : SerializedItemHandle : MLPHandle)

Deserialisieren eines serialisierten mehrschichtigen Perzeptrons (MLP).

deserialize_class_mlp deserialisiert ein mehrschichtiges Perzeptron (MLP) (einschließlich eventuell abgespeicherter Trainingsmuster), welches mit serialize_class_mlp serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte MLP wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten MLP mit dem Handle MLPHandle gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **MLPHandle** (output_control) ..... class_mlp ⇝ *handle*
  Handle des MLP.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_class_mlp

Nachfolger

classify_class_mlp, evaluate_class_mlp, create_class_lut_mlp

Siehe auch

create_class_mlp, write_class_mlp, serialize_class_mlp

Modul

Foundation

evaluate_class_mlp (: : MLPHandle, Features : Result)

Berechnen der Bewertung eines Merkmalsvektors durch ein mehrschichtiges Perzeptron.

evaluate_class_mlp berechnet das Ergebnis Result der Evaluierung des Merkmalsvektors Features durch das mehrschichtige Perzeptron (MLP) MLPHandle. Die Berechnungsformeln sind bei create_class_mlp angegeben. Das MLP muss vor der Verwendung von evaluate_class_mlp mit train_class_mlp trainiert werden.

Falls das MLP zur Regression (Funktionsapproximation) verwendet wird (OutputFunction = *'linear'*), ist Result der Funktionswert der Funktion an der Koordinate Features. Für OutputFunction = *'logistic'* und *'softmax'* können die Werte in Result als Wahrscheinlichkeiten interpretiert werden. Für OutputFunction = *'logistic'* geben die Elemente von Result somit für jedes der unabhängigen Attribute die Wahrscheinlichkeit des Vorhandenseins des jeweiligen Attributes an. Typischerweise wird hier ein Schwellenwert von 0.5 verwendet, um zu entscheiden, ob das Attribut vorhanden ist. Je nach Anwendung können aber auch andere Schwellenwerte in Betracht kommen. Für OutputFunction = *'softmax'* wird typischerweise die Position des Maximums von Result als die Klasse des Merkmalsvektors interpretiert und der zugehörige Wert als die Wahrscheinlichkeit der

Klasse. In diesem Fall sollte statt `evaluate_class_mlp` `classify_class_mlp` verwendet werden, da `classify_class_mlp` direkt die Klasse und Wahrscheinlichkeit zurückliefert.

Parameter

- ▷ **`MLPHandle`** (input_control) . . . . . class_mlp ⇝ *handle*
  Handle des MLP.
- ▷ **`Features`** (input_control) . . . . . real-array ⇝ *real*
  Merkmalsvektor.
- ▷ **`Result`** (output_control) . . . . . real-array ⇝ *real*
  Ergebnis der Auswertung des Merkmalsvektors durch das MLP.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `evaluate_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_mlp`, `read_class_mlp`

Alternativen

`classify_class_mlp`

Siehe auch

`create_class_mlp`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Andrew Webb: „Statistical Pattern Recognition“; Arnold, London; 1999.

Modul

Foundation

`get_class_train_data_mlp` (: : `MLPHandle` : `ClassTrainDataHandle`)

Liest die Trainingsdaten eines MLP aus.

`get_class_train_data_mlp` liest die Trainingsdaten eines MLP aus und liefert diese als Handle in `ClassTrainDataHandle` zurück.

Parameter

- ▷ **`MLPHandle`** (input_control) . . . . . class_mlp ⇝ *handle*
  Handle eines MLP.
- ▷ **`ClassTrainDataHandle`** (output_control) . . . . . class_train_data ⇝ *handle*
  Handle der Trainingsdaten des MLP.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_class_train_data_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

add_sample_class_mlp, read_samples_class_mlp

Nachfolger

add_class_train_data_svm, add_class_train_data_gmm, add_class_train_data_knn

Siehe auch

create_class_train_data

Modul

Foundation

get_params_class_mlp (: : MLPHandle : NumInput, NumHidden, NumOutput, OutputFunction, Preprocessing, NumComponents)

Auslesen der Parameter eines mehrschichtigen Perzeptrons.

get_params_class_mlp gibt die Parameter des durch MLPHandle angegebenen mehrschichtigen Perzeptrons (MLP), die bei der Erzeugung mit create_class_mlp verwendet wurden, zurück. Dies ist insbesondere nützlich, wenn das MLP mit read_class_mlp von Datei eingelesen wurde. Die Ausgabedaten von get_params_class_mlp können z.B. verwendet werden, um zu überprüfen, ob die Merkmalsvektoren und ggf. die Zieldaten, die verwendet werden sollen, zu dem MLP passen. Zur Beschreibung der Parameter siehe create_class_mlp.

Parameter

- ▷ **MLPHandle** (input_control) ..... class_mlp ⇝ *handle*
  Handle des MLP.
- ▷ **NumInput** (output_control) ..... integer ⇝ *integer*
  Anzahl der Eingabevariablen (Merkmale) des MLP.
- ▷ **NumHidden** (output_control) ..... integer ⇝ *integer*
  Anzahl der versteckten Einheiten des MLP.
- ▷ **NumOutput** (output_control) ..... integer ⇝ *integer*
  Anzahl der Ausgabevariablen (Klassen) des MLP.
- ▷ **OutputFunction** (output_control) ..... string ⇝ *string*
  Art der Aktivierungsfunktion in der Ausgabeschicht des MLP.
- ▷ **Preprocessing** (output_control) ..... string ⇝ *string*
  Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
- ▷ **NumComponents** (output_control) ..... integer ⇝ *integer*
  Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_params_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_class_mlp, read_class_mlp

Nachfolger

add_sample_class_mlp, train_class_mlp

Siehe auch

evaluate_class_mlp, classify_class_mlp

Modul

Foundation

get_prep_info_class_mlp (: : MLPHandle, Preprocessing : InformationCont, CumInformationCont)

Berechnen des Informationsgehalts der vorverarbeiteten Merkmalsvektoren eines mehrschichtigen Perzeptrons.

get_prep_info_class_mlp berechnet den Informationsgehalt der mit der durch Preprocessing gegebenen Vorverarbeitung transformierten Komponenten der Trainingsvektoren. Preprocessing kann auf *'principal_components'* oder *'canonical_variates'* gesetzt werden. Die zugrundeliegenden Vorverarbeitungen sind bei create_class_mlp beschrieben. Der Informationsgehalt wird aus die Variation der transformierten Komponenten der Trainingsvektoren berechnet, d.h. er wird rein aufgrund der Trainingsdaten unabhängig von einer Fehlerrate bei einer Klassifikation der Trainingsdaten berechnet. Der Informationsgehalt wird für alle relevanten Komponenten der transformierten Merkmalsvektoren (NumInput für *'principal_components'* und $\min(\texttt{NumOutput}-1, \texttt{NumInput})$ für *'canonical_variates'*, siehe create_class_mlp) in InformationCont zurückgegeben. Der Informationsgehalt wird als eine Zahl zwischen 0 und 1 dargestellt. Ein prozentualer Informationsgehalt kann leicht durch Multiplikation mit 100 berechnet werden. Der kumulierte Informationsgehalt der ersten n Komponenten wird in der n-ten Komponente von CumInformationCont zurückgegeben, d.h. CumInformationCont enthält die Summen der ersten n Elemente von InformationCont. Um get_prep_info_class_mlp verwenden zu können, müssen mit add_sample_class_mlp oder read_samples_class_mlp genügend viele Trainingsmuster zu dem durch MLPHandle gegebenen mehrschichtigen Perzeptron (MLP) hinzugefügt werden.

InformationCont und CumInformationCont können dazu verwendet werden, um zu entscheiden, wie viele Komponenten der transformierten Merkmalsvektoren relevante Information enthalten. Ein oft verwendetes Kriterium ist z.B. zu verlangen, dass die transformierten Daten x% (z.B. 90%) der Daten enthalten sollten. Dies kann leicht anhand des ersten Wertes von CumInformationCont, der über x% liegt, bestimmt werden. Der so erhaltene Wert kann bei einem neuerlichen Aufruf von create_class_mlp als NumComponents verwendet werden. Da zum Aufruf von get_prep_info_class_mlp schon ein MLP mit create_class_mlp erzeugt werden muss, also auch ein initialer Wert von NumComponents bei create_class_mlp angegeben werden muss, aber bei Verwendung von get_prep_info_class_mlp typischerweise noch nicht bekannt ist, wie viele Komponenten relevant sind, empfiehlt sich folgendes zweistufiges Vorgehen, um NumComponents zu bestimmen: In einem ersten Schritt wird ein MLP mit der maximalen Anzahl von NumComponents (NumInput für *'principal_components'* und $\min(\texttt{NumOutput}-1, \texttt{NumInput})$ für *'canonical_variates'*) erzeugt. Dann werden die Trainingsmuster zu dem MLP hinzugefügt und mit write_samples_class_mlp in Datei gespeichert. Anschließend wird mit get_prep_info_class_mlp der Informationsgehalt der Komponenten und somit NumComponents bestimmt. Danach wird ein neues MLP mit der gewünschten Anzahl Komponenten erzeugt, und die abgespeicherten Trainingsdaten mit read_samples_class_mlp wieder eingelesen. Hierauf wird das MLP mit train_class_mlp trainiert.

Parameter

- ▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
  Handle des MLP.
- ▷ **Preprocessing** (input_control) . . . . . string ⇝ *string*
  Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
  Defaultwert: 'principal_components'
  Werteliste: Preprocessing ∈ {'principal_components', 'canonical_variates'}
- ▷ **InformationCont** (output_control) . . . . . real-array ⇝ *real*
  Relativer Informationsgehalt der transformierten Merkmalsvektoren.
- ▷ **CumInformationCont** (output_control) . . . . . real-array ⇝ *real*
  Kumulierter Informationsgehalt der transformierten Merkmalsvektoren.

Beispiel

```
* Create the initial MLP
create_class_mlp (NumIn, NumHidden, NumOut, 'softmax', \
                  'principal_components', NumIn, 42, MLPHandle)
* Generate and add the training data
```

```
for J := 0 to NumData-1 by 1
    * Generate training features and classes
    * Data = [...]
    * Class = [...]
    add_sample_class_mlp (MLPHandle, Data, Class)
endfor
write_samples_class_mlp (MLPHandle, 'samples.mtf')
* Compute the information content of the transformed features
get_prep_info_class_mlp (MLPHandle, 'principal_components',\
                         InformationCont, CumInformationCont)
* Determine NumComp by inspecting InformationCont and CumInformationCont
* NumComp = [...]
* Create the actual MLP
create_class_mlp (NumIn, NumHidden, NumOut, 'softmax', \
                  'principal_components', NumComp, 42, MLPHandle)
* Train the MLP
read_samples_class_mlp (MLPHandle, 'samples.mtf')
train_class_mlp (MLPHandle, 100, 1, 0.01, Error, ErrorLog)
write_class_mlp (MLPHandle, 'classifier.mlp')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_prep_info_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls `get_prep_info_class_mlp` den Fehler 9211 (Matrix ist nicht positiv definit) bei `Preprocessing` = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_class_mlp`, `read_samples_class_mlp`

Nachfolger

`clear_class_mlp`, `create_class_mlp`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Andrew Webb: „Statistical Pattern Recognition“; Arnold, London; 1999.

Modul

Foundation

get_regularization_params_class_mlp (: : MLPHandle, GenParamName : GenParamValue)

Auslesen der Regularisierungsparameter eines mehrschichtigen Perzeptrons.

`get_regularization_params_class_mlp` gibt die Regularisierungsparameter eines mehrschichtigen Perzeptrons (MLP), die mit `set_regularization_params_class_mlp` spezifiziert wurden, zurück. Weiterhin gibt `get_regularization_params_class_mlp` die Parameter zurück, die durch die automatische Bestimmung der Regularisierungsparameter bestimmt wurden. Zur Beschreibung der Parameter siehe `set_regularization_params_class_mlp`.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

▷ **GenParamName** (input_control) . . . . . string ⇝ *string*
Name des auszulesenden Regularisierungsparameters.
Defaultwert: 'weight_prior'
Werteliste: `GenParamName` ∈ {'weight_prior', 'noise_prior', 'num_well_determined_params', 'fraction_well_determined_params', 'num_outer_iterations', 'num_inner_iterations'}

▷ **GenParamValue** (output_control) . . . . . number(-array) ⇝ *real* / integer
Wert des Regularisierungsparameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_regularization_params_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`set_regularization_params_class_mlp`, `read_class_mlp`

Nachfolger

`train_class_mlp`

Modul

Foundation

get_rejection_params_class_mlp (: : MLPHandle, GenParamName : GenParamValue)

Auslesen der Parameter der Rückweisungsklasse eines mehrschichtigen Perzeptrons.

`get_rejection_params_class_mlp` gibt die Parameter der Rückweisungsklasse eines mehrschichtigen Perzeptrons (MLP), die mit `set_rejection_params_class_mlp` spezifiziert wurden, zurück. Zur Beschreibung der Parameter siehe `set_rejection_params_class_mlp`.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
MLP handle.

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Names of the generic parameters to return.
Defaultwert: 'sampling_strategy'
Werteliste: `GenParamName` ∈ {'sampling_strategy', 'hyperbox_tolerance', 'rejection_sample_factor', 'random_seed', 'rejection_class_index'}

▷ **GenParamValue** (output_control) . . . . . string(-array) ⇝ *string* / real / integer
Values of the generic parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_rejection_params_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_class_mlp

Nachfolger

train_class_mlp

Modul

Foundation

get_sample_class_mlp (: : MLPHandle, IndexSample : Features, Target)

Auslesen eines Trainingsmusters aus den Trainingsdaten eines mehrschichtigen Perzeptrons.

get_sample_class_mlp liest ein Trainingsmuster, das mit add_sample_class_mlp oder read_samples_class_mlp hinzugefügt wurde, aus dem mehrschichtigen Perzeptron (MLP) MLPHandle aus. Der Index des auszulesenden Musters wird mit IndexSample festgelegt. Er wird ab 0 gezählt, d.h. IndexSample muss zwischen 0 und IndexSamples − 1 liegen, wobei IndexSamples mit get_sample_num_class_mlp bestimmt werden kann. Das Trainingsmuster wird in Features und Target zurückgegeben. Dabei ist Features ein Merkmalsvektor der Länge NumInput und Target ein Zielvektor der Länge NumOutput (siehe add_sample_class_mlp und create_class_mlp).

get_sample_class_mlp kann z.B. dazu verwendet werden, die Trainingsdaten mit classify_class_mlp zu reklassifizieren, um festzustellen, welche der Trainingsmuster eventuell falsch klassifiziert werden.

Parameter

- ▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
  Handle des MLP.
- ▷ **IndexSample** (input_control) . . . . . integer ⇝ *integer*
  Nummer des gespeicherten Trainingsmusters.
- ▷ **Features** (output_control) . . . . . real-array ⇝ *real*
  Merkmalsvektor des Trainingsmusters.
- ▷ **Target** (output_control) . . . . . real-array ⇝ *real*
  Zielvektor des Trainingsmusters.

Beispiel

```
* Train an MLP
create_class_mlp (NumIn, NumHidden, NumOut, 'softmax', \
                  'canonical_variates', NumComp, 42, MLPHandle)
read_samples_class_mlp (MLPHandle, 'samples.mtf')
train_class_mlp (MLPHandle, 100, 1, 0.01, Error, ErrorLog)
* Reclassify the training samples
get_sample_num_class_mlp (MLPHandle, NumSamples)
for I := 0 to NumSamples-1 by 1
    get_sample_class_mlp (MLPHandle, I, Data, Target)
    classify_class_mlp (MLPHandle, Data, 1, Class, Confidence)
    Result := gen_tuple_const(NumOut,0)
    Result[Class] := 1
    Diffs := Target-Result
    if (sum(fabs(Diffs)) > 0)
        * Sample has been classified incorrectly
    endif
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sample_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_mlp, read_samples_class_mlp, get_sample_num_class_mlp

Nachfolger

classify_class_mlp, evaluate_class_mlp

Siehe auch

create_class_mlp

Modul

Foundation

get_sample_num_class_mlp (: : MLPHandle : NumSamples)

Auslesen der Anzahl der Trainingsmuster in den Trainingsdaten eines mehrschichtigen Perzeptrons.

get_sample_num_class_mlp liefert in NumSamples die Anzahl der in dem durch MLPHandle gegebenen mehrschichtigen Perzeptron (MLP) verfügbaren Trainingsmuster zurück. get_sample_num_class_mlp sollte aufgerufen werden, bevor mit get_sample_class_mlp die einzelnen Trainingsmuster ausgelesen werden, z.B. um eine Reklassifikation der Trainingsdaten durchzuführen (siehe get_sample_class_mlp).

Parameter

▷ **MLPHandle** (input_control) ..... class_mlp ⇝ *handle*
Handle des MLP.

▷ **NumSamples** (output_control) ..... integer ⇝ *integer*
Anzahl der gespeicherten Trainingsmuster.

Ergebnis

Falls MLPHandle korrekt ist, liefert get_sample_num_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_mlp, read_samples_class_mlp

Nachfolger

get_sample_class_mlp

Siehe auch

create_class_mlp

Modul

Foundation

read_class_mlp (: : FileName : MLPHandle)

Lesen eines mehrschichtigen Perzeptrons aus Datei.

read_class_mlp liest ein mit write_class_mlp abgespeichertes mehrschichtiges Perzeptron (MLP) ein. Da das Trainieren eines MLP relativ lange dauern kann, wird das MLP typischerweise in einem Offline-Prozess trainiert und mit write_class_mlp gespeichert. Im Online-Prozess wird das MLP mit

read_class_mlp gelesen und danach zur Evaluierung mit evaluate_class_mlp oder zur Klassifikation mit classify_class_mlp verwendet. Die Dateiendung für den MLP-Klassifikator ist in HALCON 'gmc'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: .gmc

▷ **MLPHandle** (output_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

classify_class_mlp, evaluate_class_mlp, create_class_lut_mlp

Alternativen

read_dl_classifier

Siehe auch

create_class_mlp, write_class_mlp

Modul

Foundation

read_samples_class_mlp (: : MLPHandle, FileName :)

Lesen von Trainingsdaten eines mehrschichtigen Perzeptrons aus Datei.

read_samples_class_mlp liest Trainingsmuster aus der durch FileName gegebenen Datei aus und fügt sie zu den schon in dem mehrschichtigen Perzeptron (MLP) MLPHandle vorhandenen Trainingsmustern hinzu. Das MLP muss zuvor mit create_class_mlp erzeugt werden. Wie bei train_class_mlp und write_samples_class_mlp beschrieben, können read_samples_class_mlp, add_sample_class_mlp und write_samples_class_mlp dazu verwendet werden, einen großen Datensatz von Trainingsmustern aufzubauen, um somit die Leistung des MLP durch erneutes Trainieren zu verbessern.

Es ist zu beachten, dass die Trainingsdaten die korrekte Dimensionalität aufweisen. Die in FileName gespeicherten Merkmalsvektoren und Zielvektoren müssen die Längen NumInput und NumOutput besitzen, die bei create_class_mlp angegeben worden ist. Falls dies nicht der Fall ist, wird eine Fehlermeldung zurückgegeben.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_samples_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_mlp`

Nachfolger

`train_class_mlp`

Alternativen

`add_sample_class_mlp`

Siehe auch

`write_samples_class_mlp`, `clear_samples_class_mlp`

Modul

Foundation

`select_feature_set_mlp` `( : : ClassTrainDataHandle, SelectionMethod, GenParamName, GenParamValue : MLPHandle, SelectedFeatureIndices, Score )`

Wählt aus einer Liste an Merkmalen eine optimale Untermenge für ein bestimmtes Klassifikationsproblem aus.

`select_feature_set_mlp` wählt die optimalen Merkmale für ein bestimmtes Klassifikationsproblem aus. Die Daten für das Klassifikationsproblem werden mit `ClassTrainDataHandle` angegeben. Als Klassifikator werden mehrschichtige Perzeptronen (MLP) verwendet.

In `MLPHandle` wird ein Klassifikator zurückgegeben, der mit der resultierenden Merkmalsauswahl parametrisiert und trainiert wurde. Die Merkmalsauswahl wird in `SelectedFeatureIndices` zurückgegeben.

Der Operator kann für folgende Probleme nützlich sein: Es gibt eine Liste von Merkmalen von denen einige wenige für die Lösung eines Problems ausreichen. So kann mit Hilfe dieses Operators eine fundierte Entscheidung getroffen werden welche dieser Merkmale relevant für das gegebene Problem sind. Außerdem können für bestimmte Verfahren der Merkmalsextraktion verschiedene Parametersätze verglichen werden.

Um das Klassifikationsproblem zu definieren, werden dem Handle `ClassTrainDataHandle` Merkmalsvektoren hinzugefügt, die aus einzelnen Untergruppen bestehen. Diese Untergruppen müssen mit Hilfe des Operators `set_feature_lengths_class_train_data` festgelegt werden. Diese Untergruppen werden dann von dem in `select_feature_set_mlp` ausgeführten Selektionsprozess entweder ausgewählt oder abgelehnt, je nachdem ob sie zur Lösung des Klassifikationsproblems beitragen.

Die so definierten Merkmale werden dann in `SelectedFeatureIndices` mit ihrem Index referenziert, außer es wurden zusätzlich Namen beim Aufruf von `set_feature_lengths_class_train_data` gesetzt. Ist das der Fall, werden statt der Indizes die Namen der ausgewählten Merkmale zurückgegeben. Wurde `set_feature_lengths_class_train_data` zuvor nicht aufgerufen, werden die einzelnen Spalten als Merkmale interpretiert.

Für den Selektionsprozess können in `SelectionMethod` zwei verschiedene Methoden ausgewählt werden: entweder die Greedy-Variante *'greedy'* (das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt) oder die dynamisch oszillierende Suche *'greedy_oscillating'* (Das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt. Danach wird getestet ob eines der hinzugefügten Merkmale entbehrlich ist.). Während *'greedy'* schneller terminiert, kann *'greedy_oscillating'* bessere Ergebnisse erzielen wenn die Dimensionen der Merkmale sehr klein ist oder redundante Daten vorliegen.

Als Optimierungskriterium dient die Klassifikationsrate, die mit einem zweifachen Kreuzvalidierungsverfahren ermittelt wird. Die beste erreichte Klassifikationsrate wird in `Score` zurückgegeben.

Über die generischen Parameter GenParamName und GenParamValue kann die Anzahl der Verwendeten Neuronen im versteckten Layer des MLP über *'num_hidden'* gesetzt werden, default ist hierfür *80*, größere Werte führen zu längeren Trainingszeiten, erlauben aber eine ausdrucksstärkere Klassifikation.

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lange sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator `select_feature_set_mlp` zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

- ▷ **ClassTrainDataHandle** (input_control) . . . . . class_train_data ⇝ *handle*
  Handle der Trainingsdaten. Sollte zuvor in Untermerkmale aufgeteilt worden sein.
- ▷ **SelectionMethod** (input_control) . . . . . string ⇝ *string*
  Methode zur Auswahl der besten Merkmale.
  Defaultwert: 'greedy'
  Werteliste: `SelectionMethod` ∈ {'greedy', 'greedy_oscillating'}
- ▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
  Namen der generischen Parameter zum Steuern des Klassifikators und der Merkmalsauswahl.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'num_hidden'}
- ▷ **GenParamValue** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Werte der generischen Parameter zum Steuern des Klassifikators und der Merkmalsauswahl.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {50, 80, 100}
- ▷ **MLPHandle** (output_control) . . . . . class_mlp ⇝ *handle*
  Trainierter Klassifikator.
- ▷ **SelectedFeatureIndices** (output_control) . . . . . string-array ⇝ *string*
  Die Indizes der ausgewählten Merkmale.
- ▷ **Score** (output_control) . . . . . real-array ⇝ *real*
  Die Klassifikationsrate die mit dem ausgewählten Merkmalssatz erreicht wurde.

Beispiel

```
* Find out which of the two features distinguishes two Classes
NameFeature1 := 'Good Feature'
NameFeature2 := 'Bad Feature'
LengthFeature1 := 3
LengthFeature2 := 2
* Create training data
create_class_train_data (LengthFeature1+LengthFeature2,\
  ClassTrainDataHandle)
* Define the features which are in the training data
set_feature_lengths_class_train_data (ClassTrainDataHandle, [LengthFeature1,\
  LengthFeature2], [NameFeature1, NameFeature2])
* Add training data
*                                                    |Feat1|  |Feat2|
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  2,1  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  2,1  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  3,4  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  3,4  ], 1)
* Add more data
* ...
* Select the better feature with a MLP
select_feature_set_mlp (ClassTrainDataHandle, 'greedy', [], [], MLPHandle,\
  SelectedFeatureMLP, Score)
* Use the classifier
* ...
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `select_feature_set_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`create_class_train_data`, `add_sample_class_train_data`, `set_feature_lengths_class_train_data`

Nachfolger

`classify_class_mlp`

Alternativen

`select_feature_set_knn`, `select_feature_set_svm`, `select_feature_set_gmm`

Siehe auch

`select_feature_set_trainf_mlp`, `gray_features`, `region_features`

Modul

Foundation

`serialize_class_mlp` (: : MLPHandle : SerializedItemHandle)

Serialisieren eines mehrschichtigen Perzeptrons (MLP).

`serialize_class_mlp` serialisiert ein mehrschichtiges Perzeptron (MLP) einschließlich eventuell abgespeicherter Trainingsmuster (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_class_mlp` und `write_samples_class_mlp` in eine Datei geschrieben werden. Das MLP wird in dem Handle `MLPHandle` übergeben. Das serialisierte MLP wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_class_mlp` wieder deserialisiert werden.

Parameter

▷ **`MLPHandle`** (input_control) ..... class_mlp ⇝ *handle*
Handle des MLP.

▷ **`SerializedItemHandle`** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_mlp`

Nachfolger

`clear_class_mlp`, `fwrite_serialized_item`, `send_serialized_item`, `deserialize_class_mlp`

Siehe auch

create_class_mlp, read_class_mlp, write_samples_class_mlp, deserialize_class_mlp

Modul

Foundation

set_regularization_params_class_mlp (: : MLPHandle, GenParamName, GenParamValue :)

Setzen der Regularisierungsparameter eines mehrschichtigen Perzeptrons.

set_regularization_params_class_mlp setzt die Regularisierungsparameter des mehrschichtigen Perzeptrons (MLP), das in MLPHandle übergeben wurde. Der zu setzende Regularisierungsparameter wird mit GenParamName spezifiziert. Sein Wert wird mit GenParamValue spezifiziert.

GenParamName kann folgende Werte annehmen:

'num_outer_iterations': Dieser Parameter legt fest, ob die Regularisierungsparameter automatisch (GenParamValue >= 1) oder manuell (GenParamValue = 0, default) bestimmt werden, wie unten in den Abschnitten „Technischer Hintergrund“ und „Automatische Bestimmung der Regularisierungsparameter“ beschrieben. Wie detailliert im Abschnitt „Automatische Bestimmung der Regularisierungsparameter“ beschrieben, sollte *'num_outer_iterations'* nicht zu groß gesetzt werden (im Bereich 1 bis 5), um eine manuelle Überprüfung der Konvergenz der automatischen Bestimmung der Regularisierungsparameter zu ermöglichen.

'num_inner_iterations': Dieser Parameter ermöglicht potentiell eine etwas schnellere Konvergenz der der automatischen Bestimmung der Regularisierungsparameter, wie unten im Abschnitt „Automatische Bestimmung der Regularisierungsparameter“ beschrieben. Er sollte im Normalfall auf seinem Standardwert von 1 belassen werden.

'weight_prior': Dieser Parameter legt einerseits das zu verwendende Regularisierungsmodell fest, wie unten im Abschnitt „Technischer Hintergrund“ beschrieben. Andererseits legt er die Werte der Regularisierungsparameter fest, falls eine manuelle Bestimmung der Regularisierungsparameter eingestellt wurde (d.h. *'num_outer_iterations'* = 0), bzw. die Initialwerte der Regularisierungsparameter, falls eine automatische Bestimmung der Regularisierungsparameter eingestellt wurde (d.h. *'num_outer_iterations'* >= 1), wie unten im Abschnitt „Automatische Bestimmung der Regularisierungsparameter“ beschrieben. Die manuelle Bestimmung der Regularisierungsparameter (siehe den Abschnitt „Regularisierungsparameter“ unten) ist nur realistisch durchzuführen, falls ein einziger Regularisierungsparameter verwendet wird. In allen anderen Fällen sollten die Regularisierungsparameter automatisch bestimmt werden.

'noise_prior': Dieser Parameter erlaubt es die inverse Varianz des Rauschens der Daten festzulegen, falls das MLP für Regressionsaufgaben konfiguriert wurde, wie unten im Abschnitt „Anwendungsgebiete“ beschrieben. Falls eine manuelle Bestimmung der Regularisierungsparameter eingestellt wurde, wird die inverse Varianz mit GenParamName gesetzt, während ihr Initialwert festgelegt wird, falls eine automatische Bestimmung der Regularisierungsparameter eingestellt wurde. Im Normalfall ist es nur sinnvoll, diesen Parameter zu verwenden, falls die Regularisierungsparameter automatisch bestimmt werden.

Es ist zu beachten, dass die automatische Bestimmung der Regularisierungsparameter sehr viel Speicher und Laufzeit benötigt, wie im Detail im Abschnitt „Komplexität“ unten beschrieben. Deshalb sollte NumHidden nicht zu groß gewählt werden, wenn das MLP mit create_class_mlp erzeugt wird. So sind zum Beispiel für normale OCR-Anwendungen selten größere Werte für NumHidden als 30-60 notwendig.

Anwendungsgebiete

Wie bei create_class_mlp beschrieben, kann es wünschenswert sein, das MLP zu regularisieren, um einen glatteren Übergang der Konfidenzen zwischen zwei Klassen zu erzwingen und um eine Überanpassung des MLPs an die Trainingsdaten zu verhindern. Um dies zu erreichen, kann ein Strafterm für große MLP-Gewichte (die den Hauptgrund für scharfe Übergänge zwischen Klassen darstellen) zum Training des MLPs in train_class_mlp hinzugefügt werden. Hierzu muss GenParamName auf *'weight_prior'* und GenParamValue auf einen Wert > 0 gesetzt werden.

Falls das MLP für eine Regressionsaufgabe konfiguriert wurde (d.h., falls OutputFunction in create_class_mlp auf *'linear'* gesetzt wurde), kann die inverse Varianz des in den Daten zu erwartenden

Rauschens spezifiziert werden, indem GenParamName auf *'noise_prior'* und GenParamValue auf einen Wert > 0 gesetzt wird. Das Setzen der A-Priori-Varianz des Rauschens hat nur dann einen Effekt, falls auch eine A-Priori-Varianz für die MLP-Gewichte spezifiziert wurde. In diesem Fall kann die A-Priori-Varianz des Rauschens dazu verwendet werden, den Datenfehlerterm (den Ausgabefehler des MLPs) gegen den Gewichtsfehlerterm zu gewichten.

Wie weiter unten im Detail beschrieben, können die Regularisierungsparameter des MLPs automatisch bestimmt werden (auf Kosten von signifikant höheren Trainingszeiten), indem GenParamName auf *'num_outer_iterations'* und GenParamValue auf einen Wert > 0 gesetzt werden.

Technischer Hintergrund

Es werden drei verschiedene Arten von Straftermen bei *'weight_prior'* unterstützt. Im folgenden bezeichnen die Parameter $w_{ji}^{(l)}$ und $b_k^{(l)}$ die Gewichte der unterschiedlichen Schichten des MLPs, wie in create_class_mlp beschrieben.

Falls ein einzelner Wert α spezifiziert wird, werden alle MLP-Gewichte gleichmäßig bestraft, indem der folgende Term zur Optimierung in train_class_mlp hinzugefügt wird:

$$E_W = \frac{\alpha}{2}\left(\sum_{i=1}^{n_i}\sum_{j=1}^{n_h}\left(w_{ji}^{(1)}\right)^2 + \sum_{j=1}^{n_h}\left(b_j^{(1)}\right)^2 + \sum_{j=1}^{n_h}\sum_{k=1}^{n_o}\left(w_{kj}^{(2)}\right)^2 + \sum_{k=1}^{n_o}\left(b_k^{(2)}\right)^2\right)$$

Alternativ können vier Werte $[\alpha_{w1}, \alpha_{b1}, \alpha_{w2}, \alpha_{b2}]$ spezifiziert werden. Diese Parameter ermöglichen es, die vier Gruppen von Gewichten individuell zu regularisieren:

$$E_W = \frac{\alpha_{w1}}{2}\sum_{i=1}^{n_i}\sum_{j=1}^{n_h}\left(w_{ji}^{(1)}\right)^2 + \frac{\alpha_{b1}}{2}\sum_{j=1}^{n_h}\left(b_j^{(1)}\right)^2 + \frac{\alpha_{w2}}{2}\sum_{j=1}^{n_h}\sum_{k=1}^{n_o}\left(w_{kj}^{(2)}\right)^2 + \frac{\alpha_{b2}}{2}\sum_{k=1}^{n_o}\left(b_k^{(2)}\right)^2$$

Schließlich können $n_i + 3$ Werte $[\alpha_1, \ldots, \alpha_{n_i}, \alpha_{b1}, \alpha_{w2}, \alpha_{b2}]$ spezifiziert werden. Diese $n_i + 3$ Parameter ermöglichen es, die einzelnen Eingabevariablen $x_1, \ldots, x_{n_i}$ sowie die übrigen drei Gruppen von Gewichten individuell zu regularisieren:

$$E_W = \sum_{i=1}^{n_i}\frac{\alpha_i}{2}\sum_{j=1}^{n_h}\left(w_{ji}^{(1)}\right)^2 + \frac{\alpha_{b1}}{2}\sum_{j=1}^{n_h}\left(b_j^{(1)}\right)^2 + \frac{\alpha_{w2}}{2}\sum_{j=1}^{n_h}\sum_{k=1}^{n_o}\left(w_{kj}^{(2)}\right)^2 + \frac{\alpha_{b2}}{2}\sum_{k=1}^{n_o}\left(b_k^{(2)}\right)^2$$

Diese Art der Regularisierung ist nur in Verbindung mit der unten beschriebenen automatischen Bestimmung der Regularisierungsparameter sinnvoll. Falls die automatische Bestimmung der Regularisierungsparameter einen sehr großen Wert für α_j (im Vergleich zu dem kleinsten der n_i Werte α_i) zurückliefert, hat die entsprechende Eingabevariable eine geringe Relevanz für die Ausgabe des MLPs. Falls dies der Fall ist, sollte überprüft werden, ob die Eingabevariable aus der Eingabe des MLPs gelöscht werden kann, ohne die Leistung des MLPs negativ zu beeinflussen. Der Vorteil, irrelevante Eingabevariablen aus der Eingabe zu entfernen, ist eine höhere Geschwindigkeit des MLPs bei der Klassifikation.

Die Parameter α können als inverse Varianz einer Gaußschen A-Priori-Wahrscheinlichkeitsverteilung der MLP-Gewichte angesehen werden, d.h. sie drücken eine Erwartung der Größe der MLP-Gewichte aus. Je größer die α gewählt werden, desto kleiner werden die MLP-Gewichte.

Regularisierungsparameter

Je größer die Regularisierungsparameter *'weight_prior'* gewählt werden, desto glatter wird der Übergang der Konfidenzen zwischen den verschiedenen Klassen. Die notwendigen Werte der Regularisierungsparameter hängen vom MLP ab, speziell von der Anzahl der Einheiten in der versteckten Schicht, von den Trainingsdaten und von der Skalierung der Trainingsdaten (falls keine Normalisierung verwendet wird). Normalerweise ist ein größerer Wert der Regularisierungsparameter notwendig, falls das MLP mehr versteckte Einheiten hat und falls der Trainingsdatensatz mehr Punkte umfasst. In typischen Anwendungen werden die Regularisierungsparameter bestimmt, indem die Qualität des MLPs auf Testdaten, die unabhängig von den Trainingsdaten sind, überprüft wird. Falls kein unabhängiger Testdatensatz zur Verfügung steht, kann ein Kreuzvalidierungsverfahren verwendet werden. Kreuzvalidierungsverfahren teilen den Datensatz in zwei separate Teile auf (z.B. 80% der Daten für das Training und 20% der Daten für das Testen). Das MLP wird mit dem Trainingsdatensatz trainiert (die 80% der Daten im obigen Beispiel). Die Qualität des MLPs wird auf dem Testdatensatz (die 20% der Daten im obigen Beispiel) überprüft. Die Prozedur kann mit den anderen möglichen Aufteilungen der Daten wiederholt werden (in dem

80%–20%-Beispiel gibt es fünf mögliche Aufteilungen). Diese Prozedur kann z.B. mit relativ großen Werten der Gewichtsregularisierungsparameter starten (was normalerweise zu Fehlklassifikationen auf den Testdaten führt). Die Gewichtsregularisierungsparameter können dann sukzessive verkleinert werden, bis eine akzeptable Qualität auf den Testdatensätzen erreicht wird.

Automatische Bestimmung der Regularisierungsparameter

Die Regularisierungsparameter, d.h. die A-Priori-Gewichte und das A-Priori-Rauschen, können auch automatisch durch `train_class_mlp` mittels der sogenannten Evidenzprozedur bestimmt werden (für Details zur Evidenzprozedur wird auf die Artikel im Abschnitt „Literatur“ verwiesen). Dieser Trainingsmodus kann slektiert werden, indem `GenParamName` auf *'num_outer_iterations'* und `GenParamValue` auf einen Wert > 0 gesetzt werden. Es ist zu beachten, dass dies die Trainingszeiten um ein bis drei Größenordnungen im Vergleich zum Training mit festen Regularisierungsparametern erhöht.

Die Evidenzprozedur ist ein iterativer Algorithmus, der die folgenden zwei Schritte für eine Anzahl äußerer Iterationen durchführt: zunächst wird das Netz mit den aktuellen Werten der Regularisierungsparameter trainiert; danach werden die Regularisierungsparameter basierend auf den optimierten Gewichten des MLPs neu geschätzt. In der ersten Iteration werden die A-Priori-Gewichte und das A-Priori-Rauschen, die mit *'weight_prior'* und *'noise_prior'* spezifiziert wurden, verwendet. Die vom Benutzer spezifizierten Regularisierungsparameter dienen daher der automatischen Bestimmung als Startwerte für die Evidenzprozedur. Die Startwerte der A-Priori-Gewichte sollten nicht zu groß gesetzt werden, da dies das Training überregularisieren könnte und zu schlecht bestimmten Regularisierungsparametern führen könnte. Die Startwerte für die A-Priori-Gewichte sollten typischerweise in dem Bereich 0.01-0.1 gewählt werden.

Die Anzahl der äußeren Iterationen kann festgelegt werden, indem `GenParamName` auf *'num_outer_iterations'* und `GenParamValue` auf einen Wert > 0 gesetzt wird. Falls `GenParamValue` auf 0 gesetzt wird (dies ist die Voreinstellung des Wertes), wird die Evidenzprozedur nicht ausgeführt und das MLP wird einfach mit den benutzerspezifizierten Regularisierungsparametern trainiert.

Die Anzahl der äußeren Iterationen sollte groß genug gesetzt werden, um die Konvergenz der Regularisierungsparameter sicherzustellen. Im Gegensatz zum Training der Gewichte des MLPs ist es typischerweise sehr schwer, ein numerisches Konvergenzkriterium anzugeben und die Konvergenz wird typischerweise durch Beurteilung des Benutzers bestimmt. Daher kann es sein, dass es nicht möglich ist, die Anzahl der äußeren Iterationen a priori festzulegen und dabei die Konvergenz der Regularisierungsparameter sicherzustellen. In diesen Fällen kann die äußere Schleife über die Schritte der Evidenzprozedur explizit implementiert werden, indem *'num_outer_iterations'* auf 1 gesetzt wird und `train_class_mlp` wiederholt aufgerufen wird. Dies hat den Vorteil, dass die A-Priori-Gewichte und das A-Priori-Rauschen nach jeder Iteration abgefragt werden und manuell auf Konvergenz überprüft werden können. Mit diesem Ansatz kann sogar nach jeder Iteration die Leistung des MLPs auf einem unabhängigen Testdatensatz überprüft werden, um die Generalisierungseigenschaften des Klassifikators zu überprüfen.

Falls für eine bestimmte Klasse von Anwendungen die Anzahl der äußeren Iterationen (näherungsweise) bestimmt worden ist, kann versucht werden, die Laufzeit des Trainings zu reduzieren (falls MLPs mit ähnlichen zukünftig trainiert werden sollen), indem *'num_inner_iterations'* und `GenParamValue` auf einen Wert > 1 gesetzt wird (die Voreinstellung ist 1) und die Anzahl der äußeren Iterationen reduziert wird. Die Anzahl der äußeren Iterationen kann dabei aber typischerweise um denselben Faktor reduziert werden, wie die Anzahl der inneren Iterationen erhöht wird. Mit diesem Vorgehen kann eine optimale Laufzeit des Trainings erreicht werden. Dieses Vorgehen lohnt sich aber nur, falls viele MLPs mit ähnlichen Datensätzen trainiert werden sollen. Falls dies nicht der Fall ist, sollte *'num_inner_iterations'* auf 1 gelassen werden.

Die automatisch bestimmten A-Priori-Gewichte und das A-Priori-Rauschen können nach dem Training mit `get_regularization_params_class_mlp` abgefragt werden, indem `GenParamName` auf *'weight_prior'* bzw. auf *'noise_prior'* gesetzt wird.

Zusätzlich zu den A-Priori-Gewichten und dem A-Priori-Rauschen bestimmt die Evidenzprozedur eine Schätzung der Anzahl der Parameter des MLPs, die durch das Training anhand der Trainingsdaten eindeutig bestimmt werden können. Dieses Ergebnis kann mit `get_regularization_params_class_mlp` abgefragt werden, indem `GenParamName` auf *'num_well_determined_params'* gesetzt wird. Alternativ hierzu kann der Anteil der eindeutig bestimmbaren Parameter abgefragt werden, indem `GenParamName` auf *'fraction_well_determined_params'* gesetzt wird. Falls die Anzahl eindeutig bestimmbarer Parameter signifikant kleiner als n_w ist (wobei n_w die Anzahl der Gewichte des MLPs ist, wie im Abschnitt „Komplexität“ unten beschrieben) oder falls der Anteil der eindeutig bestimmbaren Parameter signifikant kleiner als 1 ist, sollte in Erwägung gezogen werden, die Anzahl der Einheiten in der versteckten Schicht zu reduzieren oder, falls sich die Anzahl der Einheiten in der versteckten Schicht nicht reduzieren lässt, ohne die Fehlerrate des MLPs signifikant zu erhöhen, eine Vorverarbeitung durchzuführen, die die Anzahl der Eingabevariablen reduziert, d.h. kanonische Merkmale oder eine Hauptachsentransformation.

Es ist zu beachten, dass die Anzahl der eindeutig bestimmbaren Parameter nur bestimmt werden kann, nachdem die A-Priori-Gewichten und das A-Priori-Rauschen bestimmt wurden. Dies ist der Grund, warum die Evidenzprozedur mit der Bestimmung der Regularisierungsparameter endet und nicht mit dem Training der MLP-Gewichte. Das MLP wird im Rahmen der Evidenzprozedur daher nicht mit den letzten Werten der Regularisierungsparameter trainiert. Dies sollte keine Auswirkung haben, sofern die Regularisierungsparameter konvergiert sind. Falls das Training mit einer Optimierung der MLP-Gewichte enden soll, wobei die letzten Werte der Regularisierungsparameter verwendet werden, kann *'num_outer_iterations'* auf 0 gesetzt werden und `train_class_mlp` noch einmal aufgerufen werden. Falls dies gemacht wird, ist zu beachten, dass sich die Anzahl der eindeutig bestimmbaren Parameter ändern kann und somit der von `get_regularization_params_class_mlp` zurückgelieferte Wert technisch gesehen inkonsistent ist.

Abgespeicherte Parameter

Es ist zu beachten, dass die Parameter *'num_outer_iterations'* und *'num_inner_iterations'* nur das Training des MLPs beeinflussen. Daher werden sie nicht gespeichert, wenn das MLP mit `write_class_mlp` oder `serialize_class_mlp` gespeichert wird. Sie müssen daher neu gesetzt werden, falls das MLP mit `read_class_mlp` oder `deserialize_class_mlp` wieder geladen wird und falls das Training unter der automatischen Bestimmung der Regularisierungsparameter fortgesetzt werden soll. Alle anderen oben beschriebenen Parameter (*'weight_prior'*, *'noise_prior'*, *'num_well_determined_params'* und *'fraction_well_determined_params'*) werden gespeichert.

Parameter

▷ **`MLPHandle`** (input_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

▷ **`GenParamName`** (input_control) . . . . . string ⇝ *string*
Name des zu setzenden Regularisierungsparameters.
Defaultwert: 'weight_prior'
Werteliste: `GenParamName` ∈ {'weight_prior', 'noise_prior', 'num_outer_iterations', 'num_inner_iterations'}

▷ **`GenParamValue`** (input_control) . . . . . number(-array) ⇝ *real* / integer
Wert des Regularisierungsparameters.
Defaultwert: 1.0
Wertevorschläge: `GenParamValue` ∈ {0.01, 0.1, 1.0, 10.0, 100.0, 0, 1, 2, 3, 5, 10, 15, 20}

Beispiel

```
* This example shows how to determine the regularization parameters
* automatically without examining the convergence of the
* regularization parameters.
* Create the MLP.
create_class_mlp (NumIn, NumHidden, NumOut, 'softmax', \
                  'normalization', NumIn, 42, MLPHandle)
* Generate and add the training data
for J := 0 to NumData-1 by 1
    * Generate training features and classes.
    * Data = [...]
    * Class = [...]
    add_sample_class_mlp (MLPHandle, Data, Class)
endfor
* Set up the automatic determination of the regularization
* parameters.
set_regularization_params_class_mlp (MLPHandle, 'weight_prior', \
                                     [0.01,0.01,0.01,0.01])
set_regularization_params_class_mlp (MLPHandle, \
                                     'num_outer_iterations', 10)
* Train the MLP.
train_class_mlp (MLPHandle, 100, 1, 0.01, Error, ErrorLog)
* Read out the estimate of the number of well-determined
* parameters.
get_regularization_params_class_mlp (MLPHandle, \
                                     'fraction_well_determined_params', \
```

```
                                        FractionParams)
* If FractionParams differs substantially from 1, consider reducing
* NumHidden appropriately and consider performing a preprocessing that
* reduces the number of input variables to the net, i.e., canonical
* variates or principal components.
write_class_mlp (MLPHandle, 'classifier.mlp')



* This example shows how to determine the regularization parameters
* automatically while examining the convergence of the
* regularization parameters.
* Create the MLP.
create_class_mlp (NumIn, NumHidden, NumOut, 'softmax', \
                  'normalization', NumIn, 42, MLPHandle)
* Generate and add the training data.
for J := 0 to NumData-1 by 1
    * Generate training features and classes
    * Data = [...]
    * Class = [...]
    add_sample_class_mlp (MLPHandle, Data, Class)
endfor
* Set up the automatic determination of the regularization
* parameters.
set_regularization_params_class_mlp (MLPHandle, 'weight_prior', \
                                     [0.01,0.01,0.01,0.01])
set_regularization_params_class_mlp (MLPHandle, \
                                     'num_outer_iterations', 1)
for OuterIt := 1 to 10 by 1
    * Train the MLP
    train_class_mlp (MLPHandle, 100, 1, 0.01, Error, ErrorLog)
    * Read out the regularization parameters
    get_regularization_params_class_mlp (MLPHandle, 'weight_prior', \
                                         WeightPrior)
    * Inspect the regularization parameters manually for
    * convergence and exit the loop manually if they have
    * converged.
    * [...]
endfor
* Read out the estimate of the number of well-determined
* parameters.
get_regularization_params_class_mlp (MLPHandle,\
                                     'fraction_well_determined_params',\
                                     FractionParams)
* If FractionParams differs substantially from 1, consider reducing
* NumHidden appropriately and consider performing a preprocessing that
* reduces the number of input variables to the net, i.e., canonical
* variates or principal components.
write_class_mlp (MLPHandle, 'classifier.mlp')
```

Komplexität

Sei n_i die Anzahl der Eingabeeinheiten des MLPs (d.h. n_i = `NumHidden` oder n_i = `NumComponents`, je nachdem, auf welchen Wert `Preprocessing` gesetzt wurde, wie bei `create_class_mlp` beschrieben), n_h die Anzahl der Einheiten in der versteckten Schicht und n_o die Anzahl der Ausgabeeinheiten. Dann ist die Anzahl der Gewichte des MLPs gegeben durch $n_w = (n_i + 1)n_h + (n_h + 1)n_o$. Sei n_d die Anzahl der Trainingsmuster. Sei n_M die Anzahl der Iterationen, die mit `MaxIterations` in `train_class_mlp` gesetzt wird. Seien n_O und n_I die Anzahl der äußeren bzw. inneren Iterationen.

Die Laufzeit des Trainings ohne Regularisierung oder mit Regularisierung mit festen Regularisierungsparametern ist von der Komplexität $O(n_M n_w n_d)$. Im Gegensatz hierzu ist die Laufzeit des Trainings mit automatischer Be-

stimmung der Regularisierungsparameter von der Komplexität $O(n_O n_M n_w n_d) + O(n_O n_w^2 n_d) + O(n_O n_w^3) + O(n_O n_I n_w^3)$.

Das Training ohne Regularisierung oder mit Regularisierung mit festen Regularisierungsparametern benötigt mindestens $48n_w + 24n_h n_d + 16n_o n_d$ Bytes Speicher. Das Training mit automatischer Bestimmung der Regularisierungsparameter benötigt mindestens $24n_w^2 + 48n_w + 72n_h n_d + 56n_o n_d$ Bytes Speicher. Unter speziellen Umständen werden weitere $24n_w^2 + 8n_w$ Bytes Speicher benötigt.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_regularization_params_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_mlp`

Nachfolger

`get_regularization_params_class_mlp`, `train_class_mlp`

Literatur

David J. C. MacKay: „Bayesian Interpolation“; Neural Computation 4(3):415-447; 1992.

David J. C. MacKay: „A Practical Bayesian Framework for Backpropagation Networks“; Neural Computation 4(3):448-472; 1992.

David J. C. MacKay: „The Evidence Framework Applied to Classification Networks“; Neural Computation 4(5): 720-736; 1992.

David J. C. MacKay: „Comparison of Approximate Methods for Handling Hyperparameters“; Neural Computation 11(5):1035-1068; 1999.

Modul

Foundation

set_rejection_params_class_mlp (: : MLPHandle, GenParamName, GenParamValue :)

Setzen der Parameter einer Rückweisungsklasse.

`set_rejection_params_class_mlp` setzt die Parameter einer automatisch generierten Rückweisungsklasse innerhalb eines mehrschichtigen Perzeptrons (MLP), das in `MLPHandle` übergeben wurde. In einigen Anwendungen ist es erstrebenswert zu wissen, ob ein Merkmalsvektor ähnlich zu einem aus der Trainingsmenge ist. Falls ein Merkmalsvektor außerhalb der zur Verfügung stehenden Trainingsmenge liegt, soll er als eine spezielle Rückweisungsklasse klassifiziert werden. Das bedeutet, dass der Merkmalsvektor sich von dem Konfidenzbereich des Klassifikators unterscheidet. Falls man die Benutzung einer zusätzlichen Rückweisungsklasse beabsichtigt, muss eine zusätzliche Klasse beim Erzeugen des Klassifikators in `create_class_mlp` angegeben werden. Hier muss entsprechend der Parameter `NumOutput` um eins erhöht werden.

Der Parameter der Rückweisungsklasse werden mit `GenParamName` gewählt, der entsprechende Wert mit `GenParamValue`.

'rejection_class_index': Standardmäßig dient die letzte Klasse als Rückweisungsklasse. Falls eine andere Klasse benützt werden soll, muss `GenParamName` auf *'rejection_class_index'* und `GenParamValue` auf den Klassenindex gesetzt werden.

'sampling_strategy': Zur Zeit existieren drei Strategien, um Datensätze der Rückweisungsklasse zu erzeugen. Diese Strategien können durch Setzen von GenParamName auf *'sampling_strategy'* und GenParamValue auf *'hyperbox_around_all_classes'*, *'hyperbox_around_each_class'* oder *'hyperbox_ring_around_each_class'* gesetzt werden. Die Abtaststrategie *'hyperbox_around_all_classes'* nimmt das umschließende Rechteck um alle Trainingsdaten, die bisher angegeben wurden. Die Abtaststrategie *'hyperbox_around_each_class'* ist ähnlich, mit dem Unterschied, dass das umschließende Rechteck um jede Klasse einzeln erzeugt wird, aus dem dann die Rückweisungsdaten generiert werden. Die Abtaststrategie *'hyperbox_ring_around_each_class'* generiert Daten nur in dem vergrößerten Bereich um das umschließende Rechteck um jede Klasse, sozusagen nur einen Ring um die Originaldaten. Es ist zu beachten, dass mit steigender Dimensionaliät die Abtaststrategien *'hyperbox_around_each_class'* und *'hyperbox_ring_around_each_class'* das gleiche Ergebnis liefern. Falls keine Rückweisungsklasse verwendet werden soll, was der Standardfall ist, muss GenParamValue auf *'no_rejection_class'* gesetzt werden.

'hyperbox_tolerance': Der Faktor *'hyperbox_tolerance'* beschreibt, um welchen das umschließende Rechteck in alle Dimensionen vergrößert werden soll. Innerhalb dieses Rechtecks werden zufällig Daten mit einer gleichverteilten Wahrscheinlichkeit erzeugt. Der Standartwert beträgt 0.2.

'rejection_sample_factor': Die Anzahl von Rückweisungsdaten ist die Anzahl von übergebenen Daten multipliziert mit *'rejection_sample_factor'*. Falls nicht genügend Daten erzeugt werden, wird die Rückweisungsklasse nicht korrekt klassifiziert. Falls die Rückweisungsklasse zu viele Daten enthält, werden normale Klassen als Rückweisungsklasse klassifiziert. Der Standardwert beträgt 1.0.

'random_seed': Um reproduzierbare Ergebnisse sicherzustellen, kann mit *'random_seed'* ein Anfangswert für den Zufallsgenerator gesetzt werden. Der Standardwert beträgt 42.

Da dieser Operator nur das Training des MLP parametrisiert, werden die Werte nicht mit write_class_mlp gespeichert.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'sampling_strategy'
Werteliste: GenParamName ∈ {'sampling_strategy', 'hyperbox_tolerance', 'rejection_sample_factor', 'random_seed', 'rejection_class_index'}

▷ **GenParamValue** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
Wert des generischen Parameters.
Defaultwert: 'hyperbox_around_all_classes'
Werteliste: GenParamValue ∈ {'no_rejection_class', 'hyperbox_around_all_classes', 'hyperbox_around_each_class', 'hyperbox_ring_around_each_class'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_rejection_params_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_class_mlp

Nachfolger

train_class_mlp

Modul

Foundation

train_class_mlp (: : MLPHandle, MaxIterations, WeightTolerance, ErrorTolerance : Error, ErrorLog)

Trainieren eines mehrschichtigen Perzeptrons.

train_class_mlp trainiert das durch MLPHandle angegebene mehrschichtige Perzeptron (MLP). Bevor das MLP trainiert werden kann, müssen mit add_sample_class_mlp oder read_samples_class_mlp *alle* zum Training zu verwendenden Trainingsmuster in dem MLP abgespeichert werden. Falls nach dem Training neue zusätzliche Trainingsmuster zum Trainieren verwendet werden sollen, muss ein neues MLP mit create_class_mlp erzeugt werden, bei dem wiederum *alle* zum Training zu verwendenden Trainingsmuster (also die ursprünglichen und die zusätzlich zu verwendenden) abgespeichert werden. Hierbei empfiehlt sich die Verwendung von write_samples_class_mlp und read_samples_class_mlp. Ein Nachtrainieren mit neuen Trainingsmustern ist programmtechnisch zwar nicht verboten, führt aber typischerweise nicht zu guten Ergebnissen, da das Training eines MLP ein komplexes nichtlineares Optimierungsproblem ist, und das mehrmalige Trainieren mit neuen Trainingsdaten mit hoher Wahrscheinlichkeit dazu führt, dass die Optimierung in einem lokalen Minimum steckenbleibt.

Falls eine Rückweisungsklasse mittels set_rejection_params_class_mlp spezifiziert wurde, werden vor dem eigentlichen Training die Trainingsdaten für die Rückweisungsklasse erzeugt.

Beim Training wird der Fehler, den das MLP auf den gespeicherten Trainingsdaten erzielt, durch ein nichtlineares Optimierungsverfahren minimiert. Falls das MLP mittels set_regularization_params_class_mlp regularisiert wurde, wird ein zusätzlicher Gewichtsstrafterm berücksichtigt. Dadurch werden die bei create_class_mlp beschriebenen Gewichte des MLP bestimmt. Falls mittels set_regularization_params_class_mlp eine automatische Bestimmung der Regularisierungsparameter spezifiziert wurde, werden auch diese Parameter optimiert. Wie bei set_regularization_params_class_mlp beschrieben, benötigt das Training eines MLPs bei gleichzeitiger Bestimmung der Regularisierungsparameter signifikant mehr Zeit als das Training eines unregularisierten MLPs oder eines MLPs mit festen Regularisierungsparametern.

Als Startwerte werden die Gewichte von create_class_mlp auf zufällige Werte gesetzt, um mit hoher Wahrscheinlichkeit zu ermöglichen, dass die Optimierung zum globalen Minimum der Fehlerfunktion konvergiert. Nichtsdestotrotz kann es eventuell dazu kommen, dass mit den durch RandSeed in create_class_mlp festgelegten Werten ein relativ hoher Fehler als optimaler Wert bestimmt wird, d.h., dass die Optimierung in einem lokalen Minimum steckenbleibt. Falls vermutet werden kann, dass dies passiert ist, sollte ein neues Netz mit einem anderen Wert für RandSeed erzeugt werden, um zu überprüfen, ob ein signifikant kleinerer Fehler erzeugt werden kann.

Die Parameter MaxIterations, WeightTolerance und ErrorTolerance dienen der Steuerung des nichtlinearen Optimierungsverfahrens. Es ist zu beachten, dass diese Parameter sich auf ein Training innerhalb eines Schrittes der Evidenzprozedur beziehen, falls mittels set_regularization_params_class_mlp eine automatische Bestimmung der Regularisierungsparameter spezifiziert wurde. MaxIterations spezifiziert die maximale Anzahl von Iterationen des Optimierungsverfahrens. In der Praxis sollten Werte zwischen *100* und *200* ausreichend für die meisten Probleme sein. WeightTolerance spezifiziert eine Schwelle für die Änderung der Gewichte pro Iteration. Dabei wird der Betrag der Änderung der Gewichte aufsummiert. Dieser Wert hängt also von der Anzahl der Gewichte im MLP und der Größe der Gewichte ab, welche wiederum von der Skalierung der Eingabedaten abhängt. Typischerweise sollten hier Werte zwischen *0.00001* und *1* vorgegeben werden. ErrorTolerance spezifiziert eine Schwelle für die Änderung des Fehlerwertes pro Iteration. Hier hängt der Wert von der Anzahl der Trainingsdaten und der Anzahl der Ausgabevariablen ab. Auch hier sollten typischerweise Werte zwischen *0.00001* und *1* vorgegeben werden. Die Optimierung wird beendet, falls die Gewichtsänderung in einer Iteration unter WeightTolerance und die Änderung des Fehlers unter ErrorTolerance liegt. Spätestens nach MaxIterations Iterationen wird die Optimierung aber auf jeden Fall beendet. Es ist zu beachten, dass je nach Größe des MLP und der Anzahl der Trainingsmuster das Trainieren zwischen wenigen Sekunden und einigen Stunden dauern kann.

Als Ausgabe liefert train_class_mlp in Error den Fehler mit den optimalen Gewichten des MLP auf den Trainingsdaten zurück. Außerdem wird in ErrorLog der Fehler des MLP pro Iteration zurückgeliefert. Somit ist Error der letzte Wert von ErrorLog. Der Verlauf des Fehlers in ErrorLog kann dazu verwendet werden, zu entscheiden, ob ein nochmaliges Training des MLP mit train_class_mlp mit denselben

Trainingsdaten ohne das MLP neu zu erzeugen sinnvoll ist. Als Funktion betrachtet sollte ErrorLog am Anfang steil abfallen und zum Ende hin sehr flach werden. Falls ErrorLog am Ende noch relativ steil ist, ist normalerweise ein erneuter Aufruf von train_class_mlp sinnvoll. Es ist zu beachten, dass dieser Mechanismus *nicht* dazu verwendet werden sollte, das MLP jeweils mit MaxIterations = *1* (oder anderen sehr kleinen Werten von MaxIterations) zu trainieren, da dadurch die Anzahl der nötigen Iterationen erheblich steigt. Falls mittels set_regularization_params_class_mlp eine automatische Bestimmung der Regularisierungsparameter spezifiziert wurde, beziehen sich Error und ErrorLog auf das letzte Training, das innerhalb der Evidenzprozedur ausgeführt wurde. Falls der Fehlerverlauf innerhalb der einzelnen Iterationen der Evidenzprozedur beobachtet werden soll, muss die äußere Iteration der Evidenzprozedur explizit, wie bei set_regularization_params_class_mlp beschrieben, implementiert werden.

Parameter

- ▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
  Handle des MLP.
- ▷ **MaxIterations** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl von Iterationen des Optimierungsverfahrens.
  Defaultwert: 200
  Wertevorschläge: MaxIterations ∈ {20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 260, 280, 300}
- ▷ **WeightTolerance** (input_control) . . . . . real ⇝ *real*
  Schwellenwert für den Unterschied der Gewichte des MLP zwischen zwei Iterationen des Optimierungsverfahrens.
  Defaultwert: 1.0
  Wertevorschläge: WeightTolerance ∈ {1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001}
  Restriktion: WeightTolerance >= 1.0e-8
- ▷ **ErrorTolerance** (input_control) . . . . . real ⇝ *real*
  Schwellenwert für den Unterschied des mittleren Fehlers des MLP auf den Trainingsdaten zwischen zwei Iterationen des Optimierungsverfahrens.
  Defaultwert: 0.01
  Wertevorschläge: ErrorTolerance ∈ {1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001}
  Restriktion: ErrorTolerance >= 1.0e-8
- ▷ **Error** (output_control) . . . . . real ⇝ *real*
  Mittlerer Fehler des MLP auf den Trainingsdaten.
- ▷ **ErrorLog** (output_control) . . . . . real-array ⇝ *real*
  Mittlerer Fehler des MLP auf den Trainingsdaten pro Iteration des Optimierungsverfahrens.

Beispiel

```
* Train an MLP
create_class_mlp (NumIn, NumHidden, NumOut, 'softmax', \
                  'normalization', 1, 42, MLPHandle)
read_samples_class_mlp (MLPHandle, 'samples.mtf')
train_class_mlp (MLPHandle, 100, 1, 0.01, Error, ErrorLog)
write_class_mlp (MLPHandle, 'classifier.mlp')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert train_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls train_class_mlp den Fehler 9211 (Matrix ist nicht positiv definit) bei Preprocessing = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`add_sample_class_mlp`, `read_samples_class_mlp`, `set_regularization_params_class_mlp`

Nachfolger

`evaluate_class_mlp`, `classify_class_mlp`, `write_class_mlp`, `create_class_lut_mlp`

Alternativen

`train_dl_classifier_batch`, `read_class_mlp`

Siehe auch

`create_class_mlp`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Andrew Webb: „Statistical Pattern Recognition“; Arnold, London; 1999.

Modul

Foundation

write_class_mlp (: : MLPHandle, FileName :)

Abspeichern eines mehrschichtigen Perzeptrons in Datei.

`write_class_mlp` speichert das mehrschichtige Perzeptron (MLP) `MLPHandle` in der Datei `FileName` ab. Die Dateiendung für den MLP-Klassifikator ist in HALCON 'gmc'. `write_class_mlp` wird typischerweise aufgerufen, nachdem das MLP mit `train_class_mlp` trainiert wurde. Das MLP kann mit `read_class_mlp` wieder eingelesen werden. `write_class_mlp` speichert *nicht* die im MLP eventuell abgespeicherten Trainingsmuster ab. Dazu ist `write_samples_class_mlp` zu verwenden.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.gmc`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_mlp`

Nachfolger

`clear_class_mlp`

Siehe auch

`create_class_mlp`, `read_class_mlp`, `write_samples_class_mlp`

Modul

Foundation

write_samples_class_mlp (: : MLPHandle, FileName :)

Abspeichern der Trainingsdaten eines mehrschichtigen Perzeptrons in Datei.

write_samples_class_mlp speichert die in dem mehrschichtigen Perzeptron (MLP) MLPHandle abgespeicherten Trainingsmuster in der Datei FileName ab. write_samples_class_mlp kann dazu verwendet werden, eine Datenbank mit Trainingsmustern aufzubauen, und somit durch erneutes Trainieren mit einer erweiterten Datenbank die Leistung des MLP zu verbessern (siehe train_class_mlp). Für weitere mögliche Verwendungen von write_samples_class_mlp siehe get_prep_info_class_mlp.

Die Datei FileName wird von write_samples_class_mlp überschrieben. Eine Erweiterung der Datenbank der Trainingsmuster ist aber einfach möglich, da read_samples_class_mlp und add_sample_class_mlp die Trainingsmuster zu den bereits im Speicher des MLP gehaltenen Trainingsmustern hinzufügen.

Parameter

▷ **MLPHandle** (input_control) . . . . . class_mlp ⇝ *handle*
Handle des MLP.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_samples_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_mlp

Nachfolger

clear_samples_class_mlp

Siehe auch

create_class_mlp, get_prep_info_class_mlp, read_samples_class_mlp

Modul

Foundation

15.5 Sonstiges

add_sample_class_train_data (: : ClassTrainDataHandle, Order, Features, ClassID :)

Hinzufügen neuer Merkmalsvektoren zu Trainingsdaten.

add_sample_class_train_data fügt neue Merkmalsvektoren zu den Trainingsdaten ClassTrainDataHandle hinzu. Die neuen Merkmalsvektoren werden in Features mit der zugehörigen Klasse in ClassID übergeben. Features hat die Länge NumDim, die beim Anlegen der Trainingsdaten mit z.B. create_class_train_data angegeben wurde. Es kann mehr als ein Merkmalsvektor auf einmal hinzugefügt werden. In diesem Fall legt der Parameter Order die Reihenfolge fest, in der die Werte in Features angeordnet sind. Wenn Order auf *'row'* gesetzt ist, werden die Merkmalsvektoren zeilenweise angegeben (erst der erste Merkmalsvektor, dann der Zweite, usw). Wenn Order auf *'column'* gesetzt ist, wird angenommen, dass zuerst alle Werte der ersten Dimension für alle Merkmalsvektoren angegeben werden, und erst dann die zweite Dimension, usw. Der dritte mögliche Modus in Order ist *'feature_column'*. Dieser Modus erwartet zusammengehörige Merkmale (siehe set_feature_lengths_class_train_data) zeilenweise

aufeinander folgend, d.h. zuerst kommen alle Werte des ersten Merkmals mit seiner entsprechenden Länge für alle Merkmalsvektoren, dann folgt entsprechend das zweite Merkmal, usw.

Parameter

▷ **ClassTrainDataHandle** (input_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten.

▷ **Order** (input_control) . . . . . string ⇝ *string*
Die Reihenfolge des Merkmalsvektor.
Defaultwert: 'row'
Werteliste: `Order` ∈ {'row', 'column', 'feature_column'}

▷ **Features** (input_control) . . . . . number-array ⇝ *real*
Merkmalsvektor.

▷ **ClassID** (input_control) . . . . . integer-array ⇝ *integer*
Klasse des Merkmalsvektors.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_sample_class_train_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassTrainDataHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_train_data`

Nachfolger

`add_class_train_data_svm`, `add_class_train_data_knn`, `add_class_train_data_gmm`, `add_class_train_data_mlp`

Siehe auch

`create_class_train_data`

Modul

Foundation

clear_class_train_data (: : ClassTrainDataHandle :)

Löscht Trainingsdaten.

`clear_class_train_data` löscht die Trainingsdaten die mit `ClassTrainDataHandle` übergeben wurden. Nach dem Aufruf von `clear_class_train_data`, ist `ClassTrainDataHandle` ungültig.

Parameter

▷ **ClassTrainDataHandle** (input_control) . . . . . class_train_data ⇝ *handle*
Handle von Trainingsdaten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `clear_class_train_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassTrainDataHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_train_data`

Siehe auch

`create_class_train_data`

Modul

Foundation

create_class_train_data (: : NumDim : ClassTrainDataHandle)

Erzeugen eines Handles für Trainingsdaten für Klassifikatoren.

`create_class_train_data` erzeugt ein Handle für Trainingsdaten für Klassifikatoren. Das Handle wird in `ClassTrainDataHandle` zurückgegeben. Die Länge der Merkmalsvektoren muss hier mit Hilfe von `NumDim` festgelegt werden.

Parameter

▷ **NumDim** (input_control) ..... number ⇝ *integer*
Länge eines Merkmalsvektors in den Trainingsdaten.
Defaultwert: 10

▷ **ClassTrainDataHandle** (output_control) ..... class_train_data ⇝ *handle*
Handle der Trainingsdaten.

Beispiel

```
* Find out which of the two features distinguishes two Classes
NameFeature1 := 'Good Feature'
NameFeature2 := 'Bad Feature'
LengthFeature1 := 3
LengthFeature2 := 2
* Create training data
create_class_train_data (LengthFeature1+LengthFeature2,\
  ClassTrainDataHandle)
* Define the features which are in the training data
set_feature_lengths_class_train_data (ClassTrainDataHandle, [LengthFeature1,\
  LengthFeature2], [NameFeature1, NameFeature2])
* Add training data
*                                                        |Feat1|  |Feat2|
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  2,1  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  2,1  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  3,4  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  3,4  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [0,0,1,  5,6  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,3,2,  5,6  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [0,0,1,  5,6  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,3,2,  5,6  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [0,0,1,  5,6  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,3,2,  5,6  ], 1)
```

```
* Add more data
* ...
* Select the better feature with the classifier of your choice
select_feature_set_knn (ClassTrainDataHandle, 'greedy', [], [], KNNHandle,\
  SelectedFeature, Score)
select_feature_set_svm (ClassTrainDataHandle, 'greedy', [], [], SVMHandle,\
  SelectedFeature, Score)
select_feature_set_mlp (ClassTrainDataHandle, 'greedy', [], [], MLPHandle,\
  SelectedFeature, Score)
select_feature_set_gmm (ClassTrainDataHandle, 'greedy', [], [], GMMHandle,\
  SelectedFeature, Score)
* Use the classifier
* ...
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_class_train_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`add_sample_class_knn`, `train_class_knn`

Alternativen

`create_class_svm`, `create_class_mlp`

Siehe auch

`select_feature_set_knn`, `read_class_knn`

Modul

Foundation

deserialize_class_train_data (: : SerializedItemHandle : ClassTrainDataHandle)

Deserialisiert Trainingsdaten für Klassifikatoren.

`deserialize_class_train_data` deserialisiert Trainingsdaten für Klassifikatoren, welche mit `serialize_class_train_data` serialisiert wurden (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Die serialisierten Trainingsdaten werden in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Handle `ClassTrainDataHandle` gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **ClassTrainDataHandle** (output_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_class_train_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

serialize_class_train_data

Nachfolger

fwrite_serialized_item

Siehe auch

create_class_train_data

Modul

Foundation

get_sample_class_train_data (: : ClassTrainDataHandle, IndexSample : Features, ClassID)

Gibt einen bestimmten Merkmalsvektor aus Trainingsdaten zurück.

get_sample_class_train_data gibt einen durch IndexSample bestimmten Merkmalsvektor aus den Trainingsdaten ClassTrainDataHandle zurück. Der Merkmalsvektor muss vorher z.B. durch add_sample_class_train_data hinzugefügt worden sein. Der Index der in IndexSample angegeben wird, muss zwischen 0 und der Anzahl an verfügbaren Merkmalsvektoren liegen. Als Ergebnis enthält Features den Merkmalsvektor und ClassID die zugehörige Klasse.

Parameter

- ▷ **ClassTrainDataHandle** (input_control) ..... class_train_data ⇝ *handle*
  Trainingsdaten.
- ▷ **IndexSample** (input_control) ..... integer ⇝ *integer*
  Index des Merkmalsvektors.
- ▷ **Features** (output_control) ..... real-array ⇝ *real*
  Merkmalsvektor.
- ▷ **ClassID** (output_control) ..... integer ⇝ *integer*
  Klasse des Merkmalsvektors.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sample_class_train_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_train_data

Siehe auch

create_class_train_data

Modul

Foundation

get_sample_num_class_train_data (: : ClassTrainDataHandle : NumSamples)

Gibt die Anzahl der verfügbaren Merkmalsvektoren in Trainingsdaten zurück.

get_sample_num_class_train_data gibt in NumSamples die Anzahl an verfügbaren Merkmalsvektoren in den Trainingsdaten ClassTrainDataHandle zurück.

Parameter

▷ **ClassTrainDataHandle** (input_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten.

▷ **NumSamples** (output_control) . . . . . integer ⇝ *integer*
Anzahl der verfügbaren Merkmalsvektoren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sample_num_class_train_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_train_data

Nachfolger

get_sample_class_train_data

Siehe auch

create_class_train_data

Modul

Foundation

read_class_train_data (: : FileName : ClassTrainDataHandle)

Lesen von Trainingsdaten aus einer Datei.

read_class_train_data liest die gespeicherten Trainingsdaten aus der Datei FileName (siehe write_class_train_data). Die Dateiendung für Trainingsdaten in HALCON ist 'ctd'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname der Trainingsdaten.
Dateiendung: .ctd

▷ **ClassTrainDataHandle** (output_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_class_train_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Siehe auch

create_class_train_data, write_class_train_data

Modul

Foundation

select_sub_feature_class_train_data (: : ClassTrainDataHandle, SubFeatureIndices : SelectedClassTrainDataHandle)

Wählt eine Untermenge der definierten Merkmale in Trainingsdaten aus.

select_sub_feature_class_train_data wählt eine Untermenge der definierten Merkmale in den Trainingsdaten ClassTrainDataHandle aus in gibt diese Untermenge wieder als neue Trainingsdaten in SelectedClassTrainDataHandle zurück. Die Untermenge kann mit SubFeatureIndices angegeben werden. Wurde set_feature_lengths_class_train_data zuvor noch nicht aufgerufen, werden standardmäßig die einzelnen Spalten als Merkmal betrachtet und man kann diese über eine Liste mit Indizes auswählen.

Durch einen vorherigen Aufruf von set_feature_lengths_class_train_data kann eine Gruppierung definiert werden, die mehrere benachbarte Spalten gruppieren kann. War das der Fall, werden statt Spalten diese Gruppen indiziert und durch Angabe der entsprechenden Indizes ausgewählt. Sollten zusätzlich Namen für diese Gruppen angegeben worden sein, können diese ebenfalls verwendet werden, um einzelne Gruppen auszuwählen.

Parameter

- ▷ **ClassTrainDataHandle** (input_control) ..... class_train_data ⇝ *handle*
  Trainingsdaten.
- ▷ **SubFeatureIndices** (input_control) ..... number-array ⇝ *integer* / string
  Indizes oder Namen um Untermerkmale auszuwählen.
- ▷ **SelectedClassTrainDataHandle** (output_control) ..... class_train_data ⇝ *handle*
  Ausgewählte Untermenge der Trainingsdaten.

Beispiel

```
* Find out which of the two features distinguishes two Classes
NameFeature1 := 'Good Feature'
NameFeature2 := 'Bad Feature'
LengthFeature1 := 3
LengthFeature2 := 2
* Create training data
create_class_train_data (LengthFeature1+LengthFeature2,\
                         ClassTrainDataHandle)
* Define the features which are in the training data
set_feature_lengths_class_train_data (ClassTrainDataHandle, [LengthFeature1,\
                           LengthFeature2], [NameFeature1, NameFeature2])
* Add training data
*                                                          |Feat1| |Feat2|
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  2,1  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  2,1  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  3,4  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  3,4  ], 1)
* Add more data
* ...
* Select one of the features
select_sub_feature_class_train_data (ClassTrainDataHandle, NameFeature1, \
                                     SelectedClassTrainDataHandle)
* Add training data to a classifier
create_class_knn (LengthFeature1, KNNHandle)
add_class_train_data_knn (KNNHandle, SelectedClassTrainDataHandle)
train_class_knn (KNNHandle, [], [])
* Use the classifier
* ...
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert select_sub_feature_class_train_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_class_train_data`, `add_sample_class_train_data`, `set_feature_lengths_class_train_data`

Nachfolger

`add_class_train_data_gmm`, `add_class_train_data_mlp`, `add_class_train_data_svm`, `add_class_train_data_knn`

Modul

Foundation

`serialize_class_train_data` (: : `ClassTrainDataHandle` : `SerializedItemHandle`)

Serialisiert Trainingsdaten für Klassifikatoren.

`serialize_class_train_data` serialisiert Trainingsdaten für Klassifikatoren (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_class_train_data` in eine Datei geschrieben werden. Die Trainingsdaten werden mit dem Handle `ClassTrainDataHandle` angegeben. Die serialisierten Trainingsdaten werden in dem Handle `SerializedItemHandle` zurückgegeben und können mit `deserialize_class_train_data` wieder deserialisiert werden.

Parameter

▷ **`ClassTrainDataHandle`** (input_control) ..... class_train_data ⇝ *handle*
Handle der Trainingsdaten.

▷ **`SerializedItemHandle`** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_class_train_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`deserialize_class_train_data`

Siehe auch

`create_class_train_data`, `read_class_train_data`

Modul

Foundation

`set_feature_lengths_class_train_data` (: : `ClassTrainDataHandle`, `SubFeatureLength`, `Names` :)

Definiert benachbarte Spalten in Trainingsdaten als zusammengehörige Merkmalsgruppen.

`set_feature_lengths_class_train_data` definiert benachbarte Spalten in den Trainingsdaten `ClassTrainDataHandle` als zusammengehörige Merkmale. Es können dafür die Längen der jeweiligen Merkmalsgruppen mit `SubFeatureLength` angegeben werden. Das bedeutet, dass die Summe aller Werte in

SubFeatureLength der Anzahl der Spalten in den Trainingsdaten entsprechen muss. Die Anzahl der Spalten wird beim Aufruf von create_class_train_data im Parameter NumDim angegeben. Zusätzlich zu den Gruppierungen kann für jede Gruppierung ein Name in Names angegeben werden.

Beispielsweise kann dieser Operator nützlich sein um das folgende Problem zu beschreiben: Es sind zwei verschiedene Datenquellen für ein bestimmtes Klassifikationsproblem verfügbar, beide liefern pro Beobachtung einen Vektor einer bestimmten Länge, z.B. liefert die erste Datenquelle Vektoren der Länge n und die andere Datenquelle Vektoren der Länge m. Um zu entscheiden, welche der Datenquellen nützlich für das Klassifikationsproblem ist, sollte man Trainingsdaten dieser zwei Quellen folgendermaßen erzeugen: man sollte create_class_train_data mit NumDim $= n + m = w$ aufrufen. Anschließend sollte set_feature_lengths_class_train_data aufgerufen werden mit den Werten *[n,m]* für SubFeatureLength und *[Name1, Name2]* in Names. Anschließend können den Trainingsdaten zugeordnete Daten aus beiden Datenquellen (Länge des Merkmalsvektors: w, zuerst das erste Feature der Länge n dann das zweite Feature der Länge m) mit Hilfe von add_sample_class_train_data hinzugefügt werden. Dann können Operatoren wie select_feature_set_knn oder select_feature_set_svm aufgerufen werden. Das Ergebnis dieser Operatoren wäre dann entweder *[Name1]*, falls die erste Datenquelle alleine ausreicht *[Name2]*, falls die zweite Datenquelle alleine ausreicht, oder *[Name1, Name2]* wenn beide Datenquellen benötigt werden.

Parameter

- ▷ **ClassTrainDataHandle** (input_control) ..... class_train_data ⇝ *handle*
  Trainingsdaten.
- ▷ **SubFeatureLength** (input_control) ..... number-array ⇝ *integer*
  Längen den Merkmalsgruppen.
- ▷ **Names** (input_control) ..... string-array ⇝ *string*
  Namen der Merkmalsgruppen.

Beispiel

```
* Find out which of the two features distinguishes two Classes
NameFeature1 := 'Good Feature'
NameFeature2 := 'Bad Feature'
LengthFeature1 := 3
LengthFeature2 := 2
* Create training data
create_class_train_data (LengthFeature1+LengthFeature2,\
  ClassTrainDataHandle)
* Define the features which are in the training data
set_feature_lengths_class_train_data (ClassTrainDataHandle, [LengthFeature1,\
  LengthFeature2], [NameFeature1, NameFeature2])
* Add training data
*                                                      |Feat1| |Feat2|
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  2,1  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  2,1  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  3,4  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  3,4  ], 1)
* Add more data
* ...
* Select the better feature
select_feature_set_knn (ClassTrainDataHandle, 'greedy', [], [], KNNHandle,\
  SelectedFeature, Score)
classify_class_knn (KNNHandle, [1,1,1], Result, Rating)
classify_class_knn (KNNHandle, [2,2,2], Result, Rating)
* Use the classifier
* ...
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_feature_lengths_class_train_data den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassTrainDataHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_train_data`, `add_sample_class_train_data`

Nachfolger

`select_feature_set_knn`, `select_feature_set_svm`, `select_feature_set_mlp`, `select_feature_set_gmm`

Modul

Foundation

write_class_train_data (: : ClassTrainDataHandle, FileName :)

Schreibt Trainingsdaten für Klassifikatoren in eine Datei.

`write_class_train_data` schreibt die Trainingsdaten für Klassifikatoren `ClassTrainDataHandle` in die Datei `FileName`. Die Trainingsdaten können mit `read_class_train_data` gelesen werden. Die Dateiendung für Trainingsdaten in HALCON ist 'ctd'.

Parameter

▷ **ClassTrainDataHandle** (input_control) ..... class_train_data ⇝ *handle*
Handle der Trainingsdaten.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei, in die die Trainingsdaten geschrieben werden.
Dateiendung: `.ctd`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_class_train_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`create_class_train_data`, `read_class_train_data`

Modul

Foundation

15.6 Support-Vektor-Maschinen

add_class_train_data_svm (: : SVMHandle, ClassTrainDataHandle :)

Hinzufügen von Trainingsdaten zu einer Support-Vektor-Maschine (SVM).

`add_class_train_data_svm` fügt die Trainingsdaten `ClassTrainDataHandle` zu einer Support-Vektor-Maschine (SVM), gegeben durch `SVMHandle`, hinzu.

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle einer SVM.

▷ **ClassTrainDataHandle** (input_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten für den Klassifikator.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_class_train_data_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SVMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_svm`, `create_class_train_data`

Nachfolger

`get_sample_class_svm`

Alternativen

`add_sample_class_svm`

Siehe auch

`create_class_svm`

Modul

Foundation

add_sample_class_svm (: : SVMHandle, Features, Class :)

Hinzufügen eines Trainingsmusters zu den Trainingsdaten einer Support-Vektor-Maschine.

`add_sample_class_svm` fügt ein Trainingsmuster zu der durch `SVMHandle` gegebenen Support-Vektor-Maschine (SVM) hinzu. Das Trainingsmuster wird durch `Features` und `Class` beschrieben. `Features` ist der Merkmalsvektor des Musters und muss folglich die Länge *'NumFeatures'* besitzen, die in `create_class_svm` angegeben worden ist. `Class` kodiert die Klassenzugehörigkeit und muss Werte von 0 bis `NumClasses-1` enthalten (siehe `create_class_svm`). Im Spezialfall von 'novelty detection' muss die Klasse mit 0 übergeben werden, da nur eine Klasse angenommen wird. Bevor das SVM mit `train_class_svm` trainiert werden kann, müssen Trainingsmuster mit `add_sample_class_svm` zum SVM hinzugefügt werden. Das Verwenden von Support-Vektoren einer bereits trainierten SVM als Trainingsdaten wird in `train_class_svm` beschrieben.

Die Anzahl der abgespeicherten Trainingsmuster kann mit `get_sample_num_class_svm` abgefragt werden. Einzelne Trainingsmuster können mit `get_sample_class_svm` wieder ausgelesen werden.

Im Normalfall ist es günstig, die Trainingsmuster mit `write_samples_class_svm` in eine Datei zu speichern, damit sie wiederverwendet werden können, und damit bei Bedarf neue Trainingsmuster zum Datensatz hinzugefügt werden können und eine *neu erzeugte* SVM mit dem erweiterten Datensatz trainiert werden kann.

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle der SVM.

▷ **Features** (input_control) . . . . . real-array ⇝ *real*
Merkmalsvektor des zu speichernden Trainingsmusters.

▷ **Class** (input_control) . . . . . number ⇝ *integer / real*
Klasse des zu speichernden Trainingsmusters.

Ergebnis

Sind die Parameterwerte korrekt, liefert add_sample_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SVMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_class_svm

Nachfolger

train_class_svm, write_samples_class_svm, get_sample_num_class_svm, get_sample_class_svm

Alternativen

read_samples_class_svm

Siehe auch

clear_samples_class_svm, get_support_vector_class_svm

Modul

Foundation

classify_class_svm (: : SVMHandle, Features, Num : Class)

Berechnen der Klasse eines Merkmalsvektors durch eine Support-Vektor-Maschine.

classify_class_svm berechnet mit dem durch SVMHandle gegebene Support-Vektor-Maschine (SVM) die Num besten Klassen des Merkmalsvektors Features und gibt die Klassen in Class zurück. Falls der Klassifikator im Mode = *'one-versus-one'* erzeugt wurde, werden die Klassen nach der Anzahl der Stimmen der Sub-Klassifikatoren sortiert. Im Mode = *'one-versus-all'* erfolgt die Sortierung nach dem Ausgabewert der Unter-Klassifikatoren (siehe create_class_svm). Falls der Klassifikator im Mode = *'novelty-detection'* erzeugt wurde, wird nur darüber entschieden, ob der Merkmalsvektor zu der Trainingsklasse gehört oder als Ausreißer angesehen wird. In diesem Fall bedeutet ein Ausgabewert von Class = 0, dass die Klasse als von der Trainingsmenge abweichend betrachtet wird. Ein Wert von Class = 1 bedeutet, dass der Merkmalsvektor aus der gleichen Verteilung wie die Trainingsdaten stammt. In diesem Fall wird Num auf 1 gesetzt, da der Klassifikator nur die Zugehörigkeit zu einer Klasse testet.

Die SVM muss vor der Verwendung von classify_class_svm mit train_class_svm trainiert werden.

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle der SVM.

▷ **Features** (input_control) . . . . . real-array ⇝ *real*
Merkmalsvektor.

▷ **Num** (input_control) . . . . . integer-array ⇝ *integer*
Anzahl der zu bestimmenden besten Klassen.
Defaultwert: 1
Wertevorschläge: Num ∈ {1, 2, 3, 4, 5}

▷ **Class** (output_control) ..... integer(-array) ⇝ *integer*
Ergebnis der Klassifikation des Merkmalsvektors durch die SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `classify_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_svm`, `read_class_svm`

Alternativen

`apply_dl_classifier`

Siehe auch

`create_class_svm`

Literatur

John Shawe-Taylor, Nello Cristianini: „Kernel Methods for Pattern Analysis“; Cambridge University Press, Cambridge; 2004.

Bernhard Schölkopf, Alexander J.Smola: „Learning with Kernels“; MIT Press, London; 1999.

Modul

Foundation

clear_class_svm (: : SVMHandle :)

Löschen einer Support-Vektor-Maschine.

`clear_class_svm` löscht die Support-Vektor-Maschine (SVM), die durch `SVMHandle` angegeben wird, und gibt sämtlichen für die SVM benötigten Speicher frei. Die SVM kann nach dem Aufruf von `clear_class_svm` nicht mehr verwendet werden. Der Handle `SVMHandle` ist nach dem Aufruf ungültig.

Parameter

▷ **SVMHandle** (input_control) ..... class_svm(-array) ⇝ *handle*
Handle der SVM.

Ergebnis

Falls `SVMHandle` korrekt ist, dann liefert `clear_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SVMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`classify_class_svm`

Siehe auch

`create_class_svm`, `read_class_svm`, `write_class_svm`, `train_class_svm`

Modul

Foundation

clear_samples_class_svm (: : SVMHandle :)

Löschen aller Trainingsdaten einer Support-Vektor-Maschine.

clear_samples_class_svm löscht alle mit add_sample_class_svm oder read_samples_class_svm zur Support-Vektor-Maschine (SVM) SVMHandle hinzugefügten Trainingsmuster. Eine Verwendung von clear_samples_class_svm ist nur dann sinnvoll, wenn die SVM in demselben Prozess trainiert wird, in dem die SVM auch zur Klassifikation mit classify_class_svm verwendet wird. In diesem Fall kann der durch die Trainingsmuster belegte Speicher mit clear_samples_class_svm wieder freigegeben werden und somit Speicherplatz gespart werden. In der üblichen Verwendungsart, in der die SVM offline trainiert wird und mit write_class_svm gespeichert wird, ist die Verwendung von clear_samples_class_svm normalerweise überflüssig, da write_class_svm die Trainingsmuster nicht abspeichert, und somit im Online-Prozess, der die SVM mit read_class_svm einliest, auch kein Speicher für die Trainingsmuster benötigt wird.

Parameter

▷ **SVMHandle** (input_control) ..... class_svm(-array) ⇝ *handle*
Handle der SVM.

Ergebnis

Sind die Parameterwerte korrekt, liefert clear_samples_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SVMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

train_class_svm, write_samples_class_svm

Siehe auch

create_class_svm, clear_class_svm, add_sample_class_svm, read_samples_class_svm

Modul

Foundation

create_class_svm (: : NumFeatures, KernelType, KernelParam, Nu, NumClasses, Mode, Preprocessing, NumComponents : SVMHandle)

Erzeugen einer Support-Vektor-Maschine zur Klassifikation von Mustern.

create_class_svm erzeugt eine Support-Vektor-Maschine, welche zur Klassifikation von Mustern verwendet werden kann. Die Dimension, aus denen ein Muster besteht, wird mit NumFeatures, die Anzahl der verschiedenen Klassen mit NumClasses angegeben.

Für ein binäres Klassifikationsproblem, in dem die Klassen linear separierbar sind, werden aus einem gegebenen Trainingsdatensatz die Merkmalsvektoren ausgewählt, die die optimal trennende Hyperebene zwischen den verschieden Klassen aufspannen. Optimal bedeutet hier, dass der Abstand zwischen den konvexen Hüllen der

jeweiligen Datensätze maximal wird. Die Merkmalsvektoren, die am Rand liegen und die Hyperebene implizit aufspannen, werden *Support-Vektoren* (SV) genannt.

Die Klassifikation eines neuen Merkmalsvektors z erfolgt mit der Formel:

$$f(z) \quad = \quad sign\left(\sum_{i=1}^{n_{sv}} \alpha_i y_i < x_i, z > +b\right)$$

Dabei sind x_i die Support-Vektoren, y_i kodiert ihre jeweilige Klassenzugehörigkeit (± 1) und α_i sind die Gewichtskoeffizienten. Mit b wird der Abstand der Hyperebene zum Ursprung bezeichnet. Hierbei ist anzumerken, dass für die Klassifikation nur ein Bruchteil des originalen Datensatzes benötigt wird (n_{sv}: Anzahl der Support-Vektoren), der genau an der Hyperebene liegt. Deshalb werden die Datenvektoren, die keine Support-Vektoren sind, verworfen. Die Klassifikationsgeschwindigkeit hängt von der Auswertung des Skalarproduktes zwischen den Support-Vektoren und dem zu klassifizierenden Merkmalsvektor ab, und somit von der Länge des Merkmalsvektors und der Anzahl n_{sv} der Support-Vektoren.

Für Klassifikationsprobleme, bei denen die Klassen nicht linear separierbar sind, wird der Algorithmus in zweierlei Art erweitert. Erstens wird ein bestimmter Anteil von Fehlern (Überlappungen) durch die Verwendung von Schlupfvariablen innerhalb des Trainings ausgeglichen. Das bedeutet, dass die Gewichte (α) einen maximalen Wert nicht übersteigen. Um intuitiv die Anzahl von Überlappungen einzustellen, wird die als *Nu*-SVM bezeichnete Variante des Trainingsalgorithmus verwendet. Der Regularisierungsparameter `Nu` stellt eine obere Schranke für Trainingsfehler und eine untere Schranke für den Anteil von SV dar. Als Faustregel empfiehlt es sich, einen Wert für `Nu` einzustellen, der den Anteil der anwendungsspezifisch erwarteten Fehler entspricht, z.B. *0.01* (entsprechend 1% maximalem Trainingsfehler). Es ist zu beachten, dass ein zu großer Wert für `Nu` zu einem unzulässigen Trainingsproblem führen kann, und somit die SVM nicht trainiert werden kann (siehe `train_class_svm` für Details). Da dies aber erst während des Trainings festgestellt werden kann, erfolgt eine Fehlerbehandlung erst dort. In diesem Fall ist eine neue SVM mit angepasstem `Nu` zu erzeugen.

Zweitens ist es möglich, eine so genannte Kernelfunktion in dem Trainings- bzw. Klassifikationsalgorithmus zu verwenden, da zwischen Merkmalsvektoren ausschließlich Skalarprodukte berechnet werden. Das bedeutet, dass das Skalarprodukt durch eine Kernelfunktion ersetzt wird, die das Skalarprodukt implizit in einem höherdimensionalen Merkmalsraum ausführt. Ein im Ursprungsraum linear nicht separierbarer Datensatz wird mit einer passenden Kernelfunktion im höherdimensionalem Merkmalsraum linear separierbar.

Die entsprechende Kernelfunktion kann mit dem Parameter `KernelType` ausgewählt werden: Für `KernelType` = *'linear'* wird das ursprüngliche Skalarprodukt (wie in der Formel) verwendet. Dieser Kernel sollte nur für linear separierbare Datensätze verwendet werden. Der Parameter `KernelParam` wird ignoriert.

Die Radiale Basis Funktion (`KernelType` = *'rbf'*) stellt die beste Wahl für die Kernelfunktion dar, da mit ihr relativ leicht gute Ergebnisse erzielt werden können. Sie ist folgendermaßen definiert:

$$K(x,z) \quad = \quad e^{-\gamma \cdot \left\|x-z\right\|^2}$$

Dabei muss im Parameter `KernelParam` der Wert von γ eingestellt werden. Anschaulich bedeutet der Wert, wie groß der Einfluss eines SV auf seine Umgebung ist. Ein zu großer Wert (kleiner Einfluss auf die Umgebung) bedeutet, dass jeder Trainingsvektor ein SV wird. Der Trainingsalgorithmus lernt den Trainingsdatensatz "auswendig„, ist aber nicht in der Lage zu generalisieren (Überanpassung). Zusätzlich dauern Training und Klassifikation viel länger. Ein zu kleiner Wert von γ führt dazu, dass wenige SV einen großen Einfluss auf weit entfernte Bereiche haben (Unteranpassung). Ein typisches Vorgehen bei der Ermittlung des optimalen γ ist es, von einem kleinen γ-`Nu` Paar auszugehen und die Werte solange zu erhöhen, solange die Erkennungsraten sich verbessern.

Mit `KernelType` = *'polynomial_homogeneous'* oder *'polynomial_inhomogeneous'* können Polynome als Kernel ausgewählt werden. Sie sind folgendermaßen definiert:

$$\begin{aligned} K(x,z) &= (< x,z >)^d \\ K(x,z) &= (< x,z > +1)^d \end{aligned}$$

Der Grad des Polynoms d muss in `KernelParam` eingestellt werden. Dabei ist zu beachten, dass Polynome zu hohen Grades ($d > 10$) zu numerischen Problemen führen können.

Eine Faustregel ist, dass der RBF-Kernel für die meisten Klassifikationsprobleme eine gute Wahl darstellt und deshalb ausschließlich verwendet werden soll. Nichtsdestotrotz erzeugen der lineare und polynomielle Kernel in manchen Anwendungen gute Klassifikationsergebnisse und können zum Vergleich ausprobiert werden. Weiterhin ist zu beachten, dass nur für den RBF-Kernel der `Mode` *'novelty-detection'* und der Operator `reduce_class_svm` unterstützt werden.

Mit `Mode` wird der generelle Klassifikationsmodus angegeben, was entweder beschreibt, wie das Multi-Klassen Problem auf binäre Entscheidungen reduziert wird oder ob eine so genannte *'novelty-detection'* Aufgabe gelöst werden soll. `Mode` = *'one-versus-all'* erzeugt einen Klassifikator für jede Klasse, die sie mit allen restlichen Klassen vergleicht. In der Testphase wird die Klasse mit dem größten Klassifikationsergebnis (siehe Klassifikationsformel ohne Vorzeichen) ausgewählt. `Mode` = *'one-versus-one'* erzeugt zwischen allen Paaren von Klassen jeweils einen Klassifikator. In der Testphase erfolgt eine Abstimmung, bei der die Klasse mit den meisten Stimmen ausgewählt wird. Die optimale Strategie (`Mode`) hängt von der Anzahl der Klassen bzw. der Anwendung ab. Bei n Klassen erzeugt *'one-versus-all'* n Klassifikatoren und *'one-versus-one'* $n(n-1)/2$. Bei nur zwei Klassen erzeugt *'one-versus-one'* genau einen Klassifikator und *'one-versus-all'* unnötigerweise zwei symmetrische Klassifikatoren. Bei wenigen Klassen (ca. bis zu 10) ist *'one-versus-one'* sowohl im Training als auch in der Testphase schneller, da die Unterklassifikatoren weniger Trainingsdaten enthalten und in der Summe weniger Support-Vektoren entstehen. Für viele Klassen ist *'one-versus-all'* vorzuziehen, da die Anzahl der Klassifikatoren für *'one-versus-one'* quadratisch wächst und daher unnötig viele Sub-Klassifikatoren erzeugt werden.

Einen besonderen Fall stellt die Situation dar, falls nur die Zugehörigkeit von Testdaten zu den Trainingsdaten bestimmt werden soll (`Mode` = *'novelty-detection'*), d.h. `NumClasses` muss auf 1 gesetzt sein. Die separierende Hyperebene verläuft nun um die Trainingsdaten herum und trennt hierbei implizit Trainingsdaten von der Umgebung. Der Vorteil ist hierbei, dass eine explizite Angabe einer Rückweisungsklasse entfällt, was in bestimmten Anwendungen wie z.B. Texturklassifikation vorteilhaft ist. Die Support-Vektoren sind hierbei Daten, die am äußeren Rand der Verteilung liegen. Der Parameter `Nu` bestimmt den Anteil an Ausreißern innerhalb des Trainingsdatensatzes. Es ist zu beachten, dass bei der Klassifikation im *'novelty-detection'* Modus die Klasse der Trainingsdaten mit dem Index 1 und die Rückweisungsklasse mit dem Index 0 zurückgegeben wird. Somit dient die erste Klasse als Rückweisungsklasse. Im Gegensatz dazu ist bei der Klassifikation mit MLP standardmäßig die letzte Klasse als Rückweisungsklasse.

Das ist anders als bei den MLP, bei denen im Standardfall die letzte Klasse die Rückweisungsklasse wird.

Mit den Parametern `Preprocessing` und `NumComponents` kann eine Vorverarbeitung der Merkmalsvektoren festgelegt werden. Für `Preprocessing` = *'none'* werden die Merkmalsvektoren ohne Änderung an die SVM übergeben. `NumComponents` wird hier ignoriert.

Für alle anderen Werte von `Preprocessing` wird aus dem Trainingsdatensatz eine Transformation der Merkmale berechnet, die sowohl beim Training als auch bei der späteren Klassifikation dazu verwendet wird, die Merkmalsvektoren zu transformieren.

Für `Preprocessing` = *'normalization'* werden die Merkmalsvektoren normalisiert. Im Falle eines polynomiellen Kernels wird der minimale bzw. maximale Wert des Trainingsdatensatzes auf -1 bzw. +1 abgebildet. Im Falle eines RBF-Kernels wird normalisiert, indem der Mittelwert aller Trainingsvektoren von den Merkmalsvektoren abgezogen wird und die dadurch entstehenden Merkmalsvektoren durch die Standardabweichung der jeweiligen Komponente der Trainingsvektoren geteilt werden. Die transformierten Merkmalsvektoren haben also einen Mittelwert von 0 und eine Standardabweichung von 1 in jeder Komponente. `NumComponents` wird in beiden Fällen ignoriert. Diese Transformation empfiehlt sich bei allen Daten, bei denen die Komponenten der Merkmalsvektoren nicht in denselben Einheiten gemessen werden (z.B. falls einige der Daten Grauwertmerkmale und andere Regionenmerkmale sind, oder falls z.B. Regionenmerkmale wie 'circularity' (Einheit: Skalar) und 'area' (Einheit: Quadratpixel) gemischt werden). Die Normalisierung sollte grundsätzlich durchgeführt werden, um die numerische Stabilität während des Trainings/Testens zu verbessern.

Für `Preprocessing` = *'principal_components'* werden die Merkmalsvektoren einer Hauptachsentransformation (principal component analysis, PCA) unterzogen. Die Hauptachsentransformation normalisiert zunächst die Merkmalsvektoren (s.o.). Danach wird eine orthogonale Transformation (eine Rotation im Merkmalsraum) berechnet, die die Trainingsvektoren dekorreliert. Nach der Transformation ist der Mittelwert der Trainingsvektoren 0 und die Kovarianzmatrix der Trainingsvektoren ist eine Diagonalmatrix. Die Transformation wird so bestimmt, dass die transformierten Merkmale, die die größte Variationsbreite aufweisen, als erstes in dem transformierten Merkmalsvektor stehen. Dadurch kann erreicht werden, dass die letzten Komponenten des Merkmalsvektors, die typischerweise stark vom Rauschen beeinflusst werden, ohne großen Informationsverlust weggelassen werden können. Mit `NumComponents` wird festgelegt, wie viele der transformierten Komponenten verwendet werden sollen. Es können bis zu `NumFeatures` Komponenten selektiert werden. Mit Hilfe von

get_prep_info_class_svm kann der Informationsgehalt der einzelnen transformierten Komponenten bestimmt werden und somit NumComponents einfacher bestimmt werden. Wie die Normalisierung empfiehlt sich diese Transformation, wenn die Merkmalsvektoren Mittelwerte und Standardabweichungen haben, die sich stark von 0 bzw. 1 unterscheiden oder bei denen die Komponenten der Merkmalsvektoren nicht in denselben Einheiten gemessen werden, und wenn zusätzlich zu erwarten ist, dass die Merkmale stark korreliert sind. Hier ist anzumerken, dass der RBF-Kernel bezüglich Dimensionsreduktion durch die PCA sehr robust ist. Daher sollte er die erste Wahl sein, wenn es darum geht, die Klassifikationszeiten zu beschleunigen.

Mit Preprocessing = *'canonical_variates'* (kanonische Merkmale) wird eine Transformation bestimmt, die die Trainingsvektoren zuerst normalisiert und dann im Mittel über alle Klassen dekorreliert. Gleichzeitig werden in den transformierten Trainingsvektoren die Mittelwerte der einzelnen Klassen möglichst weit separiert. Wie bei Preprocessing = *'principal_components'* werden die transformierten Komponenten nach Informationsgehalt sortiert, so dass transformierte Merkmale mit wenig Informationsgehalt weggelassen werden können. Bei kanonischen Merkmalen können höchstens $\min(\text{NumClasses}-1, \text{NumFeatures})$ Merkmale selektiert werden. Auch hier kann mit get_prep_info_class_svm der Informationsgehalt der einzelnen transformierten Komponenten bestimmt werden. Wie die Hauptachsentransformation können die kanonischen Merkmale dazu verwendet werden, die Datenmenge ohne großen Informationsverlust zu verringern, wobei zusätzlich noch die Trennbarkeit der Daten nach der Datenreduktion optimiert wird.

Für die letzten zwei Transformationsarten (*'principal_components'* und *'canonical_variates'*) bestimmt NumComponents die Länge des transformierten Datenvektors, während NumFeatures die Dimensionalität der Eingabedaten (Länge des untransformierten Merkmalsvektors) bestimmt. Dadurch wird die Länge der Eingabevariablen der SVM geringer, wodurch sich normalerweise die Trainingszeit und die Klassifikationszeit verringert.

Nachdem die SVM mit create_class_svm erzeugt wurde, werden typischerweise mit add_sample_class_svm oder read_samples_class_svm Trainingsdaten hinzugefügt und die SVM mit train_class_svm trainiert. Daraufhin kann die SVM mit write_class_svm abgespeichert werden. Alternativ können auch sofort nach dem Training Daten mit classify_class_svm klassifiziert werden.

Ein Vergleich zwischen SVM und mehrschichtigem Perzeptron (engl.: multilayer perceptron, MLP; siehe create_class_mlp) zeigt typischerweise, dass SVMs generell schneller trainiert werden, insbesondere bei großen Trainingsdatensätzen, und eine leicht verbesserte Erkennungsrate haben. Das MLP weist schnellere Klassifikationszeiten auf und sollte daher in zeitkritischen Anwendungen verwendet werden. Es ist zu beachten, dass der Vergleich von optimal abgestimmten Parametern ausgeht.

Parameter

▷ **NumFeatures** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Eingabevariablen (Merkmale) der SVM.
Defaultwert: 10
Wertevorschläge: NumFeatures ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Restriktion: NumFeatures >= 1

▷ **KernelType** (input_control) . . . . . string ⇝ *string*
Art des Kernels.
Defaultwert: 'rbf'
Werteliste: KernelType ∈ {'linear', 'rbf', 'polynomial_inhomogeneous', 'polynomial_homogeneous'}

▷ **KernelParam** (input_control) . . . . . real ⇝ *real*
Zusätzlicher Parameter für die Kernel Funktion. Im Falle des RBF-Kernels der Wert für γ. Falls polynomieller Kernel ausgewählt wurde, der Grad des Polynoms.
Defaultwert: 0.02
Wertevorschläge: KernelParam ∈ {0.01, 0.02, 0.05, 0.1, 0.5}

▷ **Nu** (input_control) . . . . . real ⇝ *real*
Regularisierungskonstante der SVM.
Defaultwert: 0.05
Wertevorschläge: Nu ∈ {0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3}
Restriktion: Nu > 0.0 && Nu < 1.0

▷ **NumClasses** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Klassen.
Defaultwert: 5
Wertevorschläge: NumClasses ∈ {2, 3, 4, 5, 6, 7, 8, 9, 10}
Restriktion: NumClasses >= 1

▷ **Mode** (input_control) ..... string ⇝ *string*
Modus der SVM.
Defaultwert: 'one-versus-one'
Werteliste: Mode ∈ {'novelty-detection', 'one-versus-all', 'one-versus-one'}

▷ **Preprocessing** (input_control) ..... string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
Defaultwert: 'normalization'
Werteliste: Preprocessing ∈ {'none', 'normalization', 'principal_components', 'canonical_variates'}

▷ **NumComponents** (input_control) ..... integer ⇝ *integer*
Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale (ignoriert bei Preprocessing = *'none'* und Preprocessing = *'normalization'*).
Defaultwert: 10
Wertevorschläge: NumComponents ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Restriktion: NumComponents >= 1

▷ **SVMHandle** (output_control) ..... class_svm ⇝ *handle*
Handle der SVM.

Beispiel

```
create_class_svm (NumFeatures, 'rbf', 0.01, 0.01, NumClasses,\
                  'one-versus-all', 'normalization', NumFeatures,\
                  SVMHandle)
* Generate and add the training data
for J := 0 to NumData-1 by 1
    * Generate training features and classes
    * Data = [...]
    * Class = ...
    add_sample_class_svm (SVMHandle, Data, Class)
endfor
* Train the SVM
train_class_svm (SVMHandle, 0.001, 'default')
* Use the SVM to classify unknown data
for J := 0 to N-1 by 1
    * Extract features
    * Features = [...]
    classify_class_svm (SVMHandle, Features, 1, Class)
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

add_sample_class_svm

Alternativen

read_dl_classifier, create_class_mlp, create_class_gmm

Siehe auch

clear_class_svm, train_class_svm, classify_class_svm

Literatur

Bernhard Schölkopf, Alexander J.Smola: „Learning with Kernels“; MIT Press, London; 1999.

John Shawe-Taylor, Nello Cristianini: „Kernel Methods for Pattern Analysis“; Cambridge University Press, Cambridge; 2004.

Modul

Foundation

deserialize_class_svm (: : SerializedItemHandle : SVMHandle)

Deserialisieren einer serialisierten Support-Vektor-Maschine (SVM).

deserialize_class_svm deserialisiert eine Support-Vektor-Maschine (SVM) (einschließlich eventuell abgespeicherter Trainingsmuster), welche mit serialize_class_svm serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte SVM wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einer dafür automatisch erzeugten SVM mit dem Handle SVMHandle gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **SVMHandle** (output_control) ..... class_svm ⇝ *handle*
  Handle der SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_class_svm

Nachfolger

classify_class_svm, create_class_lut_svm

Siehe auch

create_class_svm, write_class_svm, serialize_class_svm

Modul

Foundation

evaluate_class_svm (: : SVMHandle, Features : Result)

Berechnen der Konfidenz eines Merkmalsvektors durch eine Support-Vektor-Maschine.

evaluate_class_svm berechnet für einen Merkmalsvektor, dass in Features übergeben wird, das Ergebnis Result für eine SVM SVMHandle. Der Operator evaluate_class_svm kann nur benutzt werden, wenn es im Modus Mode = *'novelty-detection'* erzeugt wurde. Wenn der Merkmalsvektor innerhalb der Klasse liegt, liefert Result einen Wert größer 1.0. Wenn der Merkmalsvektor außerhalb der Klassengrenze liegt, z.B. weil es ein Ausreißer ist, wird ein Wert kleiner 1.0 zurückgegeben.

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle der SVM.

▷ **Features** (input_control) . . . . . real-array ⇝ *real*
Merkmalsvektor.

▷ **Result** (output_control) . . . . . real(-array) ⇝ *real*
Ergebnis der Evaluation des Merkmalsvektors durch die SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `evaluate_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_svm`, `read_class_svm`

Siehe auch

`create_class_svm`

Modul

Foundation

get_class_train_data_svm (: : SVMHandle : ClassTrainDataHandle)

Liest die Trainingsdaten einer SVM aus.

`get_class_train_data_svm` liest die Trainingsdaten einer SVM aus und liefert diese als Handle in `ClassTrainDataHandle` zurück.

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle einer SVM.

▷ **ClassTrainDataHandle** (output_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten der SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_class_train_data_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`add_sample_class_svm`, `read_samples_class_svm`

Nachfolger

`add_class_train_data_mlp`, `add_class_train_data_gmm`, `add_class_train_data_knn`

Siehe auch

`create_class_train_data`

Modul

Foundation

get_params_class_svm (: : SVMHandle : NumFeatures, KernelType, KernelParam, Nu, NumClasses, Mode, Preprocessing, NumComponents)

Auslesen der Parameter einer Support-Vektor-Maschine.

get_params_class_svm gibt die Parameter der durch SVMHandle angegebenen Support-Vektor-Maschine (SVM), die bei der Erzeugung mit create_class_svm verwendet wurden, zurück. Dies ist insbesondere nützlich, wenn die SVM mit read_class_svm von Datei eingelesen wurde. Die Ausgabedaten von get_params_class_svm können z.B. verwendet werden, um zu überprüfen, ob die Merkmalsvektoren und ggf. die Zieldaten, die verwendet werden sollen, zu der SVM passen. Zur Beschreibung der Parameter siehe create_class_svm.

Parameter

- ▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
  Handle der SVM.
- ▷ **NumFeatures** (output_control) . . . . . integer ⇝ *integer*
  Anzahl der Eingabevariablen (Merkmale) der SVM.
- ▷ **KernelType** (output_control) . . . . . string ⇝ *string*
  Art des Kernels.
- ▷ **KernelParam** (output_control) . . . . . real ⇝ *real*
  Zusätzlicher Parameter für den Kernel.
- ▷ **Nu** (output_control) . . . . . real ⇝ *real*
  Regularisierungskonstante der SVM.
- ▷ **NumClasses** (output_control) . . . . . integer ⇝ *integer*
  Anzahl der Klassen aus denen die Testdaten bestehen.
- ▷ **Mode** (output_control) . . . . . string ⇝ *string*
  Modus der SVM.
- ▷ **Preprocessing** (output_control) . . . . . string ⇝ *string*
  Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
- ▷ **NumComponents** (output_control) . . . . . integer ⇝ *integer*
  Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale (ignoriert bei Preprocessing = *'none'* und Preprocessing = *'normalization'*).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_params_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_class_svm, read_class_svm

Nachfolger

add_sample_class_svm, train_class_svm

Siehe auch

classify_class_svm

Modul

Foundation

get_prep_info_class_svm (: : SVMHandle,
Preprocessing : InformationCont, CumInformationCont)

Berechnen des Informationsgehaltes der vorverarbeiteten Merkmalsvektoren einer Support-Vektor-Maschine.

get_prep_info_class_svm berechnet den Informationsgehalt der mit der durch Preprocessing gegebenen Vorverarbeitung transformierten Komponenten der Trainingsvektoren. Preprocessing kann auf *'principal_components'* oder *'canonical_variates'* gesetzt werden. Die zu Grunde liegenden Vorverarbeitungen sind bei create_class_svm beschrieben. Der Informationsgehalt wird aus die Variation der transformierten Komponenten der Trainingsvektoren berechnet, d.h. er wird rein aufgrund der Trainingsdaten unabhängig von einer Fehlerrate bei einer Klassifikation der Trainingsdaten berechnet. Der Informationsgehalt wird für alle relevanten Komponenten der transformierten Merkmalsvektoren (NumFeatures für *'principal_components'* und $\min(\texttt{NumClasses} - 1, \texttt{NumFeatures})$ für *'canonical_variates'*, siehe create_class_svm) in InformationCont zurückgegeben. Der Informationsgehalt wird als eine Zahl zwischen 0 und 1 dargestellt. Ein prozentualer Informationsgehalt kann leicht durch Multiplikation mit 100 berechnet werden. Der kumulierte Informationsgehalt der ersten n Komponenten wird in der n-ten Komponente von CumInformationCont zurückgegeben, d.h. CumInformationCont enthält die Summen der ersten n Elemente von InformationCont. Um get_prep_info_class_svm verwenden zu können, müssen mit add_sample_class_svm oder read_samples_class_svm genügend viele Trainingsmuster zu der durch SVMHandle gegebenen SVM hinzugefügt werden.

InformationCont und CumInformationCont können dazu verwendet werden, um zu entscheiden, wie viele Komponenten der transformierten Merkmalsvektoren relevante Information enthalten. Ein oft verwendetes Kriterium ist z.B. zu verlangen, dass die transformierten Daten x% (z.B. 90%) der Daten enthalten sollten. Dies kann leicht anhand des ersten Wertes von CumInformationCont, der über x% liegt, bestimmt werden. Der so erhaltene Wert kann bei einem neuerlichen Aufruf von create_class_svm als NumComponents verwendet werden. Da zum Aufruf von get_prep_info_class_svm schon eine SVM mit create_class_svm erzeugt werden muss, also auch ein initialer Wert von NumComponents bei create_class_svm angegeben werden muss, aber bei Verwendung von get_prep_info_class_svm typischerweise noch nicht bekannt ist, wie viele Komponenten relevant sind, empfiehlt sich folgendes zweistufiges Vorgehen, um NumComponents zu bestimmen: In einem ersten Schritt wird eine SVM mit der maximalen Anzahl von NumComponents (NumFeatures für *'principal_components'* und $\min(\texttt{NumClasses} - 1, \texttt{NumFeatures})$ für *'canonical_variates'*) erzeugt. Dann werden die Trainingsmuster zu der SVM hinzugefügt und mit write_samples_class_svm in eine Datei gespeichert. Anschließend wird mit get_prep_info_class_svm der Informationsgehalt der Komponenten und somit NumComponents bestimmt. Danach wird eine neue SVM mit der gewünschten Anzahl Komponenten erzeugt, und die abgespeicherten Trainingsdaten mit read_samples_class_svm wieder eingelesen. Hierauf wird die SVM mit train_class_svm trainiert.

Parameter

▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
Handle der SVM.

▷ **Preprocessing** (input_control) ..... string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
Defaultwert: 'principal_components'
Werteliste: Preprocessing ∈ {'principal_components', 'canonical_variates'}

▷ **InformationCont** (output_control) ..... real-array ⇝ *real*
Relativer Informationsgehalt der transformierten Merkmalsvektoren.

▷ **CumInformationCont** (output_control) ..... real-array ⇝ *real*
Kumulierter Informationsgehalt der transformierten Merkmalsvektoren.

Beispiel

```
* Create the initial SVM
create_class_svm (NumFeatures, 'rbf', 0.01, 0.01, NumClasses,\
                  'one-versus-all', 'normalization', NumFeatures,\
                  SVMHandle)
* Generate and add the training data
for J := 0 to NumData-1 by 1
    * Generate training features and classes
```

```
    * Data = [...]
    * Class = ...
    add_sample_class_svm (SVMHandle, Data, Class)
endfor
write_samples_class_svm (SVMHandle, 'samples.mtf')
* Compute the information content of the transformed features
get_prep_info_class_svm (SVMHandle, 'principal_components',\
                         InformationCont, CumInformationCont)
* Determine NumComp by inspecting InformationCont and CumInformationCont
* NumComp = [...]
* Create the actual SVM
create_class_svm (NumFeatures, 'rbf', 0.01, 0.01, NumClasses, \
                  'one-versus-all', 'principal_components', \
                  NumComp, SVMHandle)
* Train the SVM
read_samples_class_svm (SVMHandle, 'samples.mtf')
train_class_svm (SVMHandle, 0.001, 'default')
write_class_svm (SVMHandle, 'classifier.svm')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_prep_info_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls `get_prep_info_class_svm` den Fehler 9211 (Matrix ist nicht positiv definit) bei `Preprocessing` = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_class_svm`, `read_samples_class_svm`

Nachfolger

`clear_class_svm`, `create_class_svm`

Literatur

Christopher M. Bishop: „Neural Networks for Pattern Recognition“; Oxford University Press, Oxford; 1995.

Andrew Webb: „Statistical Pattern Recognition“; Arnold, London; 1999.

Modul

Foundation

get_sample_class_svm (: : SVMHandle, IndexSample : Features, Target)

Auslesen eines Trainingsmusters aus den Trainingsdaten einer Support-Vektor-Maschine.

`get_sample_class_svm` liest ein Trainingsmuster, das mit `add_sample_class_svm` oder `read_samples_class_svm` hinzugefügt wurde, aus der Support-Vektor-Maschine (SVM) `SVMHandle` aus. Der Index des auszulesenden Musters wird mit `IndexSample` festgelegt. Er wird ab 0 gezählt, d.h. `IndexSample` muss zwischen 0 und `IndexSamples` − 1 liegen, wobei `IndexSamples` mit `get_sample_num_class_svm` bestimmt werden kann. Das Trainingsmuster wird in `Features` und `Target` zurückgegeben. Dabei ist `Features` ein Merkmalsvektor der Länge `NumFeatures` (siehe `create_class_svm`) und `Target` der Index der Klasse, welcher ein Wert zwischen 0 und `NumClasses-1` ist (siehe `add_sample_class_svm`).

get_sample_class_svm kann z.B. dazu verwendet werden, die Trainingsdaten mit classify_class_svm zu reklassifizieren, um festzustellen, welche der Trainingsmuster eventuell falsch klassifiziert werden.

Parameter

- ▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
  Handle der SVM.
- ▷ **IndexSample** (input_control) ..... integer ⇝ *integer*
  Nummer des gespeicherten Trainingsmusters.
- ▷ **Features** (output_control) ..... real-array ⇝ *real*
  Merkmalsvektor des Trainingsmusters.
- ▷ **Target** (output_control) ..... integer ⇝ *integer*
  Zielvektor des Trainingsmusters.

Beispiel

```
* Train an SVM
create_class_svm (NumFeatures, 'rbf', 0.01, 0.01, NumClasses,\
                  'one-versus-all', 'normalization', NumFeatures,\
                  SVMHandle)
read_samples_class_svm (SVMHandle, 'samples.mtf')
train_class_svm (SVMHandle, 0.001, 'default')
* Reclassify the training samples
get_sample_num_class_svm (SVMHandle, NumSamples)
for I := 0 to NumSamples-1 by 1
    get_sample_class_svm (SVMHandle, I, Data, Target)
    classify_class_svm (SVMHandle, Data, 1, Class)
    if (Class != Target)
        * Sample has been classified incorrectly
    endif
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sample_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_svm, read_samples_class_svm, get_sample_num_class_svm, get_support_vector_class_svm

Nachfolger

classify_class_svm

Siehe auch

create_class_svm

Modul

Foundation

get_sample_num_class_svm (: : SVMHandle : NumSamples)

Auslesen der Anzahl der Trainingsmuster in den Trainingsdaten einer Support-Vektor-Maschine.

get_sample_num_class_svm liefert in NumSamples die Anzahl der in dem durch SVMHandle gegebenen Support Vektor Machine (SVM) verfügbaren Trainingsmuster zurück. get_sample_num_class_svm sollte aufgerufen werden, bevor mit get_sample_class_svm die einzelnen Trainingsmuster ausgelesen werden, z.B. um eine Reklassifikation der Trainingsdaten durchzuführen (siehe get_sample_class_svm).

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle der SVM.

▷ **NumSamples** (output_control) . . . . . integer ⇝ *integer*
Anzahl der gespeicherten Trainingsmuster.

Ergebnis

Falls SVMHandle korrekt ist, liefert get_sample_num_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

add_sample_class_svm, read_samples_class_svm

Nachfolger

get_sample_class_svm

Siehe auch

create_class_svm

Modul

Foundation

get_support_vector_class_svm (: : SVMHandle,
IndexSupportVector : Index)

Auslesen des Index eines Support-Vektors aus einer trainierten Support-Vektor-Maschine.

Der Operator get_support_vector_class_svm stellt eine Beziehung zwischen einem Support-Vektor einer trainierten SVM (gegeben in SVMHandle) und dem originalen Trainingsdatensatz her. Der Index des abzufragenden SV wird mit IndexSupportVector festgelegt. Er wird ab 0 gezählt, d.h. IndexSupportVector muss zwischen 0 und NumSupportVectors − 1 liegen, wobei NumSupportVectors mit get_support_vector_num_class_svm bestimmt werden kann. Der Index, den dieser Datensatz im ursprünglichen Trainingsdatensatz hatte wird in Index zurückgegeben. Diese Index kann als Parameter für get_sample_class_svm benützt werden, um die Datenvektoren zu bestimmen, die Support Vektoren geworden sind. get_support_vector_class_svm kann z.B. dazu verwendet werden, die Support Vektoren zu visualisieren.

Man beachte, dass der zurückgelieferte Index gleich *-1*, also ungültig, ist, falls train_class_svm mit einem anderen Modus als *'default'* aufgerufen wurde oder falls die SVM mit reduce_class_svm erzeugt wurde. Der Grund hierfür ist, dass ein konsistentes Abbilden von Support-Vektoren auf Trainingsdatensatz unmöglich ist.

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle der SVM.

▷ **IndexSupportVector** (input_control) . . . . . integer ⇝ *integer*
Index des gespeicherten Support-Vektors.

▷ **Index** (output_control) . . . . . real ⇝ *real*
Index des Support-Vektors im Trainingsdatensatz.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sample_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_class_svm, get_support_vector_num_class_svm

Nachfolger

get_sample_class_svm

Siehe auch

create_class_svm

Modul

Foundation

get_support_vector_num_class_svm (: : SVMHandle : NumSupportVectors, NumSVPerSVM)

Auslesen der Anzahl der Support-Vektoren einer Support-Vektor-Maschine.

get_support_vector_num_class_svm liefert in NumSupportVectors die Anzahl der in dem durch SVMHandle gegebenen SVM abgespeicherten Support-Vektoren zurück. get_support_vector_num_class_svm sollte aufgerufen werden, bevor mit get_support_vector_class_svm die einzelnen Indizes der SV ausgelesen werden, z.B. um zu Visualisieren, welche der Trainingsdaten SV geworden sind (siehe get_support_vector_class_svm). Die Anzahl der SV in jeder einzelnen SVM wird in NumSVPerSVM zurückgegeben. Die Summe der einzelnen Klassifikatoren unterscheidet sich von NumSupportVectors, da einzelne SV-Auswertungen wiederverwendet werden. NumSVPerSVM kann verwendet werden, um die Optimierung der Klassifikationszeit mit reduce_class_svm zu steuern.

Parameter

▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
Handle der SVM.

▷ **NumSupportVectors** (output_control) ..... integer ⇝ *integer*
Gesamtanzahl der Support-Vektoren.

▷ **NumSVPerSVM** (output_control) ..... integer-array ⇝ *integer*
Anzahl der SV jeder einzelnen Sub-SVM.

Ergebnis

Falls SVMHandle korrekt ist, liefert get_sample_num_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_class_svm

Nachfolger

get_sample_class_svm

Siehe auch

create_class_svm

Modul

Foundation

read_class_svm (: : FileName : SVMHandle)

Lesen einer Support-Vektor-Maschine aus einer Datei.

read_class_svm liest eine mit write_class_svm abgespeicherte Support-Vektor-Maschine (SVM) ein. Da das Trainieren einer SVM relativ lange dauern kann, wird die SVM typischerweise in einem Offline-Prozess trainiert und mit write_class_svm gespeichert. Im Online-Prozess wird die SVM mit read_class_svm gelesen und danach zur Klassifikation mit classify_class_svm verwendet. Die Dateiendung für den SVM-Klassifikator ist in HALCON 'gsc'.

Parameter

- ▷ **FileName** (input_control) ..... filename.read ⇝ *string*
  Name der Datei.
  Dateiendung: .gsc
- ▷ **SVMHandle** (output_control) ..... class_svm ⇝ *handle*
  Handle der SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

classify_class_svm, create_class_lut_svm

Alternativen

read_dl_classifier

Siehe auch

create_class_svm, write_class_svm

Modul

Foundation

read_samples_class_svm (: : SVMHandle, FileName :)

Lesen von Trainingsdaten einer Support-Vektor-Maschine aus einer Datei.

read_samples_class_svm liest Trainingsmuster aus der durch FileName gegebenen Datei aus und fügt sie zu der schon in der Support-Vektor-Maschine (SVM) SVMHandle vorhandenen Trainingsmustern hinzu. Die SVM muss zuvor mit create_class_svm erzeugt werden. Wie bei train_class_svm und write_samples_class_svm beschrieben, können read_samples_class_svm, add_sample_class_svm und write_samples_class_svm dazu verwendet werden, einen großen Datensatz von aus Trainingsmustern aufzubauen, um somit die Leistung der SVM durch erneutes Trainieren zu verbessern.

Es ist zu beachten, dass die Trainingsdaten die korrekte Dimensionalität aufweisen. Die in FileName gespeicherten Merkmalsvektoren und Zielvektoren müssen die Längen NumFeatures und NumClasses besitzen, die bei create_class_svm angegeben worden ist. Die Klassenzugehörigkeit wird aus Kompatibilitätsgründen (siehe read_samples_class_mlp) als Vektor gespeichert. Falls die Dimensionen falsch sind, wird eine Fehlermeldung zurückgegeben.

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle der SVM.

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_samples_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SVMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_class_svm

Nachfolger

train_class_svm

Alternativen

add_sample_class_svm

Siehe auch

write_samples_class_svm, clear_samples_class_svm

Modul

Foundation

reduce_class_svm (: : SVMHandle, Method, MinRemainingSV, MaxError : SVMHandleReduced)

Approximieren einer bereits trainierten Support-Vektor-Maschine durch eine reduzierte Support-Vektor-Maschine für schnellere Klassifikation.

Wie bei create_class_svm beschrieben hängt die Klassifikationszeit der SVM von der Anzahl der Kernel-Auswertungen zwischen Support-Vektoren und dem Merkmalsvektor ab. Während die Länge des Datenvektors mit einer Vorverarbeitung wie *'pricipal_components'* oder *'canonical_variates'* (siehe create_class_svm für Details) reduziert werden kann, wird die Gesamtanzahl der SV erst während des Trainings festgestellt. Um die Klassifikationszeit weiter zu reduzieren ist es möglich, die originale separierende Hyperebene mit weniger SV als ursprünglich notwendig zu approximieren. Hierfür wird eine Kopie der originalen SVM, die in SVMHandle übergeben wird, als SVMHandleReduced zurückgegeben. Diese neue SVM hat die gleiche Parametrisierung wie das Original, enthält aber eine kleinere Anzahl von SV. Die Trainingsdaten, die in SVMHandle gespeichert waren, werden hierbei nicht kopiert. Die originale SVM wird durch reduce_class_svm nicht modifiziert.

Das Reduktionsverfahren wird mit Method ausgewählt. Zur Zeit wird nur ein Bottom-Up-Ansatz unterstützt, der iterativ die SV verschmilzt. Der Algorithmus bricht ab, wenn entweder eine minimal Anzahl von SV erreicht ist

(MinRemainingSV) oder der akkumulierte Fehler eine Schwelle überschreitet (MaxError). Dabei ist anzumerken, dass aufgrund der Approximation die Komplexität der Hyperebene reduziert wird, was als Folge eine verschlechterte Klassifikationsrate hat. Das übliche Vorgehen ist deshalb, mit einem kleinen MaxError, z.B. *0.001*, anzufangen und diesen schrittweise zu erhöhen. Um die Reduktionsrate zu steuern, sollte bei jedem Schritt die Anzahl der SV mit get_support_vector_num_class_svm und die Klassifikationsrate mit einem separaten Testdatensatz mit classify_class_svm überprüft werden.

Parameter

- ▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
  Handle der originalen SVM.
- ▷ **Method** (input_control) . . . . . string ⇝ *string*
  Art der Nachbearbeitung, um die Anzahl der SV zu reduzieren.
  Defaultwert: 'bottom_up'
  Werteliste: Method ∈ {'bottom_up'}
- ▷ **MinRemainingSV** (input_control) . . . . . integer ⇝ *integer*
  Minimale Anzahl der verbleibenden SV.
  Defaultwert: 2
  Wertevorschläge: MinRemainingSV ∈ {2, 3, 4, 5, 7, 10, 15, 20, 30, 50}
  Restriktion: MinRemainingSV >= 2
- ▷ **MaxError** (input_control) . . . . . real ⇝ *real*
  Maximal erlaubter Fehler, der durch die Reduktion erzeugt wird.
  Defaultwert: 0.001
  Wertevorschläge: MaxError ∈ {0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05}
  Restriktion: MaxError > 0.0
- ▷ **SVMHandleReduced** (output_control) . . . . . class_svm ⇝ *handle*
  Handle der reduzierten SVM.

Beispiel

```
* Train an SVM
create_class_svm (NumFeatures, 'rbf', 0.01, 0.01, NumClasses,\
                  'one-versus-all', 'normalization', NumFeatures,\
                  SVMHandle)
read_samples_class_svm (SVMHandle, 'samples.mtf')
train_class_svm (SVMHandle, 0.001, 'default')
* Create a reduced SVM
reduce_class_svm (SVMHandle, 'bottom_up', 2, 0.01, SVMHandleReduced)
write_class_svm (SVMHandleReduced, 'classifier.svm')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert train_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_class_svm, get_support_vector_num_class_svm

Nachfolger

classify_class_svm, write_class_svm, get_support_vector_num_class_svm

Siehe auch

train_class_svm

Modul

Foundation

select_feature_set_svm (: : ClassTrainDataHandle, SelectionMethod, GenParamName, GenParamValue : SVMHandle, SelectedFeatureIndices, Score)

Wählt aus einer Liste an Merkmalen eine optimale Untermenge für ein bestimmtes Klassifikationsproblem aus.

select_feature_set_svm wählt die optimalen Merkmale für ein bestimmtes Klassifikationsproblem aus. Die Daten für das Klassifikationsproblem werden mit ClassTrainDataHandle angegeben. Als Klassifikator wird eine Support Vector Machine (SVM) verwendet.

In SVMHandle wird ein Klassifikator zurückgegeben, der mit der resultierenden Merkmalsauswahl parametrisiert und trainiert wurde. Die Merkmalsauswahl wird in SelectedFeatureIndices zurückgegeben.

Der Operator kann für folgende Probleme nützlich sein: Es gibt eine Liste von Merkmalen von denen einige wenige für die Lösung eines Problems ausreichen. So kann mit Hilfe dieses Operators eine fundierte Entscheidung getroffen werden welche dieser Merkmale relevant für das gegebene Problem sind. Außerdem können für bestimmte Verfahren der Merkmalsextraktion verschiedene Parametersätze verglichen werden.

Um das Klassifikationsproblem zu definieren, werden dem Handle ClassTrainDataHandle Merkmalsvektoren hinzugefügt, die aus einzelnen Untergruppen bestehen. Diese Untergruppen müssen mit Hilfe des Operators set_feature_lengths_class_train_data festgelegt werden. Diese Untergruppen werden dann von dem in select_feature_set_svm ausgeführten Selektionsprozess entweder ausgewählt oder abgelehnt, je nachdem ob sie zur Lösung des Klassifikationsproblems beitragen.

Die so definierten Merkmale werden dann in SelectedFeatureIndices mit ihrem Index referenziert, außer es wurden zusätzlich Namen beim Aufruf von set_feature_lengths_class_train_data gesetzt. Ist das der Fall, werden statt der Indizes die Namen der ausgewählten Merkmale zurückgegeben. Wurde set_feature_lengths_class_train_data zuvor nicht aufgerufen, werden die einzelnen Spalten als Merkmale interpretiert.

Für den Selektionsprozess können in SelectionMethod zwei verschiedene Methoden ausgewählt werden: entweder die Greedy-Variante *'greedy'* (das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt) oder die dynamisch oszillierende Suche *'greedy_oscillating'* (Das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt. Danach wird getestet ob eines der hinzugefügten Merkmale entbehrlich ist.). Während *'greedy'* schneller terminiert, kann *'greedy_oscillating'* bessere Ergebnisse erzielen wenn die Dimensionen der Merkmale sehr klein ist oder redundante Daten vorliegen.

Als Optimierungskriterium dient die Klassifikationsrate, die mit einem zweifachen Kreuzvalidierungsverfahren ermittelt wird. Die beste erreichte Klassifikationsrate wird in Score zurückgegeben.

Die Parameter *'nu'* und *'gamma'*, die für das SVM benötigt werden, können über die generischen Parameter GenParamName und GenParamValue entweder direkt auf einen Wert oder auf *'auto'* gesetzt werden. Wenn *'auto'* ausgewählt ist, wird für den entsprechenden Wert auch eine Optimierung durchgeführt. Diese automatische Schätzung kann die Ausführung der Merkmalsauswahl signifikant verlangsamen, was bei größeren Datensätzen zu Rechenzeiten im Bereich von Tagen führen kann. Zusätzlich kann in den generischen Parametern auch der SVM-Modus mit *'mode'* gewählt werden: Es stehen dabei die Varianten *'one-versus-all'* und *'one-versus-one'* zur Verfügung. Eine Erklärung für die beiden Varianten wie auch für *'nu'* und *'gamma'* als Kernelparameter des RBF-Kernels (RBF = radial basis function) findet sich in der Beschreibung von create_class_svm.

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lange sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator select_feature_set_svm zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

▷ **ClassTrainDataHandle** (input_control) . . . . . class_train_data ⇝ *handle*
Handle der Trainingsdaten. Sollte zuvor in Untermerkmale aufgeteilt worden sein.

▷ **SelectionMethod** (input_control) . . . . . string ⇝ *string*
Methode zur Auswahl der besten Merkmale.
Defaultwert: 'greedy'
Werteliste: SelectionMethod ∈ {'greedy', 'greedy_oscillating'}

▷ **GenParamName** (input_control) ..... string(-array) ⇝ *string*
Namen der generischen Parameter zum Steuern des Klassifikators und der Merkmalsauswahl.
Defaultwert: []
Werteliste: GenParamName ∈ {'nu', 'gamma', 'mode'}

▷ **GenParamValue** (input_control) ..... number(-array) ⇝ *real* / integer / string
Werte der generischen Parameter zum Steuern des Klassifikators und der Merkmalsauswahl.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0.02, 0.05, 'auto', 'one-versus-one', 'one-versus-all'}

▷ **SVMHandle** (output_control) ..... class_svm ⇝ *handle*
Trainierter Klassifikator.

▷ **SelectedFeatureIndices** (output_control) ..... string-array ⇝ *string*
Die Indizes der ausgewählten Merkmale.

▷ **Score** (output_control) ..... real-array ⇝ *real*
Die Klassifikationsrate die mit dem ausgewählten Merkmalssatz erreicht wurde.

Beispiel

```
* Find out which of the two features distinguishes two Classes
NameFeature1 := 'Good Feature'
NameFeature2 := 'Bad Feature'
LengthFeature1 := 3
LengthFeature2 := 2
* Create training data
create_class_train_data (LengthFeature1+LengthFeature2,\
  ClassTrainDataHandle)
* Define the features which are in the training data
set_feature_lengths_class_train_data (ClassTrainDataHandle, [LengthFeature1,\
  LengthFeature2], [NameFeature1, NameFeature2])
* Add training data
*                                                        |Feat1| |Feat2|
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  2,1  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  2,1  ], 1)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [1,1,1,  3,4  ], 0)
add_sample_class_train_data (ClassTrainDataHandle, 'row', [2,2,2,  3,4  ], 1)
* Add more data
* ...
* Select the better feature with a SVM
select_feature_set_svm (ClassTrainDataHandle, 'greedy', [], [], SVMHandle,\
  SelectedFeatureSVM, Score)
* Use the classifier
* ...
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert select_feature_set_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

create_class_train_data, add_sample_class_train_data, set_feature_lengths_class_train_data

Nachfolger

classify_class_svm

Alternativen

select_feature_set_mlp, select_feature_set_knn, select_feature_set_gmm

Siehe auch

select_feature_set_trainf_svm, gray_features, region_features

Modul

Foundation

serialize_class_svm (: : SVMHandle : SerializedItemHandle)

Serialisieren einer Support-Vektor-Maschine (SVM).

serialize_class_svm serialisiert eine Support-Vektor-Maschine (SVM) einschließlich eventuell abgespeicherter Trainingsmuster (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_class_svm und write_samples_class_svm in eine Datei geschrieben werden. Die SVM wird in dem Handle SVMHandle übergeben. Die serialisierte SVM wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_class_svm wieder deserialisiert werden.

Parameter

- ▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
  Handle der SVM.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_class_svm

Nachfolger

clear_class_svm, fwrite_serialized_item, send_serialized_item, deserialize_class_svm

Siehe auch

create_class_svm, read_class_svm, write_samples_class_svm, deserialize_class_svm

Modul

Foundation

train_class_svm (: : SVMHandle, Epsilon, TrainMode :)

Trainieren einer Support-Vektor-Maschine.

train_class_svm trainiert die durch SVMHandle angegebene Support Vektor Maschine (SVM). Bevor die SVM trainiert werden kann, müssen mit add_sample_class_svm oder read_samples_class_svm die zum Training zu verwendenden Trainingsmuster zu der SVM hinzugefügt werden.

Da es sich beim Training von SVMs um ein konvexes quadratisches Optimierungsproblem handelt, kann sichergestellt werden, dass das Training nach endlichen Schritten beim globalen Optimum terminiert. Mit dem Parameter Epsilon wird die Schwelle gesetzt, den der Gradient der Funktion, die intern optimiert wird, unterschreiten muss, damit das Training als beendet angesehen werden kann. Normalerweise sollte Epsilon auf *0.001* gesetzt werden, da dies in der Praxis zu den besten Ergebnissen führt. Ein zu großer Wert führt zu einem schnelleren Abbrechen des Trainings und dadurch zu einer sub-optimalen Lösung. Ein zu kleiner Wert führt dazu, dass der Optimierungsalgorithmus unverhältnissmäßig lange dauert, ohne dass sich die Erkennungsrate verändert. Erfahrungsgemäß gibt es zwei Gründe für das Verändern von Epsilon: Bei extrem kleinen Nu, z.B. *0.001* (siehe create_class_svm), kann ein kleineres Epsilon die Erkennungsrate deutlich verbessern. Eine zweiter Fall ist die Bestimmung von optimalen Kernelfunktionen mit ihrer Parametrisierung (z.B. KernelParam-Nu Paare beim RBF-Kernel) bei einem üblicherweise sehr lange dauernden n-fachen Vergleichsprüfungsverfahren (Cross Validation). Wählt man Epsilon beispielsweise als *0.1* so terminiert das Training zwar vorschnell, aber der Bereich der optimalen Kernelfuntion unterscheidet sich nicht wesentlich von dem Bereich, den man bei der Cross Validation mit einem normalen Epsilon erhalten hätte. Nachdem ein optimales KernelParam-Nu Paar ermittelt wurde, kann das endgültige Training mit kleinem Epsilon erfolgen.

Die Dauer des Trainings hängt von der Trainingsdatenmenge, der Anzahl der beim Training entstehenden SV sowie Epsilon ab (und kann zwischen unter einer Sekunde bis mehrere Stunden betragen).

Es empfiehlt sich daher, Nu in create_class_svm so zu wählen, dass möglichst wenige SV beim Training entstehen, ohne dass die Erkennungsrate sich verschlechtert. Insbesondere ist zu beachten, dass für Nu ein nicht zu großer Wert gewählt wird, damit die Optimierung von einem zulässigen Punkt startet. Falls zu viele Trainingsfehler mit einem zu großem Nu gewählt wurde, erfolgt eine Fehlerbehandlung. In diesem Fall muss eine SVM mit dem gleichen Trainingsdatensatz, aber kleinerem Nu trainiert werden.

Mit dem Parameter TrainMode kann zwischen verschiedenen Trainingsmodi unterschieden werden. Normalerweise wird eine SVM ohne weiteres Vorwissen trainiert, und TrainMode wird auf *'default'* gesetzt. Werden für den gleichen Trainingsdatensatz SVM mit verschiedenen Kernels trainiert (siehe oben), ist es mit TrainMode möglich, eine bereits trainierte SVM als Anfangspunkt für das Optimierungsverfahren zu verwenden und somit das Training insbesondere bei sehr großen Trainingsdatensätzen zu beschleunigen. Hierbei muss aber der Trainingsdatensatz, die Anzahl der Klassen sowie der Modus (siehe create_class_svm) übereinstimmen. Hierfür wird der SVMHandle der bereits trainierten Maschine in TrainMode übergeben.

Mit TrainMode = *'add_sv_to_train_set'* ist es möglich, dass beim Training Support-Vektoren, die aus einem früheren Aufruf von train_class_svm entstanden sind, dem Trainingsdatensatz hinzugefügt werden. Dieser TrainMode hat zwei typische Anwendungsgebiete: Erstens ist es möglich, SVM sukzessive zu trainieren. Dabei wird der Trainingsdatensatz zuerst in verschiedene Teile aufgeteilt. Danach wird ein erster Teil normal mit TrainMode = *'default'* trainiert. Anschließend wird mit clear_samples_class_svm der vorherige Trainingsdatensatz verworfen, ein nächster Teil mit add_sample_class_svm übergeben und mit train_class_svm bei TrainMode = *'add_sv_to_train_set'* nachtrainiert. Dies wird solange wiederholt, bis alle Teile des Trainingsdatensatzes trainiert wurden. Dieses Vorgehen hat den Vorteil, dass so sehr große Trainingsdatensätze effizient hinsichtlich des Speicherbedarfs trainiert werden können. Ein weiteres Anwendungsgebiet für TrainMode = *'add_sv_to_train_set'* ist, dass ein genereller Klassifikator durch Hinzufügen von charakteristischen Trainingsmustern und Nachtrainieren spezialisiert werden kann. Beim Nachtrainieren von SVMs mit TrainMode=*'add_sv_to_train_set'* muss beachtet werden, dass die Vorverarbeitung (wie in create_class_svm beschrieben) nicht verändert wird.

Parameter

▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
Handle der SVM.

▷ **Epsilon** (input_control) ..... real ⇝ *real*
Abbruchparameter für das Training.
Defaultwert: 0.001
Wertevorschläge: Epsilon ∈ {0.00001, 0.0001, 0.001, 0.01, 0.1}

▷ **TrainMode** (input_control) ..... number ⇝ *string* / integer
Modus des Trainings. Im Normalfall: 'default'. Falls die bereits enthaltenen SV als Trainingsdaten verwendet werden sollen: 'add_sv_to_train_set'. Im Falle von Alpha seeding das zu verwendende SVM Handle
Defaultwert: 'default'
Werteliste: TrainMode ∈ {'default', 'add_sv_to_train_set'}

Beispiel

```
* Train an SVM
```

```
create_class_svm (NumFeatures, 'rbf', 0.01, 0.01, NumClasses,\
                  'one-versus-all', 'normalization', NumFeatures,\
                  SVMHandle)
read_samples_class_svm (SVMHandle, 'samples.mtf')
train_class_svm (SVMHandle, 0.001, 'default')
write_class_svm (SVMHandle, 'classifier.svm')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `train_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SVMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`add_sample_class_svm`, `read_samples_class_svm`

Nachfolger

`classify_class_svm`, `write_class_svm`, `create_class_lut_svm`

Alternativen

`train_dl_classifier_batch`, `read_class_svm`

Siehe auch

`create_class_svm`

Literatur

John Shawe-Taylor, Nello Cristianini: „Kernel Methods for Pattern Analysis“; Cambridge University Press, Cambridge; 2004.

Bernhard Schölkopf, Alexander J.Smola: „Learning with Kernels“; MIT Press, London; 1999.

Modul

Foundation

write_class_svm (: : SVMHandle, FileName :)

Abspeichern einer Support-Vektor-Maschine in eine Datei.

`write_class_svm` speichert die Support-Vektor-Maschine (SVM) `SVMHandle` in der Datei `FileName` ab. Die Dateiendung für den SVM-Klassifikator ist in HALCON 'gsc'. `write_class_svm` wird typischerweise aufgerufen, nachdem die SVM mit `train_class_svm` trainiert wurde. Die SVM kann mit `read_class_svm` wieder eingelesen werden. `write_class_svm` speichert *nicht* die in der SVM eventuell abgespeicherten Trainingsmuster ab. Dazu ist `write_samples_class_svm` zu verwenden.

Parameter

▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
Handle der SVM.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.gsc`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_class_svm`

Nachfolger

`clear_class_svm`

Siehe auch

`create_class_svm`, `read_class_svm`, `write_samples_class_svm`

Modul

Foundation

write_samples_class_svm (: : SVMHandle, FileName :)

Abspeichern der Trainingsdaten einer Support-Vektor-Maschine in eine Datei.

`write_samples_class_svm` speichert die in der Support-Vektor-Maschine (SVM) `SVMHandle` abgespeicherten Trainingsmuster in der Datei `FileName` ab. `write_samples_class_svm` kann dazu verwendet werden, eine Datenbank mit Trainingsmustern aufzubauen, und somit durch erneutes Trainieren mit einer erweiterten Datenbank die Leistung der SVM zu verbessern (siehe `train_class_svm`). Die Datei `FileName` wird von `write_samples_class_svm` überschrieben. Eine Erweiterung der Datenbank der Trainingsmuster ist aber einfach möglich, da `read_samples_class_svm` und `add_sample_class_svm` die Trainingsmuster zu den bereits im Speicher der SVM gehaltenen Trainingsmustern hinzufügen.

Parameter

▷ **SVMHandle** (input_control) . . . . . class_svm ⇝ *handle*
Handle der SVM.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_samples_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`add_sample_class_svm`

Nachfolger

`clear_samples_class_svm`

Siehe auch

`create_class_svm`, `get_prep_info_class_svm`, `read_samples_class_svm`

Modul

Foundation

Kapitel 16

Legacy

16.1 2D-Messtechnik

copy_metrology_object (: : MetrologyHandle, Index : CopiedIndices)

Kopieren von Metrology-Messobjekten eines Messmodells.

copy_metrology_object ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

copy_metrology_object kopiert Metrology-Messobjekte innerhalb des Messmodells.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch das Handle MetrologyHandle definiert. Der Parameter Index legt die Metrology-Messobjekte fest, die kopiert werden sollen. Wird für den Parameter Index als Wert *'all'* übergeben, werden alle Metrology-Messobjekte kopiert. Der Operator copy_metrology_object gibt den Index der neu erstellten Metrology-Messobjekte im Parameter CopiedIndices zurück. Die Reihenfolge der neu erstellten Metrology-Messobjekte entspricht der Reihenfolge der ursprünglichen Metrology-Messobjekte. Zugriff auf die Parameter der Metrology-Messobjekte des Messmodells ist z.B. mit dem Operator get_metrology_object_param möglich.

Parameter

- ▷ **MetrologyHandle** (input_control) . . . . . metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **Index** (input_control) . . . . . integer(-array) ⇝ *string* / integer
  Index der Metrology-Messobjekte.
  Defaultwert: 'all'
  Wertevorschläge: Index ∈ {'all', 0, 1, 2}
- ▷ **CopiedIndices** (output_control) . . . . . integer(-array) ⇝ *integer*
  Indizes der kopierten Metrology-Messobjekte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert copy_metrology_object den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

2D Metrology

transform_metrology_object (: : MetrologyHandle, Index, Row, Column, Phi, Mode :)

Transformieren von Metrology-Messobjekten eines Messmodells, z.B. für ein Alignment.

transform_metrology_object ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den Operator align_metrology_model verwenden.

transform_metrology_object verschiebt die mit dem Index Index angegebenen Metrology-Messobjekte in eine Position, die mit Row und Column sowie dem Rotationswinkel Phi festgelegt ist.

Das Konzept der 2D-Messtechnik ist bei der Einleitung zu Kapitel 2D-Messtechnik beschrieben.

Das Messmodell ist durch MetrologyHandle definiert. MetrologyHandle ist ein Handle welches von create_metrology_model zurückgegeben wird.

Der Index Index ist der Index der von den Operatoren add_metrology_object_circle_measure, add_metrology_object_ellipse_measure, add_metrology_object_line_measure oder add_metrology_object_rectangle2_measure zurückgegeben wird. Dem Parameter Index darf ein einzelner Wert oder ein Tupel von Werten übergeben werden. Sollen alle Metrology-Messobjekte transformiert werden, muss Index auf den Wert *'all'* gesetzt werden.

Der Parameter Mode legt die Wirkung der Transformation fest. Falls der Wert auf *'absolute'* gesetzt ist, werden die Metrology-Messobjekte auf die Werte der Bildkoordinaten Row und Column im Bezug auf das Bildkoordinatensystem transformiert sowie um Phi rotiert. Falls der Wert *'relative'* gesetzt ist, werden die Werte für die Transformation als relative Werte betrachtet, d.h. sie sind relativ zur bisherigen Position und Rotation der Metrology-Messobjekte festgelegt.

Für ein Metrology-Messobjekt vom Typ Ellipse oder Rechteck ist der Ursprung der Rotation im Mittelpunkt des Messmodells definiert, welcher von den Operatoren add_metrology_object_ellipse_measure oder add_metrology_object_rectangle2_measure festgelegt ist. Ist das Metrology-Messobjekt eine Linie und ist Mode = *'relative'*, liegt der Ursprung der Rotation im Startpunkt der Linie. Wenn das Metrology-Messobjekt eine Linie ist und Mode auf *'absolute'* gesetzt ist oder wenn das Metrology-Messobjekt ein Kreis ist, wird der Parameter Phi ignoriert.

transform_metrology_object ist schneller, wenn keine Rotation definiert ist und wenn die Messregionen und die transformierten Messregionen beide vollständig innerhalb des Bildes liegen. In allen anderen Fällen werden die Messregionen neu erzeugt und der Operator wird langsamer.

Achtung

Zu beachten ist, dass, wenn Ergebnisse des Operators apply_metrology_model (angepasste geometrische Formen) im Metrology-Messobjekt vorhanden sind, diese Ergebnisse beim Aufruf des Operators gelöscht werden.

Parameter

- ▷ **MetrologyHandle** (input_control) ..... metrology_model ⇝ *handle*
  Handle des Messmodells.
- ▷ **Index** (input_control) ..... integer(-array) ⇝ *string* / integer
  Index der Metrology-Messobjekte.
  Defaultwert: 'all'
  Wertevorschläge: Index ∈ {'all', 0, 1, 2}
- ▷ **Row** (input_control) ..... real(-array) ⇝ *real* / integer
  Verschiebung in Zeilenrichtung.
- ▷ **Column** (input_control) ..... real(-array) ⇝ *real* / integer
  Verschiebung in Spaltenrichtung.
- ▷ **Phi** (input_control) ..... real(-array) ⇝ *real* / integer
  Rotationswinkel.
- ▷ **Mode** (input_control) ..... string(-array) ⇝ *string*
  Modus der Transformation.
  Defaultwert: 'absolute'
  Wertevorschläge: Mode ∈ {'absolute', 'relative'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert transform_metrology_object den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MetrologyHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Modul

2D Metrology

16.2 DL Klassifikation

Der Ablauf und die Operatoren dieses Kapitels sind veraltet und werden nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den Ablauf und die neuen Operatoren verwenden, die in Deep Learning / Klassifikation beschrieben sind.

Dieses Kapitel beschreibt, wie Deep Learning-basierte Klassifikation in der Trainings- und Inferenzphase genutzt wird.

Ablauf

Dieser Abschnitt beschreibt die wichtigsten Schritte und den Ablauf der Deep Learning-basierten Klassifikation mit dem veralteten Ablauf.

Vorbereitung des Netzwerks und der Daten

1. Erst muss ein vortrainierter Klassifikator mit dem Operator
   - `read_dl_classifier`.

   eingelesen werden. Dieser Operator wird auch dazu verwendet, um fertig trainierte Klassifikatoren, die mit `write_dl_classifier` geschrieben wurden, einzulesen.
2. Um die Daten für das Deep Learning Klassifikationstraining zu laden, steht die Prozedur
   - `read_dl_classifier_data_set`

   zur Verfügung. Diese Prozedur gibt eine Liste von Pfaden der Bilddateien, eine Liste der zugehörigen Ground Truth Labels (wahren Klassen) sowie eine Liste aller vorkommenden Klassen zurück.
3. Der Klassifikator stellt mehrere Anforderungen an die Bilder, wie die Bilddimensionen und den Grauwertbereich. Die Defaultwerte sind in `read_dl_classifier` aufgelistet. Mit diesen Werten wurden die Netzwerke vortrainiert. Die Netzwerkarchitekturen erlauben verschiedene Bilddimensionen, welche über `set_dl_classifier_param` gesetzt werden können, aber je nach Netzwerk macht eine Änderung ein Nachtrainieren notwendig. Die aktuellen Werte können mit
   - `get_dl_classifier_param`.

   abgefragt werden.

   Die Prozedur `preprocess_dl_classifier_images` hilft bei der Implementierung einer entsprechenden Vorverarbeitung. Wir empfehlen, alle für das Training benötigten Bilder vor dem Training vorzuverarbeiten und abzuspeichern, was das Trainieren signifikant beschleunigt.
4. Als nächstes wird empfohlen die Daten in drei getrennte Datensätze zu teilen, die für Training, Validierung und Testen verwendet werden (siehe Abschnitt „Daten“ im Kapitel Deep Learning). Dazu kann die Prozedur
   - `split_dl_classifier_data_set`.

   verwendet werden.
5. Dem Klassifikator muss mitgeteilt werden, welche Klassen er unterscheiden soll. Dazu werden die zuvor mit `read_dl_classifier_data_set` eingelesenen Klassen mit dem Operator

- set_dl_classifier_param

und dem Parameter *'classes'* gesetzt.

Dieser Operator kann auch verwendet werden, um Hyperparameter, die für das Training wichtig sind (z.B. *'batch_size'* und *'learning_rate'*), zu setzen. Für eine detailliertere Erklärung siehe das Kapitel Deep Learning und die Dokumentation von set_dl_classifier_param.

Training des Klassifikators und Auswertung des Trainingsfortschritts Sobald der Klassifikator und der Datensatz vorbereitet sind, kann der Klassifikator für die neue Aufgabe nachtrainiert werden.

1. Setzen der Hyperparameter, die für das Training verwendet werden, mit dem Operator

   - set_dl_classifier_param.

   Für einen Überblick über mögliche Hyperparameter siehe set_dl_classifier_param. Zusätzliche Erklärungen können im Kapitel Deep Learning gefunden werden.

2. Zum Trainieren des Klassifikators steht der Operator

   - train_dl_classifier_batch

   zur Verfügung. Zwischenergebnisse des Trainings werden im Ausgabehandle dieses Operators gespeichert.

   Wie der Name suggeriert, verarbeitet der Operator train_dl_classifier_batch einen Batch von Trainingsdaten (Bilder und Ground Truth Labels) in einem Schritt. Dafür wird über die Trainingsdaten iteriert, um den Klassifikator sukzessiv mit train_dl_classifier_batch zu trainieren. Dieser Prozess wird für so viele Epochen wiederholt, bis das Trainingsergebnis zufriedenstellend ist.

3. Um den Trainingsfortschritt zu veranschaulichen, wird die Prozedur

   - plot_dl_classifier_training_progress

   bereitgestellt. Damit kann der Klassifikationsfehler während des Trainings visualisiert werden. Die zur Darstellung notwendigen Eingaben können über die Prozeduren

   - select_percentage_dl_classifier_data,
   - apply_dl_classifier_batchwise und
   - evaluate_dl_classifier

   erhalten werden. Mit diesen Prozeduren kann die Anzahl der Bilder für die Evaluierung verringert werden, der Klassifikator auf die ausgewählten Daten angewendet werden und zum Beispiel der Top-1 Fehler berechnet werden.

Anwenden und Evaluieren des finalen Klassifikators Nun ist der Klassifikator für die spezifische Anwendung trainiert und bereit zur Anwendung. Doch zuvor sollte noch evaluiert werden, wie gut sich der Klassifikator auf den Testdaten verhält.

1. Um den Klassifikator auf einer beliebigen Anzahl Bilder anzuwenden, wird der Operator

   - apply_dl_classifier.

   verwendet. Die Laufzeit dieses Operators hängt von der Anzahl von Batches ab, die für den gegebenen Bilddatensatz benötigt werden.

   Die Ergebnisse werden in einem Handle gespeichert.

   Die vorausgesagten Klassen und Konfidenzen werden mit dem Operator

   - get_dl_classifier_result.

   abgefragt.

2. Jetzt können die Ergebnisse ausgewertet werden. Die Leistung des Klassifikators kann während des Trainings mit der Prozedur evaluate_dl_classifier ausgewertet werden.

   Um die Qualität des Klassifikators zu analysieren und zu visualisieren, ist die Konfusionsmatrix ein nützliches Hilfsmittel (für eine Erklärung wird auf das Kapitel Deep Learning verwiesen). Dafür können die Prozeduren

   - gen_confusion_matrix
   - gen_interactive_confusion_matrix.

verwendet werden. Die interaktive Prozedur ermöglicht es, Beispiele einer spezifischen Kategorie auszuwählen, funktioniert aber nicht mit exportiertem Code.

Zusätzlich kann, nachdem der Klassifikator auf einen Datensatz angewandt wurde, die Prozedur

- `get_dl_classifier_image_results`

verwendet werden, um Bilder nach bestimmten Kriterien zu selektieren und anzuzeigen, zum Beispiel Bilder die falsch klassifiziert wurden. Weiter kann dies als Eingabe für die Prozedur

- `dev_display_dl_classifier_heatmap`,

verwendet werden, um eine Heatmap für ein Bild anzuzeigen. Mit dieser Heatmap kann analysiert werden, welche Bildregionen für das Klassifikationsergebnis ausschlaggebend sind.

Inferenzphase Ist der Klassifikator trainiert und seine Leistung zufriedenstellend, kann er für die Klassifikation neuer Bilder verwendet werden. Dafür müssen die Bilder entsprechend den Anforderungen des Klassifikators vorverarbeitet werden (d.h. auf dieselbe Art und Weise wie für das Training). Danach können sie mit

- `apply_dl_classifier`.

klassifiziert werden.

Daten für die Klassifikation

Es wird zwischen Daten für Training und Inferenz unterschieden. Letztere bestehen ausschließlich aus Bildern. Für Erstere ist jedoch bereits bekannt zu welcher Klasse die Bilder gehören. Die entsprechenden Informationen werden über die Labels zur Verfügung gestellt.

Die Trainingsdaten werden dazu genutzt einen Klassifikator für eine spezifische Aufgabe zu trainieren. Mit Hilfe dieser Daten kann der Klassifikator lernen, welche Klassen zu unterscheiden sind und wie deren Vertreter aussehen. Bei der Klassifikation wird das Bild als Ganzes klassifiziert. Daher bestehen die Trainingsdaten aus Bildern und deren Ground Truth Labels, also die Klasse zu dem das Bild gehört. Zu beachten ist hierbei, dass die Bilder möglichst repräsentativ für die spätere Aufgabe sein sollten. Es gibt verschiedene Möglichkeiten, wie die Ground Truth Klassen als Label gespeichert und abgerufen werden können. Auf folgende Weisen können die Ground Truth Labels eines Bildes von der Prozedur `read_dl_classifier_data_set` gelesen werden:

- Als Name des letzten Ordners, der das Bild enthält
- Als Dateiname.

Zum Trainieren eines Klassifikators wird eine Technik namens Transferlernen verwendet (siehe das Kapitel Deep Learning). Die Anzahl benötigter Bilder ist dabei zwar geringer, für einen geeigneten Datensatz wird üblicherweise aber immer noch eine Anzahl in den Hundertern bis Tausendern pro Klasse benötigt. Während der Klassifikator generell verlässlicher ist, wenn er mit einem größeren Datensatz trainiert wurde, hängt die Anzahl benötigter Bilder auch von der Komplexität der Aufgabe ab. Es sollten außerdem genug Trainingsdaten vorliegen um diese in die drei Untermengen aufzuteilen, welche idealerweise unabhängig und gleich verteilt sind (siehe Abschnitt „Daten“ im Kapitel Deep Learning).

Unabhängig von der Anwendung, stellt der Klassifikator Anforderungen an die Bilder bezüglich Bilddimensionen, Grauwertbereich und Bildtyp. Die spezifischen Werte sind abhängig vom Klassifikator selbst und können mit `get_dl_classifier_param` abgefragt werden. Eine Orientierung bei der Implementierung einer entsprechenden Vorverarbeitung gibt die Prozedur `preprocess_dl_classifier_images`.

apply_dl_classifier (
Images : : DLClassifierHandle : DLClassifierResultHandle)

Klassifizieren einer Menge an Bildern durch einen Deep Learning-basierten Klassifikator.

`apply_dl_classifier` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator `apply_dl_model` verwenden.

apply_dl_classifier klassifiziert die Menge an Bildern Images durch des Deep Learning-basierten Klassifikators DLClassifierHandle. Die daraus resultierende Ergebnisse sind in DLClassifierResultHandle abgespeichert und können mit get_dl_classifier_result abgefragt werden.

Die Menge Images kann eine beliebige Anzahl an Bildern enthalten, die in einem Operatoraufruf verarbeitet werden sollen, und ist im Allgemeinen unabhängig von dem Parameter *'batch_size'*. Beachten Sie bitte, dass dies nur für apply_dl_classifier gilt und nicht für train_dl_classifier_batch. Der Operator apply_dl_classifier klassifiziert stets einen Batch mit bis zu *'batch_size'* Bildern simultan, unabhängig davon, ob dieser aufgefüllt ist oder nicht. Falls die Menge Images mehr als *'batch_size'* Bilder enthält, iteriert apply_dl_classifier über die nötige Anzahl Batches der Größe *'batch_size'*, um alle Bilder in Images zu verarbeiten. Bei einer Menge Images mit weniger als *'batch_size'* Bildern wird diese bis zu einem vollen Batch aufgefüllt, was bedeutet, dass die Laufzeit zur Verarbeitung eines solchen Batches unabhängig davon ist, ob dieser gefüllt ist oder nur ein einzelnes Bild enthält. Außerdem benötigt das Netzwerk stets den vollen Speicher, selbst wenn in einem Operatorenaufruf weniger als *'batch_size'* Bilder klassifiziert werden sollen. Deswegen ist es bei der Wahl der *'batch_size'* zu empfehlen, die Anzahl der Bilder, die in einem Operatorenaufruf übergeben werden, mit zu berücksichtigen, um eine größere Effizienz zu erzielen. Der aktuelle Wert von *'batch_size'* kann mittels get_dl_classifier_param abgefragt werden.

Es ist zu beachten, dass die Bilder für die Klassifikation Anforderungen erfüllen müssen betreffend Größe, Grauwertbereich, Anzahl Kanäle und Typ. Die genauen Bildanforderungen des Netzwerkes können mit get_dl_classifier_param abgerufen werden. Besonders hervorheben möchten wir den Datentyp: Die Bilder müssen vom Typ 'real' sein. Für eine möglicherweise notwendige Umwandlung steht der Operator convert_image_type zur Verfügung. Die Prozedur preprocess_dl_classifier_images hilft Ihnen bei der Vorverarbeitung der Bilder entsprechend den spezifischen Anforderungen.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Achtung

Um diesen Operator ausführen zu können, werden cuDNN und cuBLAS benötigt, falls *'runtime'* auf *'gpu'* gesetzt ist. Für weitere Details wird auf den „Installation Guide“, Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods“, verwiesen.

Parameter

- ▷ **Images** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* real
  Tupel von Eingabebilder.
- ▷ **DLClassifierHandle** (input_control) ..... dl_classifier ⇝ *handle*
  Handle des Deep Learning-basierten Klassifikators.
- ▷ **DLClassifierResultHandle** (output_control) ..... dl_classifier_result ⇝ *handle*
  Handle der Ergebnisse eines Deep Learning-basierten Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert apply_dl_classifier den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_dl_classifier, train_dl_classifier_batch, set_dl_classifier_param

Nachfolger

get_dl_classifier_result, clear_dl_classifier

Alternativen

apply_dl_model, classify_class_mlp, classify_class_svm

Modul

Deep Learning Inference

clear_dl_classifier (: : DLClassifierHandle :)

Löschen eines Deep Learning-basierten Klassifikators.

clear_dl_classifier ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator clear_dl_model verwenden.

clear_dl_classifier löscht das Neuronale Netzwerk, das durch DLClassifierHandle angegeben wird, und gibt den entsprechenden Speicherplatz frei. Das Netzwerk kann nach dem Aufruf von clear_dl_classifier nicht mehr verwendet werden. Das Handle DLClassifierHandle ist nach dem Aufruf ungültig.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

▷ **DLClassifierHandle** (input_control) ..... dl_classifier(-array) ⇝ *handle*
Handle des Deep Learning-basierten Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert clear_dl_classifier den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DLClassifierHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

read_dl_classifier, apply_dl_classifier, train_dl_classifier_batch

Modul

Deep Learning Inference

clear_dl_classifier_result (: : DLClassifierResultHandle :)

Löscht das Handle mit den Resultaten der Deep Learning-basierten Klassifikation.

clear_dl_classifier_result ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

clear_dl_classifier_result löscht das Handle DLClassifierResultHandle mit den Ergebnissen der Deep Learning-basierten Klassifikation und gibt den entsprechenden Speicher frei. Nach der Aufruf von clear_dl_classifier_result sind die Ergebnisse gelöscht, und können nicht mehr abgefragt werden. Das Handle DLClassifierResultHandle wird entsprechend ungültig.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

▷ **DLClassifierResultHandle** (input_control) . . . . . dl_classifier_result(-array) ⇝ *handle*
Handle der Ergebnisse eines Deep Learning-basierten Klassifikators.

Ergebnis

Falls DLClassifierResultHandle korrekt ist, dann liefert clear_dl_classifier_result den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_dl_classifier_result, apply_dl_classifier

Modul

Deep Learning Inference

clear_dl_classifier_train_result (
: : DLClassifierTrainResultHandle :)

Löschen eines Handles von einem Trainingsschritt eines Deep Learning-basierten Klassifikators und Freigabe des verwendeten Speichers.

clear_dl_classifier_train_result ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

Der Operator clear_dl_classifier_train_result löscht ein oder mehrere Handles, die zuvor mit train_dl_classifier_batch erzeugt wurden. Dabei wird der gesamte Speicher wieder freigegeben.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

▷ **DLClassifierTrainResultHandle** (input_control) . . . . . dl_classifier_train_result(-array) ⇝ *handle*
Handle des Trainings Resultate des Deep Learning-basierten Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_dl_classifier_train_result den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_dl_classifier_batch, get_dl_classifier_train_result

Modul

Deep Learning Training

deserialize_dl_classifier (: : SerializedItemHandle : DLClassifierHandle)

Deserialisiert einen Deep Learning-basierten Klassifikator.

deserialize_dl_classifier ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator deserialize_dl_model verwenden.

Der Operator deserialize_dl_classifier deserialisiert einen Deep Learning-basierten Klassifikator, der mit dem Operator serialize_dl_classifier serialisiert wurde.

Der Operator verhält sich wie read_dl_classifier mit dem Unterschied, dass keine Datei, sondern ein serialisiertes Element als Eingabe dient.

Für eine detailliertere Beschreibung sei auf die Dokumentation des Operators read_dl_classifier verwiesen.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **DLClassifierHandle** (output_control) . . . . . dl_classifier ⇝ *handle*
Handle des Deep Learning-basierten Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_dl_classifier den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_dl_classifier

Nachfolger

apply_dl_classifier, train_dl_classifier_batch, set_dl_classifier_param, get_dl_classifier_param

Alternativen

deserialize_dl_model

Siehe auch

serialize_dl_classifier

Modul

Deep Learning Inference

get_dl_classifier_param (: : DLClassifierHandle, GenParamName : GenParamValue)

Auslesen der Parameter eines Deep Learning-basierten Klassifikators.

get_dl_classifier_param ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator get_dl_model_param verwenden.

get_dl_classifier_param gibt die Parameterwerte von GenParamName des Neuronalen Netzwerkes DLClassifierHandle in GenParamValue aus.

Die Hyperparameter und Netzwerkparameter können über den Operator set_dl_classifier_param gesetzt werden, in dessen Dokumentation sie im Detail beschrieben sind. Mit get_dl_classifier_param können all diese Werte abgefragt werden.

Zusätzlich gibt es Parameter, die vom Neuronalen Netzwerk vorgegeben und somit nur lesbar sind. Diese Parameter sind:

'image_range_min': Minimaler Grauwert.

'image_range_max': Maximaler Grauwert.

Die spezifischen Werte für diese Parameter sowie die Defaultwerte der Bilddimensionen hängen vom jeweiligen Netzwerk ab, siehe read_dl_classifier.

Jedes Bild, welches an das Netzwerk übergeben wird, muss entsprechend der aktuellen Anforderungen des Netzwerkes vorliegen. Um Bilder entsprechend vorzuverarbeiten, steht die Prozedur preprocess_dl_classifier_images zur Verfügung.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

▷ **DLClassifierHandle** (input_control) ..... dl_classifier ⇝ *handle*
Handle des Deep Learning-basierten Klassifikators.

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'gpu'
Werteliste: GenParamName ∈ {'batch_size', 'batch_size_multiplier', 'classes', 'gpu', 'image_dimensions', 'image_num_channels', 'image_height', 'image_range_max', 'image_range_min', 'image_width', 'learning_rate', 'momentum', 'runtime', 'runtime_init', 'weight_prior'}

▷ **GenParamValue** (output_control) ..... attribute.name(-array) ⇝ *integer* / string / real
Wert des generischen Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_dl_classifier_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_dl_classifier, set_dl_classifier_param

Nachfolger

train_dl_classifier_batch, apply_dl_classifier

Alternativen

get_dl_model_param

Siehe auch

set_dl_classifier_param

Modul

Deep Learning Inference

get_dl_classifier_result (: : DLClassifierResultHandle, Index, GenResultName : GenResultValue)

Abrufen der Inferenz-Ergebnisse eines Deep Learning-basierten Klassifikators.

get_dl_classifier_result ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

get_dl_classifier_result ermöglicht den Abruf der Inferenz-Resultate. Diese Resultate sind in DLClassifierResultHandle gespeichert und wurden über apply_dl_classifier erhalten.

Die Eingabeparameter GenResultName und Index geben an, welche Ergebnisse für welches Bild zurückgegeben werden sollen. Das entsprechende Ergebnis wird in GenResultValue gegeben.

GenResultName kann folgende Werte annehmen:

'confidences': gibt die Konfidenzwerte passend zum ausgewählten Bildindex Index zurück.

'predicted_classes': gibt die Klassennamen passend zum ausgewählten Bildindex Index zurück.

'predicted_class_indices': gibt die Klassenindizes passend zum ausgewählten Bildindex Index zurück. Dabei geben die Klassenindizes die jeweiligen Positionen (von *0* startend) in dem Tupel der Klassen an.

Wenn der Parameter Index den Wert *'all'* annimmt, gibt get_dl_classifier_result den Konfidenzwert, bzw. den Klassennamen, des besten Kandidaten (Vorhersage mit größtem Konfidenzwert) eines jeden Bildes. Ansonsten muss Index eine Ganzzahl aus dem Intervall *0* bis Batchgröße minus eins sein und gibt an, für welches Bild des Batches alle Konfidenzwerten, bzw. Klassennamen, zurückgegeben werden sollen. Diese Rückgabewerte sind nach absteigendem Konfidenzwert sortiert. Das ist z.B. nützlich bei der Bestimmung der besten Kandidaten einer Menge an Bildern.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

- ▷ **DLClassifierResultHandle** (input_control) ..... dl_classifier_result ⇝ *handle*
  Handle der Ergebnisse eines Deep Learning-basierten Klassifikators.
- ▷ **Index** (input_control) ..... number ⇝ *string* / integer
  Index des Bildes innerhalb des Batches.
  Defaultwert: 'all'
- ▷ **GenResultName** (input_control) ..... attribute.name(-array) ⇝ *string*
  Name des generischen Parameters.
  Defaultwert: 'predicted_classes'
  Werteliste: GenResultName ∈ {'predicted_classes', 'predicted_class_indices', 'confidences'}
- ▷ **GenResultValue** (output_control) ..... attribute.value(-array) ⇝ *real* / string / integer
  Wert des generischen Parameters, entweder die Konfidenzwerte, die Klassennamen oder Klassenindizes.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_dl_classifier_result den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

apply_dl_classifier

Nachfolger

clear_dl_classifier_result

Modul

Deep Learning Inference

get_dl_classifier_train_result (: : DLClassifierTrainResultHandle, GenParamName : GenParamValue)

Auslesen der Resultate eines Trainingsschrittes des Deep Learning-basierten Klassifikators.

get_dl_classifier_train_result ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

get_dl_classifier_train_result gibt den Wert GenParamValue des Resultats GenParamName aus dem Handle DLClassifierTrainResultHandle aus.

Folgende Parameter können durch GenParamValue abgefragt werden:

'loss' **oder:** *'total_loss'* Aktueller Wert der Zielfunktion. Für eine detaillierte Beschreibung der Zielfunktion siehe train_dl_classifier_batch.

'mll_loss': Aktueller Wert der Zielfunktion, d.h. *'loss'*, ohne *'regularization_loss'*.

'regularization_loss': Aktueller Wert des Regularisierungsterms der Zielfunktion, d.h. *'loss'* ohne *'mll_loss'*.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

- ▷ **DLClassifierTrainResultHandle** (input_control) . . . . . dl_classifier_train_result ⇝ *handle*
  Handle des Trainings Resultate des Deep Learning-basierten Klassifikators.
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Name des generischen Parameters.
  Defaultwert: 'loss'
  Werteliste: GenParamName ∈ {'loss', 'mll_loss', 'regularization_loss', 'total_loss'}
- ▷ **GenParamValue** (output_control) . . . . . attribute.name(-array) ⇝ *real* / integer / string
  Wert des generischen Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_dl_classifier_train_result den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_dl_classifier_batch

Nachfolger

clear_dl_classifier_train_result

Modul

Deep Learning Training

read_dl_classifier (: : FileName : DLClassifierHandle)

Lesen eines Deep Learning-basierten Klassifikators aus einer Datei.

read_dl_classifier ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator read_dl_model verwenden.

Der Operator read_dl_classifier liest ein Neuronales Netzwerk, welches mit write_dl_classifier geschrieben wurde. Als Ergebnis wird das Handle DLClassifierHandle zurückgegeben.

HALCON stellt Neuronale Netzwerke bereit, welche auf einem großen Datensatz vortrainiert wurden. Diese Neuronalen Netzwerke sind gute Ausgangspunkte, um damit eigene Klassifikatoren für ein individuelles Klassifikationsproblem zu trainieren. Zur Verfügung stehen folgende Netzwerke:

'pretrained_dl_classifier_compact.hdl': Dieses Neuronale Netzwerk ist besonders effizient bezüglich des Speicherverbrauchs und der Laufzeit.

Der Klassifikator erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert das Neuronale Netzwerk spezifische Bildeigenschaften, welche über `get_dl_classifier_param` abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen der Klassifikator trainiert wurde:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Dieses Neuronale Netzwerk enthält keinen Fully-connected Layer. Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen, verlangt aber eine minimale Bildbreite *'image_width'* von 15 Pixeln und eine minimale Bildhöhe von *'image_width'* 15 Pixeln.

'pretrained_dl_classifier_enhanced.hdl': Dieses Neuronale Netzwerk hat mehr verborgene Layer als *'pretrained_dl_classifier_compact.hdl'* und ist daher für komplexere Klassifikationsaufgaben in der Regel besser geeignet. Dafür ist dieses Netzwerk rechen- und speicheraufwändiger. Dies führt zum Beispiel dazu, dass man im Vergleich zum obigen kompakten Netzwerk während des Trainings nur mit einer kleineren Batchgröße trainieren kann, siehe `set_dl_classifier_param`.

Der Klassifikator erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert das Neuronale Netzwerk spezifische Bildeigenschaften, welche über `get_dl_classifier_param` abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen der Klassifikator trainiert wurde:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen, verlangt aber eine minimale Bildbreite *'image_width'* von 47 Pixeln und eine minimale Bildhöhe von *'image_width'* 47 Pixeln. Von der Netzwerkarchitektur her gibt es keine Beschränkung der Bildgröße nach oben, aber zunehmende Bildgrößen erhöhen den Speicher- und Laufzeitbedarf erheblich. Jede Änderung der Bildgröße führt zu einem Reinitialisieren der Gewichte in den Fully-connected Layern und macht dadurch ein Nachtrainieren notwendig.

'pretrained_dl_classifier_resnet50.hdl': Wie das Neuronale Netzwerk *'pretrained_dl_classifier_enhanced'* ist dieser Klassifikator für komplexere Aufgaben geeignet. Aber dieser Klassifikator hat aufgrund der verschiedenen Struktur den Vorteil, das Training stabiler zu machen und intern robuster zu sein.

Der Klassifikator erwartet Eingabebilder vom Typ *'real'*. Zusätzlich erfordert das Neuronale Netzwerk spezifische Bildeigenschaften, welche über `get_dl_classifier_param` abgefragt werden können. Hier werden die Defaultwerte gelistet, mit welchen der Klassifikator trainiert wurde:

'image_width': 224
'image_height': 224
'image_num_channels': 3
'image_range_min': -127
'image_range_max': 128

Die Netzwerkarchitektur erlaubt Änderungen der Bilddimensionen, aber eine minimale Bildbreite *'image_width'* von 32 Pixeln und eine minimale Bildhöhe von *'image_width'* 32 Pixeln wird empfohlen. Von der Netzwerkarchitektur her gibt es keine Beschränkung der Bildgröße nach oben, aber zunehmende Bildgrößen erhöhen den Speicher- und Laufzeitbedarf erheblich. Trotz Fully-connected Layer führt eine Änderung der Bildgröße nicht zu einem Reinitialisieren der Gewichte.

Oben aufgelistet sind die default Bilddimensionen und Grauwertbereiche, mit denen die Klassifikatoren trainiert wurden. Die Netzwerkarchitekturen erlauben jedoch auch verschiedene Bildgrößen, welche über `set_dl_classifier_param` gesetzt werden können. Für Netzwerke mit Fully-connected Layern wird dadurch ein Nachtrainieren notwendig. Netzwerke ohne Fully-connected Layer sind direkt auf verschiedenen Bildgrößen anwendbar. Jedoch wird für Bilder, deren Größe von der Bildgröße der Trainingsbilder abweicht, eine geringere Genauigkeit bei der Klassifikation erwartet.

Die konkreten Bilddimensionen des Netzwerkes können über get_dl_classifier_param abgefragt werden. Jedes Eingabebild muss den Anforderungen des Netzwerkes entsprechend übergeben werden. Für eine solche Vorverarbeitung steht die Prozedur preprocess_dl_classifier_images bereit.

Typischerweise ist es einfacher, schneller und besser, ein vortrainiertes Netzwerk zur Lösung einer Klassifikationsaufgabe zu verwenden, da es bereits generell nützliche Merkmale erlernt hat. Um ein bereits vortrainiertes Netzwerk nachzutrainieren, muss lediglich der Parameter *'classes'* des Klassifikators mittels set_dl_classifier_param verändert werden.

Das Neuronale Netzwerk wird aus der Datei FileName gelesen. Diese Datei wird sowohl im aktuellen Verzeichnis als auch im Ordner ($HALCONROOT/dl/) gesucht.

Man beachte, dass der laufzeitrelevante Parameter *'gpu'* des Klassifikators nicht in die Datei geschrieben wird. Stattdessen wird der Parameter beim Lesen mit dem Standardwert *0* initialisiert.

Die Standard-Dateiendung eines Deep Learning-basierten Klassifikators ist *'.hdl'*.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname.
Defaultwert: 'pretrained_dl_classifier_compact.hdl'
Werteliste: FileName ∈ {'pretrained_dl_classifier_compact.hdl', 'pretrained_dl_classifier_enhanced.hdl', 'pretrained_dl_classifier_resnet50.hdl'}
Dateiendung: .hdl

▷ **DLClassifierHandle** (output_control) . . . . . dl_classifier ⇝ *handle*
Handle des Deep Learning-basierten Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_dl_classifier den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

set_dl_classifier_param, get_dl_classifier_param, apply_dl_classifier, train_dl_classifier_batch

Alternativen

read_dl_model, read_class_mlp, read_class_svm

Modul

Deep Learning Inference

serialize_dl_classifier (
: : DLClassifierHandle : SerializedItemHandle)

Serialisiert einen Deep Learning-basierten Klassifikator.

serialize_dl_classifier ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator serialize_dl_model verwenden.

Der Operator serialize_dl_classifier serialisiert einen Deep Learning-basierten Klassifikator (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Der Klassifkator wird durch DLClassifierHandle definiert. Der serialisierte Klassifikator wird über den Parameter

SerializedItemHandle zurückgegeben und kann durch den Operator deserialize_dl_classifier deserialisiert werden.

Der Operator verhält sich genau wie write_dl_classifier mit dem Unterschied, dass keine Datei, sondern ein serialisiertes Element als Ausgabe generiert wird.

Für eine detailliertere Beschreibung sei auf die Dokumentation des Operators write_dl_classifier verwiesen.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

▷ **DLClassifierHandle** (input_control) ..... dl_classifier ⇝ *handle*
Handle des Deep Learning-basierten Klassifikators.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_dl_classifier_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_dl_classifier_batch

Nachfolger

apply_dl_classifier, train_dl_classifier_batch, fwrite_serialized_item, send_serialized_item, set_dl_classifier_param, get_dl_classifier_param

Alternativen

serialize_dl_model

Siehe auch

deserialize_dl_classifier, apply_dl_classifier, train_dl_classifier_batch

Modul

Deep Learning Inference

set_dl_classifier_param (: : DLClassifierHandle, GenParamName, GenParamValue :)

Setzen der Parameter des Deep Learning-basierten Klassifikators.

set_dl_classifier_param ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator set_dl_model_param verwenden.

set_dl_classifier_param setzt die Parameter und Hyperparameter GenParamName des Neuronalen Netzwerkes DLClassifierHandle auf die Werte in GenParamValue.

GenParamName kann folgende Werte annehmen:

Die vortrainierten Klassifikatoren sind mit ihren default Bilddimensionen trainiert, siehe read_dl_classifier.

Die Netzwerkarchitekturen erlauben verschiedene Bildgrößen. Aber für Netzwerke mit Fully-connected Layern macht eine Änderung der Bildgröße ein Nachtrainieren notwendig. Netzwerke ohne Fully-connected Layer sind direkt auf verschiedene Bildgrößen anwendbar. Jedoch wird für Bilder, deren Größe von der Bildgröße der Trainingsbilder abweicht, eine geringere Genauigkeit bei der Klassifikation erwartet.

'batch_size': Anzahl der Bilder (und ihren entsprechenden Labels) in einem Batch, die in den Gerätespeicher transferiert werden. Die Menge der Bilder, die beim Training in einer Iteration miteinander verarbeitet werden, beinhaltet eine Anzahl *'batch_size'* mal *'batch_size_multiplier'* Bilder. Mehr Informationen können in der Dokumentation von `train_dl_classifier_batch` gefunden werden. Der Parameter *'batch_size'* ist im vortrainierten Modell gespeichert. Standardmäßig ist *'batch_size'* so gesetzt, dass ein Training des vortrainierten Klassifikators mit bis zu `100` Klassen problemlos auf einer GPU mit einem Speicher von `8` Gigabyte durchgeführt werden kann. Für die vortrainierten Klassifikatoren ergeben sich so folgende Standardwerte für die Batchgröße:

Vortrainierter Klassifikator	Standardwerte für *'batch_size'*
'pretrained_dl_classifier_compact.hdl'	*160*
'pretrained_dl_classifier_enhanced.hdl'	*96*
'pretrained_dl_classifier_resnet50.hdl'	*23*

Bei der Inferenz kann die *'batch_size'* im Allgemeinen unabhängig von der Anzahl der gleichzeitig zu inferierenden Bilder gesetzt werden. Siehe `apply_dl_classifier` für weitere Informationen, wie man diesen Parameter setzt, um eine größere Effizienz zu erzielen.

'batch_size_multiplier': Multiplikationsfaktor für *'batch_size'*, um das Training mit einer größeren Anzahl an Bildern in einem Schritt zu ermöglichen, was aufgrund limitierter GPU-Speichergröße sonst nicht möglich wäre. Auf die Evaluierung und Inferenz hat dieser Parameter keinen Einfluss. Für weitere Details, siehe `train_dl_classifier_batch`. Der *'batch_size_multiplier'* ist standardmäßig auf 1 gesetzt.

'classes': Tupel von Labels entsprechend der Objektklassen die erkannt werden sollen. Die Reihenfolge der Objektklassen bleibt nach dem Setzen unverändert.

'gpu': Identifikationsnummer der GPU, auf der die Trainings- und Inferenz-Operatoren (`train_dl_classifier_batch` und `apply_dl_classifier`) ausgeführt werden. Standardmäßig wird die erste verfügbare GPU verwendet. Mit dem `get_system` Parameter *'cuda_devices'* kann die Liste der verfügbaren GPUs abgefragt werden. In *'gpu'* muss der Index der gewünschten GPU in dieser Liste übergeben werden.

'image_width': Breite der Bilder, die das Netzwerk verarbeitet. Der Defaultwert ist durch das Neuronale Netzwerk bestimmt, siehe `read_dl_classifier`.

'image_height': Höhe der Bilder, die das Netzwerk verarbeitet. Der Defaultwert ist durch das Neuronale Netzwerk bestimmt, siehe `read_dl_classifier`.

'image_num_channels': Anzahl Kanäle, die das Netzwerk verarbeitet. Möglich sind ein Kanal (Grauwertbild) oder drei Kanäle (dreikanaliges Bild). Der Defaultwert ist durch das Neuronale Netzwerk bestimmt, siehe `read_dl_classifier`. Ein Wechsel auf einen Kanal modifiziert die Netzwerk-Konfiguration. Dieser Prozess entfernt die Farbinformation, die in gewissen Layern enthalten ist, und ist nicht invertierbar.

'image_dimensions': Ein Tupel mit den Bilddimensionen Bildbreite *'image_width'*, Bildhöhe *'image_hight'* und Anzahl der Kanäle *'image_num_channels'*. Die Defaultwerte sind durch das Neuronale Netzwerk gegeben, siehe `read_dl_classifier`. Bezüglich der Anzahl Kanäle sind ein Kanal (Grauwertbild) oder drei Kanäle (dreikanaliges Bild) möglich. Ein Wechsel auf einen Kanal modifiziert die Netzwerk-Konfiguration. Dieser Prozess entfernt die Farbinformation, die in gewissen Layern enthalten ist, und ist nicht invertierbar.

'learning_rate': Anfangswert des Faktors, der den Einfluss des Gradienten während des Trainings bestimmt. Mehr Information können in der Dokumentation von `train_dl_classifier_batch` gefunden werden. Der Defaultwert ist vom Klassifikator abhängig.

'momentum': Werden die Argumente der Zielfunktionswerte aktualisiert, gibt der Hyperparameter *'momentum'* an, zu welchem Ausmaß vorherige Aktualisierungsvektoren im aktuellen Aktualisierungsvektor berücksichtigt werden. Für weitere Informationen verweisen wir auf die Dokumentation von `train_dl_classifier_batch`. Standardmäßig ist *'momentum'* auf *0.9* gesetzt.

'runtime': Definiert die Einheit, auf der die Operatoren ausgeführt werden. Standardmäßig ist *'runtime'* auf *'gpu'* gesetzt.

'cpu': Der Operator `apply_dl_classifier` wird auf der CPU ausgeführt, der Operator `train_dl_classifier_batch` kann hingegen nicht ausgeführt werden.
Falls zuvor auf der GPU gerechnet wurde, wird Speicher auf der CPU initialisiert und falls notwendig werden auf der GPU gespeicherte Werte auf die CPU verschoben.
Auf Intel oder AMD Architekturen verwendet die *'cpu'* Laufzeit zur Parallelisierung des Operators `apply_dl_classifier` OpenMP, wobei standardmäßig alle verfügbaren Threads der OpenMP

Laufzeitumgebung verwendet werden. In diesem Fall kann mit dem threadspezifischen Parameter *'tsp_thread_num'* des Operators `set_system` die zu verwendende Anzahl von Threads spezifiziert werden.
Auf Arm Architekturen verwendet die *'cpu'* Laufzeit zur Parallelisierung einen globalen thread pool. In diesem Fall kann mit dem globalen Parameter *'thread_num'* des Operators `set_system` die zu verwendende Anzahl von Threads spezifiziert werden. Auf Arm Architekturen kann keine Thread-spezifische Anzahl von Threads eingestellt werden.

'gpu': Der Speicher wird auf der GPU initialisiert und das entsprechende Handle angelegt. Die Operatoren `apply_dl_classifier` und `train_dl_classifier_batch` werden auf der GPU ausgeführt. Mehr Informationen zu diesen speziellen Voraussetzungen befinden sich im HALCON „`Installation Guide`".

'runtime_init': Kann mit *'immediately'* dazu verwendet werden um den Speicher auf der GPU zu initialisieren und das entsprechende Handle anzulegen. Andernfalls passiert das sobald es notwendig ist, was zu signifikant längeren Laufzeiten beim ersten Aufruf von `apply_dl_classifier` oder `train_dl_classifier_batch` führen kann. Nachfolgende Aufrufe von `set_dl_classifier_param` mit *'gpu'* oder *'batch_size'* führen dazu, dass der Speicher auf der GPU neu initialisiert wird.

Dieser Parameter hat keinen Einfluss, falls *'runtime'* auf *'cpu'* gesetzt ist.

'weight_prior': Regularisierungsparameter $\alpha \geq 0.0$ für die l_2 Regularisierung der Zielfunktion. Diese Regularisierung ist hilfreich falls während des Klassifikatortrainings Überanpassung auftritt. Ist der Hyperparameter *'weight_prior'* auf einen Wert ungleich Null gesetzt, so wird der folgende Regularisierungsterm zur Zielfunktion addiert (siehe `train_dl_classifier_batch`):

$$E_\alpha(w) = \frac{\alpha}{2} \sum_{k=0}^{K-1} |w_k|^2$$

Der Index k läuft hierbei über alle Gewichte des Netzwerks, mit der Ausnahme von Bias-Termen. Diese werden nicht regularisiert. Der Regularisierungsterm $E_\alpha(w)$ schiebt die Gewichte gegen Null und bestraft damit insbesondere die größten Gewichte, was effektiv die Komplexität des Modells verringert. Einfach gesagt, begünstigt die Regularisierung einfachere Modelle, die weniger anfällig sind, Rauschen in den Daten zu lernen und dadurch generalisieren diese Modelle in der Regel besser. Für den Fall einer Überanpassung des Klassifikators empfehlen wir verschiedene Werte für den Parameter *'weight_prior'* zu testen. Die Wahl des Wertes ist ein Abwägen zwischen der Fähigkeit des Netzwerkes zu Generalisieren, einer Überanpassung oder einer Unteranpassung. Ist α zu klein, ist eine Überanpassung möglich, ist es zu groß kann es passieren, dass das Modell seine Fähigkeit verliert, sich an die Daten anzupassen, da dann alle Gewichte effektiv Null sind. Um einen idealen Wert für α zu finden, empfehlen wir eine Kreuzvalidierung durchzuführen. Das heißt, das Training wird für eine Reihe von Werten durchgeführt und man wählt den Wert, der den besten Validierungsfehler liefert. Üblich sind zum Testen die Werte für *'weight_prior'* auf einer logarithmischen Skala zwischen $1.0, ..., 0.0001$. Falls das Training sehr lange dauert, besteht die Möglichkeit für die Optimierung der Hyperparameter einen reduzierten Datensatz zu verwenden. Der Defaultwert ist vom Klassifikator abhängig.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Achtung

Zum erfolgreichen Setzen von *'gpu'* Parametern werden cuDNN und cuBLAS benötigt, d.h. sowohl zum Setzen des Parameters `GenParamName` *'runtime'* auf *'gpu'* als auch des Parameters `GenParamName` *'gpu'*. Für weitere Details wird auf den „`Installation Guide`", Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods", verwiesen.

Parameter

▷ **`DLClassifierHandle`** (input_control) ..... dl_classifier ⇝ *handle*
Handle des Deep Learning-basierten Klassifikators.

▷ **`GenParamName`** (input_control) ..... attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'classes'
Werteliste: `GenParamName` ∈ {'batch_size', 'batch_size_multiplier', 'classes', 'gpu', 'image_dimensions',

'image_num_channels', 'image_height', 'image_width', 'learning_rate', 'momentum', 'runtime', 'runtime_init', 'weight_prior'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / real / integer
Wert des generischen Parameters.
Defaultwert: ['class_1','class_2','class_3']
Wertevorschläge: GenParamValue ∈ {1, 2, 3, 50, 0.001, 'cpu', 'gpu', 'immediately'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_dl_classifier_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_dl_classifier

Nachfolger

get_dl_classifier_param, apply_dl_classifier, train_dl_classifier_batch

Alternativen

set_dl_model_param

Siehe auch

get_dl_classifier_param

Modul

Deep Learning Inference

train_dl_classifier_batch (BatchImages : : DLClassifierHandle, BatchLabels : DLClassifierTrainResultHandle)

Ausführen eines Trainingsschrittes eines Deep Learning-basierten Klassifikators auf Basis eines Batches von Bildern.

train_dl_classifier_batch ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator train_dl_model_batch verwenden.

train_dl_classifier_batch führt einen Trainingsschritt für den Deep Learning-basierten Klassifikator in DLClassifierHandle aus, welcher zuvor mit read_dl_classifier eingelesen wurde. Bevor der Trainingsschritt ausgeführt werden kann, müssen die Klassen mittels set_dl_classifier_param gesetzt werden. Zudem empfiehlt es sich, für die Optimierung wichtige Hyperparameter wie die Lernrate und das Momentum mit set_dl_classifier_param zu setzen. Der Trainingsschritt für das Neuronale Netzwerk basiert auf den Daten eines einzelnen Batches aus dem Trainings-Datensatz, d.h. den Bilder BatchImages und zugehörigen Labels BatchLabels. Dabei muss die Anzahl der Bilder des Batches einem Vielfachen der *'batch_size'* entsprechen, wobei der Parameter *'batch_size'* durch die Größe des vorhandenen GPU-Speichers limitiert ist. Um mehr Bilder in einem Trainingsschritt verarbeiten zu können, kann der Parameter *'batch_size_multiplier'* auf einen Wert größer 1 gesetzt werden. Die Anzahl an Bildern, die dem Trainings-Operator übergeben werden, beträgt *'batch_size'* mal *'batch_size_multiplier'*. Dabei sei angemerkt, dass ein Trainingsschritt auf einem Batch mit einem *'batch_size_multiplier'* größer 1 eine Approximation eines Trainingsschrittes ist, der auf dem gleichen Batch, aber mit einem *'batch_size_multiplier'* von 1 und einer entsprechend größeren *'batch_size'* berechnet wurde. Beispielsweise kann man Unterschiede erwarten zwischen Werten, die mit einer *'batch_size'* von 4 und *'batch_size_multiplier'* von 2 berechnet wurden, und Werten, die mit einer *'batch_size'* von 8 und *'batch_size_multiplier'* von 1 berechnet wurden, obwohl in beiden Fällen mit der gleichen Anzahl an Bildern gearbeitet wurde. Allerdings liefert diese Approximation im Allgemeinen vergleichbar gute Ergebnisse, so dass sie verwendet werden kann, wenn man mit mehr Bildern trainieren möchte als der Speicher der GPU erlaubt. In seltenen Fällen kann es vorkommen, dass die Approximation mit einer *'batch_size'* von 1 und einem entsprechend großen *'batch_size_multiplier'* nicht die erwartete Performance zeigt, was beispielsweise bei der Verwendung des

vortrainierten Netzes *'pretrained_dl_classifier_resnet50.hdl'* auftreten kann. Das Problem kann in den meisten Fällen gelöst werden, indem die *'batch_size'* auf einen Wert größer 1 gesetzt wird.

Man beachte außerdem, dass die Bilder in `BatchImages` je nach gewähltem Netzwerk gewisse Voraussetzungen an beispielsweise der Bildgröße oder dem Grauwertbereich erfüllen muss. Für detailliertere Informationen siehe `read_dl_classifier` und `set_dl_classifier_param`. Die Labels in `BatchImages` können als Tupel von Klassennamen oder als Tupel von Indizes (von *0* startend) übergeben werden. Dabei geben die Indizes die jeweiligen Positionen der Labels in dem Tupel der Klassen an, welches zuvor mit `set_dl_classifier_param` über *'classes'* gesetzt wurde. Informationen über die Resultate des Trainingsschrittes wie der Wert der Zielfunktion werden in `DLClassifierTrainResultHandle` gespeichert und können mit `get_dl_classifier_train_result` abgefragt werden.

Man beachte, dass eine Epoche in der Regel aus mehreren Batches besteht und für ein erfolgreiches Training das Iterieren über mehrere Epochen notwendig ist. Daher muss der Operator `train_dl_classifier_batch` mehrere Male mit unterschiedlichen Batches ausgeführt werden. Für einer detaillierte Erklärung siehe Legacy / DL Klassifikation.

Während des Trainingsschrittes wird der Wert der Zielfunktion, der von dem Neuronalen Netzwerk auf Basis der Batch-Daten erreicht wird, mit einem nichtlinearen Optimierungsalgorithmus minimiert. Als Algorithmus wird ein stochastisches Gradientenabstiegsverfahren (SGD) verwendet. Dabei werden im Trainingsschritt $t+1$ die Gewichte w_t des vorhergehenden Trainingsschrittes t wie folgt auf w_{t+1} aktualisiert:

$$\begin{aligned} v^{t+1} &= \mu v^t - \lambda \nabla_w L\big(f(x, w^t)\big) \\ w^{t+1} &= w^t + v^{t+1}, \end{aligned}$$

Bei obiger Formel stehen λ für die Lernrate, μ für das Momentum und $f(x, w)$ für das Klassifikationsergebnis des Deep Learning-basierten Klassifikators, welches von den Netzwerk Gewichten w und dem Eingabe-Batch x abhängt. Die Variable v_t wird verwendet um die Wirkung des Momentums μ einfließen zu lassen. Als Zielfunktion wird der Multinomial Logistic Loss in Kombination mit einem quadratischen Regularisierungsterm $E_\alpha(w)$ verwendet:

$$L\big(f(x,w)\big) = -\frac{1}{N}\sum_{n=0}^{N-1} \langle y_n, \log\big(f(x_n, w)\big)\rangle + E_\alpha(w), \quad E_\alpha(w) = \frac{\alpha}{2}\sum_{k=0}^{K-1} |w_k|^2,$$

Dabei kodiert der Vektor y_n das Label des n-ten Bildes x_n des Batches x welches N-viele Bilder beinhaltet. Der Term $\log(f(x_n, w))$ ist als Vektor zu verstehen, der in jedem Eintrag den Logarithmus der Einträge von $f(x_n, w)$ enthält. Der Regularisierungsterm $E_\alpha(w)$ besteht aus der gewichteten l^2-Norm, die alle K Gewichte außer Bias-Terme einbezieht. Der Einfluss der Regularisierung kann über α gesteuert werden. In der obigen Formel steht α für den Hyperparameter *'weight_prior'* und kann mit `set_dl_classifier_param` gesetzt werden. Für einen besseren Einblick kann mit dem Operator `get_dl_classifier_train_result` der aktuelle Wert von sowohl der Gesamt-Zielfunktion als auch von einzelnen Termen abgefragt werden.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Achtung

Der Operator `train_dl_classifier_batch` basiert auf Algorithmen, die in manchen Fällen nicht deterministisch sind. Daher können Ergebnisse von mehreren Aufrufen von `train_dl_classifier_batch` trotz gleicher Eingabewerte leicht voneinander abweichen.

Um diesen Operator ausführen zu können, werden cuDNN und cuBLAS benötigt. Für weitere Details wird auf den „`Installation Guide`", Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods", verwiesen.

Parameter

- ▷ **`BatchImages`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* real
  Bilder zusammengefasst in einem Batch.
- ▷ **`DLClassifierHandle`** (input_control) ..... dl_classifier ⇝ *handle*
  Handle des Deep Learning-basierten Klassifikators.

▷ **BatchLabels** (input_control) . . . . . string-array ⇝ *string* / integer
Labels der Bilder in dem Batch
Defaultwert: []
Werteliste: BatchLabels ∈ {[]}

▷ **DLClassifierTrainResultHandle** (output_control) . . . . . dl_classifier_train_result ⇝ *handle*
Handle der Trainingsresultate des Deep Learning-basierten Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_dl_classifier_train_result den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_dl_classifier, set_dl_classifier_param, get_dl_classifier_param

Nachfolger

get_dl_classifier_train_result, apply_dl_classifier, clear_dl_classifier_train_result, clear_dl_classifier

Alternativen

train_dl_model_batch, train_class_mlp, train_class_svm

Siehe auch

apply_dl_classifier

Modul

Deep Learning Training

write_dl_classifier (: : DLClassifierHandle, FileName :)

Abspeichern eines Deep Learning-basierten Klassifikators in eine Datei.

write_dl_classifier ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den allgemeinen CNN-basierten Operator write_dl_model verwenden.

write_dl_classifier speichert den Deep Learning-basierten Klassifikator DLClassifierHandle in der Datei FileName ab.

write_dl_classifier wird typischerweise aufgerufen, nachdem der Klassifikator mit train_dl_classifier_batch trainiert wurde. Der Klassifikator kann mit Hilfe von read_dl_classifier gelesen werden.

Man beachte, dass der laufzeitrelevante Parameter *'gpu'* des Klassifikators nicht mit abgespeichert wird.

Die Standard-Dateiendung eines Deep Learning-basierten Klassifikators ist *'.hdl'*.

Das Konzept der Deep Learning-basierten Klassifikation ist in der Einleitung zu Kapitel Deep Learning / Klassifikation beschrieben. Der Ablauf mit diesem veralteten Operator ist im Kapitel Legacy / DL Klassifikation beschrieben.

Parameter

▷ **DLClassifierHandle** (input_control) . . . . . dl_classifier ⇝ *handle*
Handle des Deep Learning-basierten Klassifikators.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Dateiname.
Dateiendung: `.hdl`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_dl_classifier_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_dl_classifier`, `train_dl_classifier_batch`, `set_dl_classifier_param`

Alternativen

`write_dl_model`

Modul

Deep Learning Inference

16.3 Develop

`dev_map_par` `( : : : )`

Öffnen des Dialogs zum Einstellen der Visualisierungsparameter.

`dev_map_par` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen `dev_open_tool` mit dem `ToolType` `'visualization_parameters_dialog'` verwenden.

`dev_map_par` öffnet den Dialog zur Einstellung der `Visualisierungsparameter`. Der Dialog lässt sich auch interaktiv über das Menü `Visualisierung` ▷ `Parameter setzen...` oder den entsprechenden Knopf der Werkzeugleiste öffnen. Im Dialog lassen sich die Visualisierungsparameter einstellen, die zur Darstellung von ikonischen Objekten benutzt werden.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Beispiel

```
read_image (Image, 'fabrik')
threshold (Image, Region, 128, 255)
dev_map_par ()
```

Ergebnis

`dev_map_par` liefert immer TRUE zurück.

Nachfolger

`dev_unmap_par`

Modul

Foundation

dev_map_prog (: : :)

Sichtbarmachen des Hauptfensters von HDevelop.

dev_map_prog ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen dev_open_tool mit dem ToolType 'program_window' verwenden.

dev_map_prog zeigt das Hauptfenster von HDevelop wieder an, nachdem es mit dev_unmap_prog unsichtbar gemacht wurde.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

In Abhängigkeit vom Betriebssystem und dem Fenstermanager kann es vorkommen, dass nach der Ausführung von dev_map_prog nur ein Icon des Programmfensters sichtbar ist. In diesem Fall muss das Programmfenster mit Hilfe der Maus richtig geöffnet werden.

Ergebnis

dev_map_prog liefert immer TRUE zurück.

Vorgänger

dev_unmap_prog

Nachfolger

dev_unmap_prog

Siehe auch

dev_map_par, dev_map_var

Modul

Foundation

dev_map_var (: : :)

Öffnen des Variablenfensters von HDevelop.

dev_map_var ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen dev_open_tool mit dem ToolType 'variable_window' verwenden.

dev_map_var zeigt das Variablenfenster wieder an, nachdem es mit dev_unmap_var unsichtbar gemacht wurde.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Ergebnis

dev_map_var liefert immer TRUE zurück.

Vorgänger

dev_unmap_var

Nachfolger

dev_unmap_var

Siehe auch

dev_map_par, dev_map_prog

Modul

Foundation

dev_unmap_par (: : :)

Schließt den Dialog zum Einstellen der Visualisierungsparameter.

dev_unmap_par ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen dev_close_tool mit dem ToolType 'visualization_parameters_dialog' verwenden.

dev_unmap_par schließt den Dialog zur Einstellung der Visualisierungsparameter. Der Dialog lässt sich auch interaktiv durch Drücken des Schließen-Knopfes in seiner Titelzeile schließen. Über dev_map_par lässt er sich wieder öffnen.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Ergebnis

dev_unmap_par liefert immer TRUE zurück.

Nachfolger

dev_map_par

Siehe auch

dev_map_par, dev_map_prog, dev_map_var

Modul

Foundation

dev_unmap_prog (: : :)

Verstecken des Hauptfensters.

dev_unmap_prog ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen dev_close_tool mit dem ToolType 'program_window' verwenden.

dev_unmap_prog versteckt im MDI-Modus das Hauptfenster und im SDI-Modus das Programmfenster. Es kann durch Aufruf von dev_map_prog wieder angezeigt werden.

Achtung: Wenn das Hauptfenster versteckt wurde, ist es nicht mehr möglich, den Operator dev_map_prog einzugeben und auszuführen. Es ist daher darauf zu achten, dev_unmap_prog nur im laufenden Programm auszuführen, wenn ein nachfolgender Aufruf von dev_map_prog erreicht werden kann.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Ergebnis

dev_unmap_prog liefert immer TRUE zurück.

Nachfolger

dev_map_prog, stop

Siehe auch

dev_map_par, dev_map_prog, dev_map_var

Modul

Foundation

dev_unmap_var (: : :)

Verstecken des Variablenfensters.

dev_unmap_var ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen dev_close_tool mit dem ToolType 'variable_window' verwenden.

dev_unmap_var versteckt das Variablenfenster. Das Fenster lässt sich auch interaktiv durch Drücken des Schließen-Knopfes in seiner Titelzeile schließen. Über dev_map_var oder interaktiv über das Fenster-Menü lässt es sich wieder öffnen.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Ergebnis

`dev_unmap_var` liefert immer TRUE zurück.

Nachfolger

`dev_map_var`

Siehe auch

`dev_map_par`, `dev_map_prog`

Modul

Foundation

16.4 Filter

`gauss_image` (`Image` : `ImageGauss` : `Size` :)

Glättung mit diskreten Gaußfunktionen.

`gauss_image` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen zum Glätten von Bildern sollten stattdessen den Operator `gauss_filter` verwenden.

`gauss_image` glättet Bilder mittels der diskreten Gaußfunktion. Die glättende Wirkung erhöht sich dabei mit zunehmender Filtergröße. Es werden folgende Filtergrößen (`Size`) unterstützt (in Klammer steht der sigma-Wert der Gaußfunktion):

3	(0.65)
5	(0.87)
7	(1.43)
9	(1.88)
11	(2.31)

Zur Randbehandlung werden die Grauwerte der Bilder an den Bildrändern gespiegelt.

Als Alternative zu `gauss_image` bietet sich `binomial_filter` an. `binomial_filter` ist deutlich effizienter als `gauss_image`. Es ist zu beachten, dass die Maskengröße in `binomial_filter` nicht dieselbe Glättung wie bei `gauss_image` bewirkt. Korrespondierende Maskengrößen lassen sich über die jeweiligen sigma-Werte der Gaußfunktion bestimmen.

`gauss_image` kann auf OpenCL-Geräten ausgeführt werden. Allerdings kann das Ergebnis von dem der skalaren Implementierung geringfügig abweichen.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Damit `gauss_image` auf einem OpenCL-Gerät ausgeführt werden kann, muss `Image` eine Breite und Höhe von mindestens 64 Pixeln aufweisen.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Zu glättendes Bild.
- ▷ **`ImageGauss`** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Gefiltertes Bild.
- ▷ **`Size`** (input_control) . . . . . integer ⇝ *integer*
  Gewünschte Filtergröße.
  Defaultwert: 5
  Werteliste: `Size` ∈ {3, 5, 7, 9, 11}

Beispiel

```
gauss_image(Input,Gauss,7)
regiongrowing(Gauss,Segments,7,7,5,100)
```

Komplexität

Pro Bildpunkt: $O(Size * 2)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gauss_image` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Modul

Foundation

polar_trans_image (ImageXY : ImagePolar : Row, Column, Width, Height :)

Anwenden einer Polartransformation

`polar_trans_image` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den Operator `polar_trans_image_ext` verwenden.

`polar_trans_image` wandelt ein Bild mit (x,y)-Koordinaten in ein Bild mit (Winkel,Radius)-Koordinaten um. Die Größe des Zielbildes wird mit `Width` und `Height` angegeben. `Width` gibt dabei die Auflösung des Winkels und `Height` die Auflösung des Radius an. `Row` und `Column` geben die Position im Eingabebild an, in der das Zentrum des Polarkoordinatensystems liegt. Dieser Punkt wird auf die erste (oberste) Zeile im Ergebnisbild abgebildet.

Ein Punkt (x',y') im Ergebnisbild ist der Punkt (x,y) im Eingabebild wie folgt zugeordnet:

$$\begin{aligned} x &= y' \cos(2\pi(x'/resultwidth)) + \texttt{Column} \\ y &= y' \sin(2\pi(x'/resultwidth)) + \texttt{Row}. \end{aligned}$$

`polar_trans_image` kann auf OpenCL-Geräten ausgeführt werden, wenn das Eingangsbild die maximale Größe für Bildobjekte des ausgewählten Gerätes nicht überschreitet. Aus numerischen Gründen kann es dabei zu geringfügigen Abweichungen des Ergebnisses gegenüber der Ausführung auf der CPU kommen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

▷ **`ImageXY`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Eingabebild in x/y-Koordinaten.

▷ **`ImagePolar`** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / real
Ausgabebild in Polar-Koordinaten.

▷ **`Row`** (input_control) . . . . . point.y ⇝ *integer*
Zeilenindex des Zentrums.
Defaultwert: 100
Wertevorschläge: `Row` ∈ {0, 10, 100, 200}
Typischer Wertebereich: 0 ≤ `Row` ≤ 512
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Zentrums.
Defaultwert: 100
Wertevorschläge: Column ∈ {0, 10, 100, 200}
Typischer Wertebereich: 0 ≤ Column ≤ 512
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Zielbildes.
Defaultwert: 314
Wertevorschläge: Width ∈ {100, 200, 157, 314, 512}
Typischer Wertebereich: 2 ≤ Width ≤ 512
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Zielbildes.
Defaultwert: 200
Wertevorschläge: Height ∈ {100, 128, 256, 512}
Typischer Wertebereich: 2 ≤ Height ≤ 512
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

Beispiel

```
read_image(Image,'monkey')
dev_display (Image)
polar_trans_image(Image,PolarImage,100,100,314,200)
dev_display (PolarImage)
```

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.

Modul

Foundation

16.5 Graphik

```
clear_rectangle ( : : WindowHandle, Row1, Column1, Row2,
    Column2 : )
```

Löschen eines Rechtecks auf dem Ausgabefenster.

clear_rectangle ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

clear_rectangle löscht alle Einträge in dem Rechteck, das durch die linke obere Ecke (Row1,Column1) und die rechte untere Ecke (Row2,Column2) aufgespannt wird. Löschen bedeutet, dass das angegebene Rechteck auf die Hintergrundfarbe gesetzt wird (siehe open_window).

Falls mehr als ein Rechteck gelöscht werden soll, können auch mehrere Rechtecke übergeben werden, d.h. die Parameter Row1, Column1, Row2 und Column2 sind jeweils Tupel.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Row1** (input_control) . . . . . rectangle.origin.y(-array) ⇝ *integer*
Zeilenindex der linken oberen Ecke.
Defaultwert: 10
Typischer Wertebereich: 0 ≤ `Row1` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Column1** (input_control) . . . . . rectangle.origin.x(-array) ⇝ *integer*
Spaltenindex der linken oberen Ecke.
Defaultwert: 10
Typischer Wertebereich: 0 ≤ `Column1` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Row2** (input_control) . . . . . rectangle.corner.y(-array) ⇝ *integer*
Zeilenindex der rechten unteren Ecke.
Defaultwert: 118
Typischer Wertebereich: 0 ≤ `Row2` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Row2 > Row1

▷ **Column2** (input_control) . . . . . rectangle.corner.x(-array) ⇝ *integer*
Spaltenindex der rechten unteren Ecke.
Defaultwert: 118
Typischer Wertebereich: 0 ≤ `Column2` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Column2 >= Column1

Beispiel

```
* Erase a rectangle in the output window interactively:
draw_rectangle1(WindowHandle,L1,C1,L2,C2)
```

Ergebnis

Ist ein Ausgabefenster vorhanden und sind die angegebenen Parameter korrekt, dann liefert `clear_rectangle` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`, `set_rgb`, `set_hsi`, `draw_rectangle1`

Alternativen

`clear_window`, `disp_rectangle1`

Siehe auch

`open_window`

Modul

Foundation

disp_distribution (: : WindowHandle, Distribution, Row, Column, Scale :)

Ausgeben einer Rauschverteilung.

disp_distribution ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

disp_distribution gibt eine Rauschverteilung im Fenster aus. Die Parameter Row und Column geben die Zeile bzw. Spalte des Mittelpunktes der Grafik (für die Positionierung) an. Der Skalierungsfaktor Scale legt die Größe fest. Dabei bedeutet ein Faktor von *1*, dass 256 Werte dargestellt werden; bei *2* werden 128 Werte dargestellt, bei *3* werden 64 Werte dargestellt, usw. Rauschverteilung können mit Operationen wie gauss_distribution oder noise_distribution_mean erzeugt werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **Distribution** (input_control) . . . . . real-array ⇝ *real*
  Grauwertverteilung (513 Werte).
- ▷ **Row** (input_control) . . . . . point.y ⇝ *integer*
  Zeilenindex des Mittelpunktes.
  Defaultwert: 256
  Wertevorschläge: Row ∈ {0, 64, 128, 256}
  Typischer Wertebereich: 0 ≤ Row ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
- ▷ **Column** (input_control) . . . . . point.x ⇝ *integer*
  Spaltenindex des Mittelpunktes.
  Defaultwert: 256
  Wertevorschläge: Column ∈ {0, 64, 128, 256}
  Typischer Wertebereich: 0 ≤ Column ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
- ▷ **Scale** (input_control) . . . . . integer ⇝ *integer*
  Größe der Darstellung.
  Defaultwert: 1
  Wertevorschläge: Scale ∈ {1, 2, 3, 4, 5, 6}

Beispiel

```
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
set_draw(WindowHandle,'fill')
set_color(WindowHandle,'white')
read_image(Image,'monkey')
draw_region(Region,WindowHandle)
noise_distribution_mean(Region,Image,21,Distribution)
disp_distribution (WindowHandle,Distribution,100,100,3)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi, noise_distribution_mean, gauss_distribution

Siehe auch

gen_region_histo, gauss_distribution, noise_distribution_mean

Modul

Foundation

disp_lut (: : WindowHandle, Row, Column, Scale :)

Grafisches Darstellen der Farbtabelle.

disp_lut ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

disp_lut gibt die Farbtabelle („look-up-table“, kurz: „lut“) auf dem Fenster grafisch aus. Eine Farbtabelle ist die Umsetzung der Grauwerte eines Bildes in Farben, bzw. Grauwerten auf dem Bildschirm. Diese Umsetzung kann bei den meisten Ausgabegeräten mit set_lut verändert werden. disp_lut erzeugt eine grafische Darstellung dieser drei Umsetzungstabellen (für Rot, Grün und Blau). Die Ausgabe erfolgt im Fenster mit der logischen Fensternummer WindowHandle und gibt für dieses die Farbtabelle aus. Die Parameter Row und Column geben die Zeile bzw. Spalte des Mittelpunktes der Grafik (für die Positionierung) an. Der Skalierungsfaktor Scale legt die Größe fest. Dabei bedeutet *1*, dass 256 Werte dargestellt werden; bei *2* werden 128 Werte dargestellt, bei *3* werden 64 Werte dargestellt, usw. Bei Farbtabellen für Schwarz/Weiß - Darstellungen erfolgt die Ausgabe der Grafik in der aktuellen Farbe (siehe: set_color, set_rgb, etc.). Bei Falschfarbtabellen werden die Farben rot, grün und blau für die Darstellung der jeweiligen Farbkomponenten verwendet.

Achtung

disp_lut ist nur für Rechner geeignet, die mit Farbtabellen arbeiten.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fensteridentifikator.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeilenindex des Schwerpunktes der Grafik.
  Defaultwert: 128
  Typischer Wertebereich: $0 \leq \texttt{Row} \leq 511$
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenindex des Schwerpunktes der Grafik.
  Defaultwert: 128
  Typischer Wertebereich: $0 \leq \texttt{Column} \leq 511$
- ▷ **Scale** (input_control) ..... integer ⇝ *integer*
  Verkleinerungsfaktor.
  Defaultwert: 1
  Werteliste: $\texttt{Scale} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
  Typischer Wertebereich: $0 \leq \texttt{Scale} \leq 20$

Ergebnis

Ist Ausgabefenster gültig und verfügt der Rechner über Farbtabellen, dann liefert disp_lut den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_lut

Siehe auch

open_window, set_lut, set_fix, set_pixel, write_lut, get_lut, set_color

Modul

Foundation

get_comprise (: : WindowHandle : Mode)

Abfragen der Behandlung der Bildmatrix bei der Ausgabe.

get_comprise ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

get_comprise gibt den von den Operatoren disp_image und disp_color verwendeten Modus des Ausgabefensters WindowHandle für die Behandlung von Grauwerten an. Es wird hierdurch festgelegt, ob nur die Grauwerte der Objekte oder das ganze Bild verwendet wird. Diese Abfrage wird verwendet, wenn der aktuelle Modus kurzzeitig überschrieben werden soll. Hierzu wird zunächst der aktuelle Wert abgefragt, dann überschrieben und schließlich auf den alten Wert zurückgesetzt. Das Zurücksetzen erfolgt mit set_comprise.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Mode** (output_control) ..... string ⇝ *string*
  Aktueller Ausgabemodus für Bilder.

Ergebnis

get_comprise liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_comprise, disp_image, disp_image

Siehe auch

set_comprise, disp_image, disp_color

Modul

Foundation

get_fix (: : WindowHandle : Mode)

Abfragen des aktuellen Fixiermodus.

get_fix ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

get_fix liefert den durch set_fix gesetzten Fixierungsmodus für die aktuelle Farbtabelle des gültigen Fensters .

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fensteridentifikator.
- ▷ **Mode** (output_control) ..... string ⇝ *string*
  Aktueller Fixiermodus.

Ergebnis

Wenn das Fenster gültig ist liefert get_fix TRUE. Ansonsten wird eine Fehlerbehandlung ausgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_fix, set_pixel, set_rgb

Siehe auch

set_fix

Modul

Foundation

get_fixed_lut (: : WindowHandle : Mode)

Abfragen der Fixierung der Farbtabelle bei Echtfarbenbildern

get_fixed_lut ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

Abfrage der Fixierung der Farbtabelle bei Echtfarbenbildern

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fensteridentifikator.
- ▷ **Mode** (output_control) ..... string ⇝ *string*
  Modus für Fixierung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_fixed_lut

Modul

Foundation

get_insert (: : WindowHandle : Mode)

Abfragen der aktuellen Darstellungsfunktion.

get_insert ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

get_insert gibt den Zeichenmodus des Ausgabefensters aus, der von Operatoren wie disp_region, disp_line, disp_rectangle1, etc. verwendet wird. Gesetzt wird der Parameter mit dem Operator set_insert. Mögliche Werte für Mode können mit Hilfe des Operators query_insert abgefragt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Mode** (output_control) ..... string ⇝ *string*
  Zeichenmodus.

Ergebnis

get_insert liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

query_insert

Nachfolger

set_insert, disp_image

Siehe auch

set_insert, query_insert, disp_region, disp_line

Modul

Foundation

get_line_approx (: : WindowHandle : Approximation)

Abfragen des aktuellen Approximationsfehlers für Konturen.

get_line_approx ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

get_line_approx gibt einen Parameter für die Darstellungsart für Konturen der Regionenausgabe des Ausgabefensters an, die von dem Operator disp_region verwendet wird. Konkret steuert dieser Parameter die Polygonapproximation für Konturen (0 ⇔ keine Approximation). Gesetzt wird der Parameter mit dem Operator set_line_approx. Der Parameter ist nur von Bedeutung, wenn die Ränder von Objekten dargestellt werden. Insbesondere wenn mit set_line_style ein Muster gesetzt wurde.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Approximation** (output_control) ..... string ⇝ *integer*
Aktueller Approximationsfehler für Konturdarstellung.

Ergebnis

get_line_approx liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_line_approx, set_line_style, disp_region

Siehe auch

get_region_polygon, set_line_approx, set_line_style, disp_region

Modul

Foundation

get_lut_style (: : WindowHandle : Hue, Saturation, Intensity)

Abfragen von Modifikationsparametern der Farbtabelle.

get_lut_style ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

get_lut_style liefert die Werte, die mit set_lut_style gesetzt wurden. Voreingestellt sind:

Hue: 0.0

Saturation 1.0

Intensity 1.0

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fensteridentifikator.
- ▷ **Hue** (output_control) . . . . . real ⇝ *real*
  Modifikation des Farbwertes.
- ▷ **Saturation** (output_control) . . . . . real ⇝ *real*
  Modifikation der Sättigung.
- ▷ **Intensity** (output_control) . . . . . real ⇝ *real*
  Modifikation der Intensität.

Ergebnis

`get_lut_style` liefert den Wert TRUE, falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`set_lut_style`, `set_lut`

Siehe auch

`set_lut_style`

Modul

Foundation

get_pixel (: : WindowHandle : Pixel)

Abfragen des aktuellen Farbtabellenindex.

get_pixel ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`get_pixel` gibt die Kodierung des Ausgabegrauwerts bzw. der Ausgabefarbe für das Ausgabefenster aus. Wird mit den Operatoren `set_color` oder `set_gray` der Modus für die Ausgabe (`disp_region`, `disp_ellipse`, etc.) auf Farbe(n) bzw. Grauwert(e) gesetzt, dann wird der angegebene Werte in einen internen Code umgewandelt. Dieser Code wird dann (physikalisch) bei der Darstellung auf dem Bildschirm verwendet. Die Umsetzung ist von den Abbildungseigenschaften und dem Zustand des Ausgabegerätes abhängig und kann bei verschiedenen Programmläufen unterschiedlich ausfallen. Der Begriff „Pixel“ ist nicht mit dem des „Pixels“ in der Bildverarbeitung zu verwechseln (hierfür gibt es den Operator `get_grayval`). Ein Pixel ist hier i.a. der Index in der Farbtabelle.

`get_pixel` erlaubt es, die Ausgabeart zu sichern, ohne dass man wissen muss, ob Farbe(n) oder Grauwert(e) gesetzt wurden. Zurückgesetzt wird der Parameter mit dem Operator `set_pixel`.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **Pixel** (output_control) . . . . . integer-array ⇝ *integer*
  Index in der aktuellen Farbtabelle.

Ergebnis

`get_pixel` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`set_pixel`, `disp_region`, `disp_image`

Siehe auch

`set_pixel`

Modul

Foundation

get_tshape (: : WindowHandle : TextCursor)

Form des Textcursors abfragen.

`get_tshape` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`get_tshape` gibt für das Ausgabefenster die Art des momentan verwendeten Textcursors aus. Gesetzt wird ein neuer Textcursor mit dem Operator `set_tshape`.

Unter einem Textcursor (im Gegensatz zum Mauscursor) wird hier die Markierung der aktuellen Schreibposition verstanden (die allerdings auch unsichtbar sein kann). Welche Darstellungsarten für den Textcursor existieren, kann mit Hilfe des Operators `query_tshape` erfragt werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **`TextCursor`** (output_control) ..... string ⇝ *string*
  Name des aktuellen Textcursors.

Ergebnis

`get_tshape` liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `set_font`

Nachfolger

`set_tshape`, `set_tposition`, `write_string`, `read_string`, `read_char`

Siehe auch

`set_tshape`, `query_tshape`, `write_string`, `read_string`

Modul

Foundation

gnuplot_close (: : GnuplotFileID :)

Schließen von offenen gnuplot-Dateien oder Beendigung von laufenden gnuplot-Unterprozessen.

`gnuplot_close` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

gnuplot_close schließt alle offenen Dateien, die mit gnuplot_open_file geöffnet wurden, oder beendet den gnuplot-Unterprozess, der mit gnuplot_open_pipe gestartet wurde. In letzterem Fall werden auch alle temporären Dateien, die zur 3D-Darstellung von Bildern und von Steuerparametern benötigt werden, wieder gelöscht. Daher muss nach Beendigung einer Plot-Sequenz unbedingt gnuplot_close ausgeführt werden. GnuplotFileID ist der Identifikator des zugehörigen gnuplot-Ausgabestroms.

Parameter

▷ **GnuplotFileID** (input_control) . . . . . gnuplot ⇝ *handle*
Identifikator für den gnuplot-Ausgabestrom.

Ergebnis

gnuplot_close liefert TRUE, falls GnuplotFileID ein gültiger gnuplot-Ausgabestrom ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gnuplot_open_pipe, gnuplot_open_file, gnuplot_plot_image

Siehe auch

gnuplot_open_pipe, gnuplot_open_file, gnuplot_plot_image

Modul

Foundation

gnuplot_open_file (: : FileName : GnuplotFileID)

Öffnen einer gnuplot-Datei für Bilder und Steuerparameter.

gnuplot_open_file ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

gnuplot_open_file erlaubt die Ausgabe von Bildern und Steuerparametern in einem Format, das später von gnuplot verarbeitet werden kann. Der Parameter FileName gibt den Basisnamen für die mit gnuplot_plot_image zu erstellenden Plots an. gnuplot_open_file erzeugt daraus eine Steuerdatei mit dem Namen <FileName>.gp, in der die einzelnen Plot-Befehle abgelegt werden. Für jedes ausgegebene Bild (oder Steuerparameter) wird dann von gnuplot_plot_image eine Datei angelegt, die die eigentlichen Daten enthält, und den Namen <FileName>.dat.<Zahl> erhält, wobei Zahl die Nummer des Plots in der aktuellen Folge ist. Die erzeugte Steuerdatei kann später vom Benutzer an die eigenen Bedürfnisse angepasst werden. Nach Ausgabe des letzten Plots muss gnuplot_close ausgeführt werden, um die offenen Dateien zu schließen. Der zugehörige Identifikator für den gnuplot-Ausgabestrom wird in GnuplotFileID zurückgeliefert.

Parameter

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Basisname für die Steuer- und Ausgabedateien.
Dateiendung: .gp

▷ **GnuplotFileID** (output_control) . . . . . gnuplot ⇝ *handle*
Identifikator für den gnuplot-Ausgabestrom.

Ergebnis

gnuplot_open_file liefert den Wert TRUE, falls die Steuerdatei geöffnet werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`gnuplot_plot_image`, `gnuplot_close`

Alternativen

`gnuplot_open_pipe`

Siehe auch

`gnuplot_open_pipe`, `gnuplot_close`, `gnuplot_plot_image`

Modul

Foundation

gnuplot_open_pipe (: : : GnuplotFileID)

Öffnen einer Pipe zu einem gnuplot-Unterprozess zur Darstellung von Bildern und Steuerparametern.

gnuplot_open_pipe ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`gnuplot_open_pipe` öffnet eine Pipe zu einem gnuplot-Unterprozess mit dem nachfolgend durch `gnuplot_plot_image` Bilder als 3D-Plots und `gnuplot_plot_ctrl` Steuerparameter dargestellt werden können. Der Unterprozess muss nach Ausgabe des letzten Plots mit `gnuplot_close` beendet werden. Der zugehörige Identifikator für den gnuplot-Ausgabestrom wird in `GnuplotFileID` zurückgeliefert.

Achtung

`gnuplot_open_pipe` ist nur unter unixartigen Systemen implementiert, da sich gnuplot unter Windows (`wgnuplot`) nicht durch einen externen Prozess steuern lässt.

Parameter

▷ **GnuplotFileID** (output_control) ..... gnuplot ⇝ *handle*
Identifikator für den gnuplot-Ausgabestrom.

Ergebnis

`gnuplot_open_pipe` liefert den Wert TRUE, falls der gnuplot-Unterprozess kreiert werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`gnuplot_plot_image`, `gnuplot_plot_ctrl`, `gnuplot_close`

Alternativen

`gnuplot_open_file`

Modul

Foundation

gnuplot_plot_ctrl (: : GnuplotFileID, Values :)

Ausgabe von Tupeln durch gnuplot.

gnuplot_plot_ctrl ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

gnuplot_plot_ctrl gibt ein Tupel unter Verwendung von gnuplot als Plot aus. Falls vorher ein gnuplot-Unterprozess mittels gnuplot_open_pipe gestartet wurde, erfolgt die Darstellung in einem gnuplot-Fenster. Ansonsten erfolgt die Ausgabe in eine Datei, die später mit gnuplot verarbeitet werden kann. In beiden Fällen wird der gnuplot-Ausgabestrom durch GnuplotFileID bestimmt.

Parameter

- ▷ **GnuplotFileID** (input_control) ..... gnuplot ⇝ *handle*
  Identifikator für den gnuplot-Ausgabestrom.
- ▷ **Values** (input_control) ..... number-array ⇝ *real* / integer
  Auszugebende Werte (y-Werte).

Ergebnis

gnuplot_plot_ctrl liefert den Wert TRUE, falls nur ganze Zahlen oder Gleitpunktzahlen übergeben wurden, GnuplotFileID ein gültiger gnuplot-Ausgabestrom ist und die Datei für den aktuellen plot erzeugt werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gnuplot_open_pipe, gnuplot_open_file

Nachfolger

gnuplot_close

Siehe auch

gnuplot_open_pipe, gnuplot_open_file, gnuplot_close

Modul

Foundation

gnuplot_plot_funct_1d (: : GnuplotFileID, Function :)

Ausgabe einer Funktion durch gnuplot.

gnuplot_plot_funct_1d ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

gnuplot_plot_ctrl gibt eine Funktion unter Verwendung von gnuplot als Plot aus. Falls vorher ein gnuplot-Unterprozess mittels gnuplot_open_pipe gestartet wurde, erfolgt die Darstellung in einem gnuplot-Fenster. Ansonsten erfolgt die Ausgabe in eine Datei, die später mit gnuplot verarbeitet werden kann. In beiden Fällen wird der gnuplot-Ausgabestrom durch GnuplotFileID bestimmt.

Parameter

- ▷ **GnuplotFileID** (input_control) ..... gnuplot ⇝ *handle*
  Identifikator für den gnuplot-Ausgabestrom.
- ▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
  Auszugebende Funktion.

Ergebnis

gnuplot_plot_funct_1d liefert den Wert TRUE, falls GnuplotFileID ein gültiger gnuplot-Ausgabestrom ist und die Datei für den aktuellen plot erzeugt werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gnuplot_open_pipe, gnuplot_open_file

Nachfolger

gnuplot_close

Alternativen

gnuplot_plot_ctrl

Siehe auch

gnuplot_open_pipe, gnuplot_open_file, gnuplot_close

Modul

Foundation

gnuplot_plot_image (Image : : GnuplotFileID, SamplesX, SamplesY, ViewRotX, ViewRotZ, Hidden3D :)

Ausgabe von Bildern durch gnuplot.

gnuplot_plot_image ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

gnuplot_plot_image gibt Bilder unter Verwendung von gnuplot als 3D-Plots aus. Falls vorher ein gnuplot-Unterprozess mittels gnuplot_open_pipe gestartet wurde, erfolgt die Darstellung in einem gnuplot-Fenster. Ansonsten erfolgt die Ausgabe in eine Datei, die später mit gnuplot verarbeitet werden kann. In beiden Fällen wird der gnuplot-Ausgabestrom durch GnuplotFileID bestimmt. Die Parameter SamplesX und SamplesY geben an, wie viele Datenpunkte in x- bzw. y-Richtung an gnuplot ausgegeben werden sollen. Sie sind vollkommen analog zu den Variablen samples und isosamples in gnuplot. Mit den Parametern ViewRotX und ViewRotZ lässt sich die Position des Beobachters festlegen. ViewRotX gibt die Rotation des Koordinatensystems um die x-Achse an. Dies entspricht im wesentlichen der Neigung des Plots. ViewRotZ gibt die Rotation des Plots um die z-Achse an. Diese beiden Parameter entsprechen den ersten beiden Parametern des 'set view'-Befehls von gnuplot. Hidden3D gibt an, ob die verdeckten Oberflächenteile entfernt werden sollen oder nicht. Dies entspricht dem 'set hidden3d'-Befehl von gnuplot. Falls ein einziges Bild übergeben wird, erfolgt die Ausgabe in einen separaten Plot. Falls mehrere Bilder übergeben werden, werden alle in einem einzigen Plot dargestellt.

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / real
Auszugebendes Bild.

▷ **GnuplotFileID** (input_control) ..... gnuplot ⇝ *handle*
Identifikator für den gnuplot-Ausgabestrom.

▷ **SamplesX** (input_control) ..... integer ⇝ *integer*
Anzahl der Datenpunkte in x-Richtung.
Defaultwert: 64
Typischer Wertebereich: $2 \leq$ SamplesX ≤ 10000
Restriktion: SamplesX >= 2

▷ **SamplesY** (input_control) ..... integer ⇝ *integer*
Anzahl der Datenpunkte in y-Richtung.
Defaultwert: 64
Typischer Wertebereich: $2 \leq$ SamplesY ≤ 10000
Restriktion: SamplesY >= 2

▷ **ViewRotX** (input_control) . . . . . number ⇝ *real* / integer
Rotation des Plots um die x-Achse.
Defaultwert: 60
Typischer Wertebereich: 0 ≤ ViewRotX ≤ 180
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 10
Restriktion: 0 <= ViewRotX && ViewRotX <= 180

▷ **ViewRotZ** (input_control) . . . . . number ⇝ *real* / integer
Rotation des Plots um die z-Achse.
Defaultwert: 30
Typischer Wertebereich: 0 ≤ ViewRotZ ≤ 360
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 10
Restriktion: 0 <= ViewRotZ && ViewRotZ <= 360

▷ **Hidden3D** (input_control) . . . . . string ⇝ *string*
Ausgabe als hidden-surface-plot.
Defaultwert: 'hidden3d'
Werteliste: Hidden3D ∈ {'hidden3d', 'nohidden3d'}

Ergebnis

gnuplot_plot_image liefert den Wert TRUE, falls GnuplotFileID ein gültiger gnuplot-Ausgabestrom ist und die Datei für den aktuellen plot erzeugt werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gnuplot_open_pipe, gnuplot_open_file

Nachfolger

gnuplot_close

Siehe auch

gnuplot_open_pipe, gnuplot_open_file, gnuplot_close

Modul

Foundation

move_rectangle (: : WindowHandle, Row1, Column1, Row2, Column2, DestRow, DestColumn :)

Kopieren innerhalb eines Ausgabefensters.

move_rectangle ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

move_rectangle kopiert alle Einträge in dem Rechteck (Row1,Column1), (Row2,Column2) des Ausgabefensters an eine neue Position innerhalb desselben Fensters. Diese Position wird durch die linke obere Ecke (DestRow,DestColumn) festgelegt. Die durch das Bewegen des Rechtecks „bloßgelegten" Bereiche des Fensters werden auf die Farbe des Hintergrundes gesetzt.

Falls mehrere Rechtecke auf einmal bewegt werden sollen, können die Parameter jeweils in Form von Tupeln übergeben werden.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Row1** (input_control) . . . . . rectangle.origin.y(-array) ⇝ *integer*
Zeilenindex der linken oberen Ecke des Quell-Rechtecks.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ `Row1` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Column1** (input_control) . . . . . rectangle.origin.x(-array) ⇝ *integer*
Spaltenindex der linken oberen Ecke des Quell-Rechtecks.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ `Column1` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Row2** (input_control) . . . . . rectangle.corner.y(-array) ⇝ *integer*
Zeilenindex der rechten unteren Ecke des Quell-Rechtecks.
Defaultwert: 64
Typischer Wertebereich: 0 ≤ `Row2` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Column2** (input_control) . . . . . rectangle.corner.x(-array) ⇝ *integer*
Spaltenindex der rechten unteren Ecke des Quell-Rechtecks.
Defaultwert: 64
Typischer Wertebereich: 0 ≤ `Column2` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **DestRow** (input_control) . . . . . point.y(-array) ⇝ *integer*
Zeilenindex der linken oberen Ecke der Zielposition.
Defaultwert: 64
Typischer Wertebereich: 0 ≤ `DestRow` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **DestColumn** (input_control) . . . . . point.x(-array) ⇝ *integer*
Spaltenindex der linken oberen Ecke der Zielposition.
Defaultwert: 64
Typischer Wertebereich: 0 ≤ `DestColumn` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Beispiel

```
* "Interactive" copy of a rectangle in the same window
draw_rectangle1(WindowHandle,L1,C1,L2,C2)
get_mbutton(WindowHandle,LN,CN,Button)
move_rectangle(WindowHandle,L1,C1,L2,C2,LN,CN)
```

Ergebnis

Ist das Ausgabefenster gültig und sind die angegebenen Parameter korrekt, dann liefert `move_rectangle` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`

Alternativen

`copy_rectangle`

Siehe auch

open_window

Modul

Foundation

open_textwindow (: : Row, Column, Width, Height, BorderWidth, BorderColor, BackgroundColor, FatherWindow, Mode, Machine : WindowHandle)

Öffnen eines Textfensters.

open_textwindow ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den Operator open_window verwenden.

open_textwindow öffnet ein neues Textfenster, das zur Aus- und Eingabe von Text, sowie für die Ausgabe von Bildern verwendet werden kann. Unter Verwendung der logischen Fensternummer WindowHandle können alle Ausgaben (write_string, read_string, disp_region, etc.) auf dieses Fenster geleitet werden.

Textfenster besitzen neben dem Mauscursor auch einen Textcursor, der die aktuelle Schreibposition bezeichnet (genauer: die linke untere Ecke des auszugebenden Strings ohne Berücksichtigung der Unterlängen). Seine Position wird durch einen Unterstrich (oder eine andere Form) angezeigt (die Anzeige dieser Position kann aber auch ausgeschaltet sein (= Defaulteinstellung); vgl. set_tshape). Mit den Operatoren set_tposition und get_tposition kann die Position gesetzt bzw. abgefragt werden.

Beim Öffnen eines Textfensters wird die Cursorposition auf (H,0) gesetzt, wobei H die Höhe des Defaultfonts abzüglich der Unterlängen angibt. Der Cursor wird aber nicht angezeigt. Die Ausgabe beginnt also beim Schreiben in der linken oberen Ecke des Fensters.

Die Farben für den Hintergrund und den Bildrand können mit query_color abgefragt werden. Dies kann auch mit einem Fenster vom Typ 'invisible' durchgeführt werden. Das Clipping von Texten bei der Ausgabe (write_string) an den Fensterrändern kann mit set_check(::'~text':) eingeschaltet werden, d.h. es wird dann keine Fehlermeldung mehr erzeugt, falls der Text über den Rand des Fensters hinausragt.

Der Ursprung des Koordinatensystems des Fensters liegt in der linken oberen Ecke (Koordinaten: (0,0)). Nach unten nimmt der Zeilenindex zu (maximal: Height-1), nach rechts steigt der Spaltenindex (maximal: Width-1) an.

Der Parameter Machine gibt den Namen des Rechners an, auf dem das Fenster geöffnet werden soll. Im Falle von X-Window wird bei TCP-IP nur der Name, bei DEC-Net wird noch ein Doppelpunkt hinter den Namen gesetzt. Der „Server“ bzw. der „Screen“ wird nicht mit angegeben. Übergibt man den leeren String, so wird die Environmentvariable DISPLAY zur Festlegung des Zielrechners verwendet. Hierbei wird der Name in der üblichen Syntax

```
<Host>:0.0
```

angegeben.

Mit dem Parameter FatherWindow kann für die Fenstertypen *'WIN32-Window'*, *'X-Window'* und *'Cocoa'* das Vaterfenster für das zu öffnende Fenster angegeben werden. Ist die Kontrolle 'father' mittels set_check eingeschaltet, so muss FatherWindow ein HALCON-Fenster sein, andernfalls (set_check('~father')) kann es auch ein Betriebssystem-Fenster sein. Wird für FatherWindow der Wert 0 oder 'root' übergeben, so ist der Desktop (Windows) bzw. das Root-Fenster (unixartige Systeme) das Vaterfenster. In diesem Fall ist die Einstellung der Kontrolle 'father' mittels set_check nicht relevant. Der Aufrufer muss sicherstellen, dass FatherWindow ein gültiges Fensterhandle ist und dass dieses nicht zerstört wird, solange das eingebettete HALCON Fenster verwendet wird.

Die Position und Größe des Fensters kann sich im Laufe des Programms ändern. Dies kann zum einen mit set_window_extents erfolgen, aber auch durch externe Eingriffe (Window-Manager) hervorgerufen werden. Für den zweiten Fall ist der Operator set_window_extents vorgesehen.

Dem Fenster wird beim Öffnen ein sog. Defaultfont zugeordnet. Dieser wird für Operatoren wie write_string verwendet und kann mit set_font nach dem Aufruf von open_textwindow überschrieben werden. Es ist jedoch auch möglich, den Defaultfont mit dem Aufruf set_system(::'default_font',<Fontname>:) vor dem Öffnen eines Fensters (und auch aller weiteren) festzulegen (siehe hierzu auch query_font).

Die Farbe der Schrift (write_string, read_string) wird mit set_color, set_rgb, set_hsi oder set_gray festgelegt. Mit set_insert wird bestimmt, wie der Text (bzw. die Grafik) mit dem Inhalt des Bildwiederholspeichers verknüpft wird. Hierbei kann z.B. mit set_insert(::'not':) erreicht werden, dass nach zweimaligem Schreiben des Textes an der gleichen Position die Schrift wieder beseitigt wird.

Normalerweise wird jede Ausgabe (z.B. mit write_string, disp_region, disp_circle, etc.) in ein Fenster durch einen sog. „Flush" abgeschlossen. Dies bewirkt, dass die Daten nach Beendigung der Ausgabeprozedur auf dem Bildschirm vollständig sichtbar werden. Dies ist aber nicht in allen Fällen erforderlich, insbesondere dann, wenn ständig Ausgaben erfolgen, oder eine Maus-Prozedur aktiv ist. Hier ist es günstiger (d.h. schneller), wenn die Daten gepuffert werden, bis genügend Daten vorhanden sind. Dieses Verhalten kann durch den Aufruf von set_system(::'flush_graphic','false':) eingestellt werden.

Der Inhalt von Fenstern wird (falls die Treibersoftware dies unterstützt) gesichert; d.h. er bleibt erhalten, auch wenn das Fenster von anderen Fenstern überlappt wird. Dies ist jedoch nicht in allen Fällen notwendig: Wenn man ein Textfenster z.B. als Vaterfenster für andere Fenster verwendet, so kann für dieses der Sicherungsmechanismus abgestellt und damit der hierfür nötige Speicher eingespart werden. Dies geschieht mit dem Aufruf von set_system(::'backing_store','false':) vor dem Öffnen des Fensters.

Unterschied: Grafikfenster - Textfenster

- Im Gegensatz zu Grafikfenstern (open_window) können für Textfenster mehr Parameter (Farbe, Rand) beim Öffnen festgelegt werden.
- Nur Textfenster können zum Einlesen von Benutzerdaten verwendet werden (read_string).
- Die Ausgabe von Bildern, Regionen und Grafiken wird bei Textfenstern an den Rändern „geclippt", während bei Grafikfenstern „gezoomt" wird.
- Das Koordinatensystem (z.B. bei get_mbutton oder get_mposition) ist unabhängig von der Bildgröße immer in Bildschirmkoordinaten. Die maximalen Koordinaten entsprechen also der Größe des Fensters minus 1. Im Gegensatz dazu wird bei Grafikfenstern (open_window) immer ein Koordinatensystem verwendet, das dem Bildformat entspricht.

Der Parameter Mode legt den Modus des Fensters fest. Dieser kann folgende Werte annehmen:

'visible': Dies ist der normale Modus für Textfenster: Das Fenster wird entsprechend den Parametern erzeugt und alle Ein- und Ausgaben sind möglich.

'invisible': Unsichtbare Fenster werden nicht auf dem Bildschirm dargestellt. Die Parameter Row, Column, BorderWidth, BorderColor, BackgroundColor und FatherWindow sind ohne Bedeutung. Ausgaben auf diese Fenster haben keine Wirkung. Eingaben (read_string, Maus, etc.) sind nicht möglich. Diese Fenster werden verwendet, wenn man Darstellungsparameter für ein Ausgabegerät abfragen möchte, ohne ein (sichtbares) Fenster zu öffnen. Übliche Anfragen sind z.b. query_color und get_string_extents.

'transparent': Diese Fenster sind durchsichtig: Das Fenster selbst ist nicht sichtbar (Rand und Hintergrund), jedoch sind alle sonstigen Operationen möglich und alle Ausgaben werden dargestellt. Die Parameter BorderColor und BackgroundColor sind hier ohne Bedeutung. Eine übliche Verwendung für diesen Modus ist die Erzeugung von Maus-sensitiven Bereichen.

'buffer': Dies sind ebenfalls nicht sichtbare Fenster. Die Ausgabe von Bildern, Regionen und Grafik ist auf dem Bildschirm nicht sichtbar, wird jedoch im Speicher gepuffert. Die Parameter Row, Column, BorderWidth, BorderColor, BackgroundColor und FatherWindow sind ohne Bedeutung. Puffer-Fenster verwendet man beispielsweise, wenn man Ausgaben (im Hintergrund) vorbereitet und schließlich mit copy_rectangle in ein sichtbares Fenster kopiert. Eine andere Anwendung wäre die schnelle Verarbeitung von Bildauschnitten bei interaktiven Manipulationen. Texteingaben und Mausinteraktion ist mit dem Modus 'buffer' nicht möglich.

Achtung

Es ist zu beachten, dass die Parameter Row, Column, Width und Height durch das Ausgabegerät beschränkt sind. Wird ein Vaterfenster (FatherWindow <> 'root') angegeben, dann sind die Koordinaten relativ zum diesem Fenster.

Parameter

▷ **Row** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilenindex der linken oberen Ecke.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ Row (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Row >= 0

▷ **Column** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltenindex der linken oberen Ecke.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ Column (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Column >= 0

▷ **Width** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
Breite des Fensters.
Defaultwert: 256
Typischer Wertebereich: 0 ≤ Width (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Width > 0

▷ **Height** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
Höhe des Fensters.
Defaultwert: 256
Typischer Wertebereich: 0 ≤ Height (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Height > 0

▷ **BorderWidth** (input_control) . . . . . integer ⇝ *integer*
Breite des Fensterrandes.
Defaultwert: 2
Typischer Wertebereich: 0 ≤ BorderWidth (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: BorderWidth >= 0

▷ **BorderColor** (input_control) . . . . . string ⇝ *string*
Farbe des Fensterrandes.
Defaultwert: 'white'

▷ **BackgroundColor** (input_control) . . . . . string ⇝ *string*
Hintergrundfarbe.
Defaultwert: 'black'

▷ **FatherWindow** (input_control) . . . . . pointer ⇝ *integer* / string
Logische Nummer des Vaterfensters. Für den Bildschirm als Vater kann 'root' oder 0 eingegeben werden.
Defaultwert: 0
Restriktion: FatherWindow >= 0

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Fenstermodus.
Defaultwert: 'visible'
Werteliste: `Mode` ∈ {'visible', 'invisible', 'transparent', 'buffer'}

▷ **Machine** (input_control) . . . . . string ⇝ *string*
Name des Rechners, auf dem das Fenster geöffnet werden soll oder leerer String.
Defaultwert: ''

▷ **WindowHandle** (output_control) . . . . . window ⇝ *handle*
Fenster-Handle.

Beispiel

```
open_textwindow(0,0,900,600,1,'black','slate blue','root','visible', \
                '',WindowHandle1)
open_textwindow(10,10,300,580,3,'red','blue',WindowHandle1,'visible', \
                '',WindowHandle2)
open_window(10,320,570,580,WindowHandle1,'visible','',WindowHandle)
set_color(WindowHandle,'red')
read_image(Image,'monkey')
disp_image(Image,WindowHandle)
Button := 0
repeat
  try
      get_mposition(WindowHandle,Row,Column,Button)
      get_grayval(Image,Row,Column,Gray)
      write_string(WindowHandle2,['  Position (',Row,',',Column,')  '])
      write_string(WindowHandle2,['Gray value (',Gray,')  '])
      new_line(WindowHandle2)
  catch (Exception)
  endtry
until(Button == 4)
close_window(WindowHandle1)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `open_textwindow` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`reset_obj_db`

Nachfolger

`set_color`, `query_window_type`, `get_window_type`, `set_window_type`, `get_mposition`, `set_tposition`, `set_tshape`, `set_window_extents`, `get_window_extents`, `query_color`, `set_check`, `set_system`

Alternativen

`open_window`

Siehe auch

`write_string`, `read_string`, `new_line`, `get_string_extents`, `get_tposition`, `set_color`, `query_window_type`, `get_window_type`, `set_window_type`, `get_mposition`, `set_tposition`, `set_tshape`, `set_window_extents`, `get_window_extents`, `query_color`, `set_check`, `set_system`

Modul

Foundation

`query_insert` (: : WindowHandle : Mode)

Abfragen der Grafikmodi.

`query_insert` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

query_insert gibt an, in welcher Art die Pixelwerte in das Ausgabefenster eingetragen werden können. Eine Möglichkeit ist z.B., dass die neuen Pixel die vorhandenen Werte überschreiben. Im allgemeinen wird ein funktionaler Zusammenhang aus vorhandenen Werten und neuen Werten abgeleitet.

Mögliche Darstellungsfunktionen:

'copy': vorhandene Pixel überschreiben

'xor': trage ein: vorhandene Pixel „xor" neue Pixel

'complement': komplementiere bisher dargestellte Pixel

Immer vorhanden ist die Funktion 'copy'.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **Mode** (output_control) . . . . . string-array ⇝ *string*
  Name der Darstellungsprozedur.

Ergebnis

query_insert liefert den Wert TRUE, falls das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_insert, disp_region

Siehe auch

set_insert, get_insert

Modul

Foundation

query_tshape (: : WindowHandle : TextCursor)

Abfrage der verfügbaren Textcursor.

query_tshape ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

query_tshape gibt für das Ausgabefenster die Namen aller Darstellungsarten des Textcursors aus. Diese können von dem Operator set_tshape verwendet werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
  Fenster-Handle.
- ▷ **TextCursor** (output_control) . . . . . string-array ⇝ *string*
  Namen der verfügbaren Textcursor.

Ergebnis

query_tshape liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Nachfolger

set_tshape, write_string, read_string

Siehe auch

set_tshape, get_shape, set_tposition, write_string, read_string

Modul

Foundation

set_comprise (: : WindowHandle, Mode :)

Setzen der Behandlung der Bildmatrix bei der Ausgabe.

set_comprise ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

set_comprise gibt an, ob bei der Ausgabe von Grauwerten (disp_image) nur solche Grauwerte die zu dem auszugebenden Objekt gehören ('object') oder die gesamte Bildkomponente ('image') ausgegeben werden soll. Voreinstellung ist 'object'.

Achtung

Falls der Modus 'image' gewählt wurde, werden bei der Ausgabe undefinierte Grauwertbereiche dargestellt werden. Diese sind je nach Kontext schwarz oder mit beliebigen Speicherinhalten besetzt. Siehe hierzu das Beispiel.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus für die Ausgabe von Grauwerten.
  Defaultwert: 'object'
  Werteliste: Mode ∈ {'image', 'object'}

Beispiel

```
read_image(Image,'fabrik')
gen_circle (Circle, 200, 200, 100.5)
reduce_domain (Image, Circle, ImageReduced)
set_system('init_new_image','false')
sobel_amp(ImageReduced,SobelReduced,'sum_abs',3)
dev_display (SobelReduced)
get_comprise(WindowHandle,Mode)
set_comprise(WindowHandle,'image')
stop ()
dev_display (SobelReduced)
```

Ergebnis

set_comprise liefert den Wert TRUE, falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_comprise

Nachfolger

disp_image

Siehe auch

get_comprise, disp_image, disp_color

Modul

Foundation

set_fix (: : WindowHandle, Mode :)

Setzen der Farbtabellenfixierung.

set_fix ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

Verhalten bei Mode = *'true'*: set_fix fixiert das Pixel, das zuletzt durch eine der Funktionen set_pixel, set_gray, set_color, set_hsi oder set_rgb ermittelt wurde. (Bemerkung: Ein Pixel ist hier der Index in die aktuelle Farbtabelle). Fixierten Pixeln wird durch Setzen einer Farbe oder eines Grauwertes (set_color, set_rgb, set_hsi oder set_gray) die entsprechende neue Farbe zugewiesen (d.h überschrieben). Hierdurch ist es möglich an einer Stelle in der Farbtabelle jede Farbe (set_color), jeden Grauwert (set_gray) und jede Farbkombination (set_rgb,set_hsi) zu erzeugen.

Wenn Mode den Wert 'false' hat, wird die Fixierung zurückgenommen. Durch fortgesetzte Anwendung von set_pixel, set_fix(::WindowHandle,'true':), set_rgb und set_fix(::WindowHandle,'false':) kann eine Farbtabelle modifiziert oder erzeugt werden.

Achtung

Es ist zu beachten, dass bei der Verwendung von set_fix auch die Farben von „nicht HALCON Fenster“ verändert werden können.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fensteridentifikator.

▷ **Mode** (input_control) ..... string ⇝ *string*
Modus für Fixierung.
Defaultwert: 'true'
Werteliste: Mode ∈ {'true', 'false'}

Ergebnis

Wenn das Fenster gültig ist, die Hardware über eine Farbtabelle verfügt und der Parameter richtig besetzt ist, liefert set_fix TRUE. Ansonsten wird eine Fehlerbehandlung ausgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_fix

Nachfolger

set_pixel, set_rgb

Siehe auch

get_fix, set_pixel, set_rgb, set_color, set_hsi, set_gray

Modul

Foundation

set_fixed_lut (: : WindowHandle, Mode :)

Festhalten der Farbtabelle bei Echtfarbenbilder.

set_fixed_lut ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

Festhalten der Farbtabelle bei Echtfarbenbilder.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fensteridentifikator.

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Modus für Fixierung.
Defaultwert: 'true'
Werteliste: Mode ∈ {'true', 'false'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_fixed_lut

Modul

Foundation

set_insert (: : WindowHandle, Mode :)

Festlegen der Darstellungsfunktion in den Bildwiederholspeicher.

set_insert ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

set_insert gibt an, in welcher Art die Pixelwerte in das Ausgabefenster (Bildwiederholspeicher) eingetragen werden. Möglich ist etwa, dass die neuen Pixel die vorhandenen Werte überschreiben. Im allgemeinen wird ein funktionaler Zusammenhang aus vorhandenen Werten und neuen Werten abgeleitet. Der angegebene Wert wird dem Fenster zugeordnet, das zum Zeitpunkt des Aufrufs gültig ist. Die Ausgabe der Objekte erfolgt mit Operatoren wie disp_region, disp_polygon,disp_circle.

Mögliche Darstellungsfunktionen:

'copy': vorhandenen Wert mit Pixelwert überschreiben

'xor': trage ein: vorhandener Wert „xor" Pixelwert

'complement': komplementiere bisher dargestellten Wert

Abhängig von der jeweiligen Implementierung sind nicht alle Funktionen verfügbar (siehe query_insert). Immer vorhanden ist jedoch die Funktion 'copy'.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fenster-Handle.

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Name der Darstellungsfunktion.
Defaultwert: 'copy'
Werteliste: Mode ∈ {'copy', 'xor', 'complement'}

Ergebnis

set_insert liefert den Wert TRUE, falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

query_insert, get_insert

Nachfolger

disp_region

Siehe auch

get_insert, query_insert

Modul

Foundation

set_line_approx (: : WindowHandle, Approximation :)

Festlegen der Glättung für die Konturdarstellung.

set_line_approx ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

set_line_approx bestimmt den Approximationsfehler bei der Darstellung von Regionenrändern und XLD Konturen im Fenster WindowHandle. Ein Werte größer Null für Approximation bewirkt eine Vereinfachung/Glättung des Linienverlaufs mit weniger Punkten und ermöglicht eine schnellere Visualisierung. Dabei gibt Approximation die maximale Abweichung des angenäherten Linienzugs vom ursprünglichen in Pixel an (Ramer-Douglas-Peucker Algorithmus).

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **Approximation** (input_control) ..... integer ⇝ *integer*
Maximale Abweichung von der Originalkontur.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ Approximation
Restriktion: Approximation >= 0

Beispiel

```
* Calling...
set_line_approx(WindowHandle,Approximation)
set_draw(WindowHandle,'margin')
disp_region(Obj,WindowHandle)

* ...corresponds with
get_region_polygon(Obj,Approximation,Row,Col)
disp_polygon(WindowHandle,Row,Col)
```

Ergebnis

set_line_approx liefert den Wert TRUE, falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_line_approx

Nachfolger

disp_region

Alternativen

get_region_polygon, disp_polygon

Siehe auch

get_line_approx, set_line_style, set_draw, disp_region, disp_polygon

Modul

Foundation

set_lut_style (: : WindowHandle, Hue, Saturation, Intensity :)

Verändern der Farbtabelle.

set_lut_style ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

set_lut_style verändert die Farbtabelle des Gerätes, auf dem das Ausgabefenster dargestellt wird. Dabei haben die drei Parameter folgende Bedeutung:

Hue: Rotation des Farbraumes

Hue = 1.0 entspricht einer einmaligen Rotation des Farbraumes.

Keine Veränderung: Hue = 0.0

Komplementärfarben: Hue = 0.5

Saturation: Veränderung der Sättigung

Keine Veränderung: Saturation = 1.0

Grauwertbild: Saturation = 0.0

Intensity Veränderung der Helligkeit

Keine Veränderung: Intensity = 1.0

schwarzes Bild: Intensity = 0.0

Verändert werden nur der Teil der Farbtabelle, der für die Darstellung der Bildinformation verwendet wird. Die Modifikationsparameter bleiben erhalten bis set_lut_style erneut aufgerufen wird. Ein Aufruf von set_lut hat keinen Einfluss auf diese Parameter.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fensteridentifikator.
- ▷ **Hue** (input_control) ..... real ⇝ *real*
  Farbwertmodifikation.
  Defaultwert: 0.0
  Typischer Wertebereich: 0.0 ≤ Hue ≤ 1.0
  Restriktion: 0.0 >= Hue && Hue <= 1.0
- ▷ **Saturation** (input_control) ..... real ⇝ *real*
  Modifikation der Sättigung.
  Defaultwert: 1.5
  Typischer Wertebereich: 0.0 ≤ Saturation
  Restriktion: 0.0 <= Saturation
- ▷ **Intensity** (input_control) ..... real ⇝ *real*
  Modifikation der Intensität.
  Defaultwert: 1.5
  Typischer Wertebereich: 0.0 ≤ Intensity
  Restriktion: 0.0 <= Intensity

Beispiel

```
read_image(Image,'monkey')
dev_set_lut('color1')
repeat
  get_mbutton(WindowHandle,Row,Column,Button)
  Saturation := Row/300.0
  Hue := Column/512.0
  set_lut_style(WindowHandle,Hue,Saturation,1.0)
  dev_display(Image)
until(Button == 4)
```

Ergebnis

`set_lut_style` liefert den Wert TRUE, falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_lut_style`

Nachfolger

`set_lut`

Alternativen

`set_lut`, `scale_image`

Siehe auch

`get_lut_style`

Modul

Foundation

`set_pixel` (: : WindowHandle, Pixel :)

Setzen eines Index in der Farbtabelle.

`set_pixel` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`set_pixel` setzt die Pixelwerte, die mit `get_pixel` abgefragt wurden. Farben (`set_color`, `set_rgb`, etc.) und Grauwert (`set_gray`) werden gemeinsam in eine ganze Zahl kodiert. Diese Zahl wird Pixel genannt. Ein Pixel ist ein Index in der Farbtabelle und hat einen Wertebereich von 0,1 bei S/W-Bildschirmen und 0..255 bei Farbbildschirmen mit 8 Bildebenen. Dieses Pixel ist also nicht mit dem Pixel („picture element“) der Bildverarbeitung (Bildpunkt) zu verwechseln. Deshalb wird bei HALCON immer zwischen Pixel und Bildpunkt (oder auch Grauwert) unterschieden.

Dieser Code (Pixel) wird mit `get_pixel` abgefragt und kann mit `set_pixel` wieder zurückgesetzt werden. Der Wertebereich des Parameters beginnt bei Null und läuft bis zu einer oberen Grenze, wobei diese der Anzahl darstellbarer Graustufen bzw. Farbwerte der Ausgabegeräte entspricht.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`Pixel`** (input_control) ..... integer(-array) ⇝ *integer*
Index in der aktuellen Farbtabelle.
Defaultwert: 128
Typischer Wertebereich: $0 \leq$ `Pixel` ≤ 255

Ergebnis

`set_pixel` liefert den Wert TRUE, falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_pixel`

Nachfolger

`disp_image`, `disp_region`

Alternativen

`set_rgb`, `set_color`, `set_hsi`

Siehe auch

`get_pixel`, `set_lut`, `disp_region`, `disp_image`, `disp_color`

Modul

Foundation

set_tshape (: : WindowHandle, TextCursor :)

Form des Textcursors setzen.

`set_tshape` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`set_tshape` setzt die Darstellungsart, die für den Textcursor des Ausgabefensters verwendet werden soll.

Unter Textcursor (im Gegensatz zu Mauscursor; beide werden einfach als Cursor bezeichnet, wenn Missverständnisse ausgeschlossen sind) wird hier die Markierung der aktuellen Schreibposition verstanden.

Ein Tupel aller verfügbaren Arten kann mit dem Operator `query_tshape` abgefragt werden.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *handle*
Fenster-Handle.

▷ **`TextCursor`** (input_control) ..... string ⇝ *string*
Name der Cursorform.
Defaultwert: 'invisible'

Ergebnis

`set_tshape` liefert den Wert TRUE, falls das Fenster gültig ist und für dieses die angegebene Form definiert ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_window`, `query_tshape`, `get_tshape`

Nachfolger

`write_string`, `read_string`

Siehe auch

`get_tshape`, `query_tshape`, `write_string`, `read_string`

Modul

Foundation

slide_image (: : WindowHandleSource1, WindowHandleSource2, WindowHandle :)

Interaktives Ausgeben aus zwei Fenster-Puffern.

Der Operator slide_image funktioniert nicht mit HDevelop Grafikfenstern. Er wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

slide_image teilt das Fenster in Abhängigkeit von der Mausposition horizontal in zwei logische Bereiche. In den oberen wird der Inhalt des ersten angegebenen Fensters kopiert, in den unteren der Inhalt des zweiten Fensters. Drückt man die linke Maustaste, kann die Grenze zwischen den beiden Bereichen verschoben werden (die Maus darf dazu auch außerhalb des Fensters bewegt werden. Dabei legt die Position der Maus relativ zum Fenster die Trennlinie fest).

Durch einen Klick mit der rechten Maustaste in das Fenster wird der Operator slide_image beendet.

Der Operator slide_image lässt sich sinnvoll verwenden, um die Auswirkung etwa einer Filteroperation auf ein Bild zu visualisieren. Die Ausgabe erfolgt auf das aktuell gesetzte Fenster (WindowHandle).

Achtung

Die drei Fenster müssen die gleiche Größe haben und sich auf dem gleichen Rechner befinden.

Parameter

- ▷ **WindowHandleSource1** (input_control) ..... window ⇝ *handle*
  Logische Fensternummer des „oberen Fensters“.
- ▷ **WindowHandleSource2** (input_control) ..... window ⇝ *handle*
  Logische Fensternummer des „unteren Fensters“.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *handle*
  Fenster-Handle.

Beispiel

```
read_image (Image, 'fabrik')
sobel_dir (Image, EdgeAmplitude, EdgeDirection, 'sum_abs', 3)
dev_open_window (0, 0, 512, 512, 'black', WindowHandle1)
dev_display (EdgeAmplitude)
dev_open_window (0, 0, 512, 512, 'black', WindowHandle2)
dev_display (EdgeDirection)
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
slide_image (WindowHandle1, WindowHandle2, WindowHandle)
```

Ergebnis

slide_image liefert den Wert TRUE, falls beide Fenster existieren und ein Fenster davon gültig ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_window

Alternativen

copy_rectangle, get_mposition

Siehe auch

open_window, move_rectangle

Modul

Foundation

write_lut (: : WindowHandle, FileName :)

Schreiben der aktuellen Farbtabelle in eine Datei.

write_lut ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

write_lut legt die aktuelle Farbtabelle (das ist der für Grauwerte relevante Ausschnitt der Tabelle) des gültigen Ausgabefenster in der Datei FileName ab. Sie kann dann später mittels set_lut wieder geladen werden.

Achtung

write_lut kann nur auf Bildschirmen mit 256 Farben verwendet werden.

Parameter

▷ **WindowHandle** (input_control) . . . . . window ⇝ *handle*
Fensteridentifikator.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei für die aktuelle Farbtabelle.
Defaultwert: '/tmp/lut'
Dateiendung: .lut

Ergebnis

Ist der Dateiname in Ordnung und hat das Fenster die richtigen Eigenschaften (256 Farben), dann liefert write_lut den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_lut

Siehe auch

set_lut, set_pixel, get_pixel

Modul

Foundation

16.6 Klassifikation

clear_sampset (: : SampKey :)

Speicher eines Datensatzes freigeben.

clear_sampset ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

clear_sampset gibt den Speicherbereich, der von einem mit read_sampset eingelesenen Trainingsdatensatz belegt wird, wieder frei. Dieser kann nur mittels read_sampset wiederverwendet werden.

Parameter

▷ **SampKey** (input_control) . . . . . feature_set ⇝ *handle*
Nummer des Datensatzes.

Ergebnis

clear_sampset liefert den Wert TRUE. Eine Fehlerbehandlung wird ausgelöst, falls der Schlüssel SampKey nicht existiert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SampKey

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_box`, `enquire_class_box`, `learn_class_box`, `write_class_box`

Siehe auch

`test_sampset_box`, `learn_sampset_box`, `read_sampset`

Modul

Foundation

`close_class_box` (: : ClassifHandle :)

Löschen des Klassifikators.

`close_class_box` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

`close_class_box` löscht den Klassifikator und stellt den Speicherplatz wieder zur Verfügung. Die gesamte gelernte Information geht verloren. Es kann gegebenenfalls `write_class_box` verwendet werden um die trainierten Daten abzuspeichern.

Parameter

▷ **`ClassifHandle`** (input_control) . . . . . class_box ⇝ *handle*
Handle des Klassifikators.

Ergebnis

`close_class_box` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassifHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_box`, `enquire_class_box`, `learn_class_box`, `write_class_box`

Siehe auch

`create_class_box`, `enquire_class_box`, `learn_class_box`

Modul

Foundation

`create_class_box` (: : : ClassifHandle)

Erzeugen eines neuen Klassifikators.

create_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

create_class_box erzeugt einen neuen, lernfähigen Klassifikator. Alle im Kapitel Klassifikation beschriebenen Operatoren beziehen sich auf einen derart initialisierten Klassifikator (vom Typ 2). Eine genauere Beschreibung ist beim Operator learn_class_box zu finden.

Parameter

▷ **ClassifHandle** (output_control) . . . . . class_box ⇝ *handle*
Handle des Klassifikators.

Ergebnis

create_class_box liefert den Wert TRUE, falls der Parameter korrekt ist. Falls ein Klassifikator mit diesem Namen bereits existiert, oder nicht genügend Speicher vorhanden ist, wird eine Fehlerbehandlung ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

learn_class_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

learn_class_box, enquire_class_box, close_class_box

Modul

Foundation

descript_class_box (: : ClassifHandle, Dimensions : ClassIdx, BoxIdx, BoxLowerBound, BoxHigherBound, BoxNumSamplesTrain, BoxNumSamplesWrong)

Beschreiben des Box-Klassifikators.

descript_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

descript_class_box beschreibt die Klassen eines Box-Klassifikators. Ein Box-Klassifikator verwendet für jede Klasse eine Menge von Hyperquadern (Boxen). Diese Boxen repräsentieren die Verteilung der Merkmalsvektoren jeder Klasse.

descript_class_box liefert für jede Klasse (ClassIdx) die Grenzen aller enthaltener Boxen (BoxIdx) von Dimension 1 bis Dimensions (BoxLowerBound, BoxHigherBound) wie auch die Anzahl der Trainingsdaten, die während der Lernphase für jede Dimension verwendet wurden (BoxNumSamplesTrain). Des weiteren wird die Anzahl der Trainingsdaten zurückgegeben, die während der Trainingsphase der falschen Klasse zugeordnet wurden (BoxNumSamplesWrong).

Die Information über die Grenzen der Boxen kann zur Inspektion des Box-Klassifikators verwendet werden.

Parameter

▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *handle*
Handle des Klassifikators.

▷ **Dimensions** (input_control) . . . . . integer ⇝ *integer*
Höchste auszugebende Dimension.
Defaultwert: 3

▷ **ClassIdx** (output_control) ..... integer(-array) ⇝ *integer*
Indizes der Klassen.

▷ **BoxIdx** (output_control) ..... integer(-array) ⇝ *integer*
Indizes der Boxen.

▷ **BoxLowerBound** (output_control) ..... integer(-array) ⇝ *integer*
Untere Grenzen der Boxen (für jede Dimension).

▷ **BoxHigherBound** (output_control) ..... integer(-array) ⇝ *integer*
Obere Grenzen der Boxen (für jede Dimension).

▷ **BoxNumSamplesTrain** (output_control) ..... integer(-array) ⇝ *integer*
Anzahl der Trainingsdaten, die während der Lernphase (für jede Dimension) verwendet wurden.

▷ **BoxNumSamplesWrong** (output_control) ..... integer(-array) ⇝ *integer*
Anzahl der Trainingsdaten, die während der Lernphase der falschen Klasse zugewiesen wurden.

Ergebnis

`descript_class_box` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_class_box`, `learn_class_box`, `set_class_box_param`

Nachfolger

`enquire_class_box`, `learn_class_box`, `write_class_box`, `close_class_box`

Siehe auch

`create_class_box`, `enquire_class_box`, `learn_class_box`, `read_class_box`, `write_class_box`

Modul

Foundation

deserialize_class_box (: : ClassifHandle, SerializedItemHandle :)

Deserialisieren eines serialisierten Klassifikators.

deserialize_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

`deserialize_class_box` deserialisiert einen Klassifikator, welcher mit `serialize_class_box` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Der serialisierte Klassifikator wird in dem Handle `SerializedItemHandle` übergeben. Die Werte des Klassifikators, welcher in dem Handle `ClassifHandle` übergeben wird, werden dabei überschrieben.

Achtung

Alle Werte des Klassifikators werden überschrieben.

Parameter

▷ **ClassifHandle** (input_control) ..... class_box ⇝ *handle*
Handle des Klassifikators.

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_class_box` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_class_box, create_class_box

Nachfolger

test_sampset_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

create_class_box, serialize_class_box

Modul

Foundation

enquire_class_box (: : ClassifHandle, FeatureList : Class)

Klassifizieren eines Merkmalstupels.

enquire_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

FeatureList ist ein Tupel von beliebigen Gleitpunkt- oder ganzen Zahlen (Merkmalen), das mit Hilfe eines früher entsprechend trainierten (learn_class_box) Klassifikators einer Klasse zugeordnet werden soll. Es ist möglich, Merkmale als unbekannt anzugeben, indem anstelle einer Zahl das Zeichen '∗' angegeben wird. Falls n Werte angegeben sind, werden automatisch alle weiteren, also Merkmal n+1 bis max, als undefiniert angenommen.

Näheres zur Funktionsweise siehe learn_class_box.

Die Operatoren learn_class_box und enquire_class_box können abwechselnd aufgerufen werden, so dass bereits in der Lernphase klassifiziert werden kann. Auf diese Weise lässt sich feststellen, wann ein zufriedenstellendes Verhalten erreicht wurde.

Parameter

▷ **ClassifHandle** (input_control) ..... class_box ⇝ *handle*
Handle des Klassifikators.

▷ **FeatureList** (input_control) ..... real-array ⇝ *real* / integer / string
Merkmalsvektor, der klassifiziert werden soll.
Defaultwert: 1.0

▷ **Class** (output_control) ..... integer ⇝ *integer*
Nummer der Klasse, der der Merkmalsvektor zugeordnet wurde.

Ergebnis

enquire_class_box liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_class_box, learn_class_box, set_class_box_param

Nachfolger

learn_class_box, write_class_box, close_class_box

Alternativen

enquire_reject_class_box

Siehe auch

test_sampset_box, learn_class_box, learn_sampset_box

Modul

Foundation

enquire_reject_class_box (: : ClassifHandle, FeatureList : Class)

Klassifizieren eines Merkmalstupels mit Rückweisungsklasse.

enquire_reject_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

FeatureList ist ein Tupel von beliebigen Gleitpunkt- oder ganzen Zahlen (Merkmalen), das mit Hilfe eines früher entsprechend trainierten (learn_class_box) Klassifikators einer Klasse zugeordnet werden soll. Es ist möglich, Merkmale als unbekannt anzugeben, indem anstelle einer Zahl das Zeichen '$*$' angegeben wird. Falls n Werte angegeben sind, werden automatisch alle weiteren, also Merkmal n+1 bis max, als undefiniert angenommen.

Kann der Merkmalsvektor keiner Klasse zugeordnet werden, d.h. der Vektor liegt nicht innerhalb einer der hyper-Boxen, dann wird im Gegensatz zu enquire_class_box nicht die nächste Klasse, sonder die Rückweisungsklasse *-1* als Ergebnis übergeben.

Näheres zur Funktionsweise siehe learn_class_box.

Die Operatoren learn_class_box und enquire_class_box können abwechselnd aufgerufen werden, so dass bereits in der Lernphase klassifiziert werden kann. Auf diese Weise lässt sich feststellen, wann ein zufriedenstellendes Verhalten erreicht wurde.

Parameter

- ▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *handle*
  Handle des Klassifikators.
- ▷ **FeatureList** (input_control) . . . . . real-array ⇝ *real* / integer / string
  Merkmalsvektor, der klassifiziert werden soll.
  Defaultwert: 1.0
- ▷ **Class** (output_control) . . . . . integer ⇝ *integer*
  Nummer der Klasse, der der Merkmalsvektor zugeordnet wurde oder -1 für die Rückweisungsklasse.

Ergebnis

enquire_reject_class_box liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_class_box, learn_class_box, set_class_box_param

Nachfolger

learn_class_box, write_class_box, close_class_box

Alternativen

enquire_class_box

Siehe auch

test_sampset_box, learn_class_box, learn_sampset_box

Modul

Foundation

get_class_box_param (: : ClassifHandle, Flag : Value)

Abfragen von Informationen über den aktuellen Parameter.

get_class_box_param ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

get_class_box_param holt die Parameter des Klassifikators. Die Bedeutung der Parameter wird bei set_class_box_param näher erklärt.

Standardwerte: 'min_samples_for_split' = 80, 'split_error' = 0.1, 'prop_constant' = 0.25.

Parameter

▷ **ClassifHandle** (input_control) ..... class_box ⇝ *handle*
Handle des Klassifikators.

▷ **Flag** (input_control) ..... string ⇝ *string*
Name des Systemparameters.
Defaultwert: 'split_error'
Werteliste: Flag ∈ {'split_error', 'prop_constant', 'used_memory', 'min_samples_for_split'}

▷ **Value** (output_control) ..... number ⇝ *real / integer*
Wert des Systemparameters.

Ergebnis

get_class_box_param liefert den Wert TRUE. Eine Fehlerbehandlung wird ausgelöst, falls Flag mit falschen Werten belegt wird.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_class_box, enquire_class_box, learn_class_box, write_class_box

Nachfolger

set_class_box_param, learn_class_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

create_class_box, set_class_box_param

Modul

Foundation

learn_class_box (: : ClassifHandle, Features, Class :)

Trainieren des Klassifikators.

learn_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

Features ist ein Tupel von beliebigen Gleitpunktzahlen oder ganzen Zahlen (Merkmalen), das der Klasse Class, die durch eine ganze Zahl angegeben wird, zugeordnet werden soll. Der Operatoren enquire_class_box kann später verwendet werden, um zu beliebigen Vektoren (=Tupeln) die plausibelste Klasse zu finden. Das Verfahren versucht, die Menge der zu einer Klasse gehörigen Vektoren durch Hyperquader im Merkmalsraum zu beschreiben, wobei nach Bedarf auch mehrere Quader pro Klasse erzeugt werden. Daher ist es auch möglich, disjunkte Konzepte zu lernen, das heißt solche, die in mehrere „Punktwolken“ im Merkmalsraum zerfallen. Die Datenstruktur bleibt dem Benutzer verborgen und ist nur mit Hilfe der in diesem Abschnitt beschriebenen Operatoren zugänglich.

Falls eine Klasse aus disjunkten Unterklassen besteht, die zum Aufspalten des Hyperquaders führen, sollte die Reihenfolge der Trainingsdaten in Bezug auf die Unterklassen zufällig sein. Ansonsten kann der Hyperquader nicht optimal aufgespaltet werden.

Es ist möglich, Merkmale als unbekannt anzugeben, indem anstelle einer Zahl das Zeichen '$*$' angegeben wird. Falls n Werte angegeben werden, werden automatisch alle weiteren, also Merkmal n+1 bis max, als undefiniert angenommen.

Die Operatoren `learn_class_box` und `enquire_class_box` können abwechselnd aufgerufen werden, so dass bereits in der Lernphase klassifiziert werden kann. Auf diese Weise lässt sich feststellen, wann ein zufriedenstellendes Verhalten erreicht wurde.

Der Klassifikator wird durch weiteres Training nur größer, das heißt es ist nicht ratsam, nach Erreichen eines zufriedenstellenden Verhaltens weiter zu trainieren.

Parameter

▷ **`ClassifHandle`** (input_control) . . . . . class_box ⇝ *handle*
Handle des Klassifikators.

▷ **`Features`** (input_control) . . . . . number-array ⇝ *real* / integer / string
Zu lernender Merkmalsvektor.
Defaultwert: [1.0,1.5,2.0]

▷ **`Class`** (input_control) . . . . . integer ⇝ *integer*
Klasse, dem der Vektor zugeordnet werden soll.
Defaultwert: 1

Ergebnis

`learn_class_box` liefert im Normalfall den Wert TRUE. Eine Fehlerbehandlung wird ausgelöst, falls Speicherplatzprobleme auftreten. Die Anzahl der Klassen ist beschränkt, bei Überschreitung dieser Grenze tritt ebenfalls eine Fehlerbehandlung ein.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassifHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_box`, `enquire_class_box`

Nachfolger

`test_sampset_box`, `enquire_class_box`, `write_class_box`, `close_class_box`, `clear_sampset`

Siehe auch

`test_sampset_box`, `close_class_box`, `create_class_box`, `enquire_class_box`, `learn_sampset_box`

Modul

Foundation

`learn_sampset_box` `( : : ClassifHandle, SampKey, Outfile, NSamples, StopError, ErrorN : )`

Trainieren des Klassifikators mit einem Datensatz.

`learn_sampset_box` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

learn_sampset_box trainiert den Klassifikator mit den Daten zum Schlüssel SampKey (siehe read_sampset). Der Trainingsvorgang wird spätestens nach NSamples Beispielen abgebrochen. Falls NSamples größer als die Anzahl der Beispiele in SampKey ist, wird zyklisch von vorne begonnen. Falls der Fehler den Wert StopError unterschreitet, wird der Trainingsvorgang vorzeitig beendet. StopError berechnet sich dabei N / ErrorN, wobei N die Anzahl der Beispiele ist, die während der letzten ErrorN Trainingsbeispiele falsch klassifiziert wurden. Typischerweise ist ErrorN die Anzahl der Beispiele in SampKey und NSamples ein Vielfaches hiervon. Soll ein Datensatz mit 100 Beispielen höchstens fünf mal durchlaufen werden und bei einem Fehler von weniger als 5% abgebrochen werden, dann lauten die entsprechenden Werte NSamples = 500, ErrorN = 100, StopError = 0.05. Ein Protokoll des Lernvorgangs wird in die Datei Outfile geschrieben.

Parameter

- ▷ **ClassifHandle** (input_control) ..... class_box ⇝ *handle*
  Handle des Klassifikators.
- ▷ **SampKey** (input_control) ..... feature_set ⇝ *handle*
  Nummer des Trainingsdatensatzes.
- ▷ **Outfile** (input_control) ..... filename.write ⇝ *string*
  Name der Protokolldatei.
  Defaultwert: 'training_prot'
- ▷ **NSamples** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu lernenden Merkmalsvektoren.
  Defaultwert: 500
- ▷ **StopError** (input_control) ..... real ⇝ *real*
  Klassifikationsfehler für Abbruch.
  Defaultwert: 0.05
- ▷ **ErrorN** (input_control) ..... integer ⇝ *integer*
  Fehler bei der Zuordnung
  Defaultwert: 100

Ergebnis

learn_sampset_box liefert den Wert TRUE. Eine Fehlerbehandlung wird ausgelöst, falls der Schlüssel SampKey nicht existiert oder Probleme beim Öffnen der Datei auftreten.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassifHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_class_box

Nachfolger

test_sampset_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

test_sampset_box, enquire_class_box, learn_class_box, read_sampset

Modul

Foundation

read_class_box (: : ClassifHandle, FileName :)

Einlesen eines Klassifikators von Datei.

read_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

read_class_box übernimmt den in der Datei FileName (siehe write_class_box) gespeicherten Klassifikator. Die Werte des Klassifikators werden dabei überschrieben. Die Dateiendung für den Box-Klassifikator ist in HALCON 'gbc'.

Achtung

Alle Werte des Klassifikators werden überschrieben.

Parameter

- ▷ **ClassifHandle** (input_control) ..... class_box ⇝ *handle*
  Handle des Klassifikators.
- ▷ **FileName** (input_control) ..... filename.read ⇝ *string*
  Dateiname des Klassifikators.
  Dateiendung: .gbc

Ergebnis

read_class_box liefert den Wert TRUE. Eine Fehlerbehandlung wird ausgelöst, falls die Datei FileName nicht geöffnet werden konnte, oder falls die Datei das falsche Format hatte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassifHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_class_box

Nachfolger

test_sampset_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

create_class_box, write_class_box

Modul

Foundation

read_sampset (: : FileName : SampKey)

Einlesen eines Trainingsdatensatzes von Datei.

read_sampset ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

Die Trainingsbeispiele sind über den Schlüssel SampKey mit den Operatoren clear_sampset und learn_sampset_box zugreifbar. Die Datei kann mit einem Editor erstellt werden. Jede Zeile enthält einen Merkmalsvektor mit zugehöriger Klasse. Ein Beispiel für das Format:

```
(1.0, 25.3, *, 17 | 3)
```

Diese Zeile spezifiziert einen Merkmalsvektor, der zur Klasse 3 gehört und bei dem das dritte Merkmal unbekannt ist. Merkmale von fünf aufwärts werden ebenfalls als unbekannt angenommen. Kommentare der Form /* ... */ können an beliebigen Stellen eingefügt werden.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname des Trainingsdatensatzes.
Defaultwert: 'sampset1'

▷ **SampKey** (output_control) . . . . . feature_set ⇝ *handle*
Identifikation des Trainingsdatensatzes.

Ergebnis

`read_sampset` liefert den Wert TRUE. Eine Fehlerbehandlung wird ausgelöst, falls die Datei nicht geöffnet werden kann, falls sie Syntaxfehler enthält oder falls nicht genug Speicher zur Verfügung steht.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`create_class_box`

Nachfolger

`test_sampset_box`, `enquire_class_box`, `write_class_box`, `close_class_box`, `clear_sampset`

Siehe auch

`test_sampset_box`, `clear_sampset`, `learn_sampset_box`

Modul

Foundation

serialize_class_box (: : ClassifHandle : SerializedItemHandle)

Serialisieren eines Klassifikators.

serialize_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

`serialize_class_box` serialisiert einen Klassifikator (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_class_box` in eine Datei geschrieben werden. Der Klassifikator wird in dem Handle `ClassifHandle` übergeben. Der serialisierte Klassifikator wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_class_box` wieder deserialisiert werden.

Parameter

▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *handle*
Handle des Klassifikators.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_class_box` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_class_box`, `enquire_class_box`, `learn_class_box`, `test_sampset_box`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_class_box`

Siehe auch

`create_class_box`, `deserialize_class_box`

Modul

Foundation

`set_class_box_param` (: : ClassifHandle, Flag, Value :)

Setzen von Systemparametern zur Klassifikation.

`set_class_box_param` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

`set_class_box_param` ändert Parameter, die den Lernvorgang beim Aufruf von `learn_class_box` beeinflussen. Die Parameter werden nur für den Klassifikator `ClassifHandle` geändert, für alle anderen Klassifikatoren bleiben sie unverändert. *'min_samples_for_split'* ist die Anzahl der Trainingsbeispiele, die mindestens in einen Quader des Klassifikators gefallen sein müssen, bevor sich dieser Quader teilen darf. *'split_error'* gibt den kritischen Fehler an, bei dessen Überschreitung sich ein Quader teilt, falls er schon mehr als *'min_samples_for_split'* Trainingsbeispiele eingefangen hat. *'prop_constant'* beeinflusst die Ausdehnung der Quader. Sie ist proportional zum durchschnittlichen Abstand der Trainingsbeispiele in diesem Quader zum Mittelpunkt des Quaders. Genauer gesagt gilt:

Ausdehnung $\times$ Prop = durchschnittlicher Abstand vom Erwartungswert.

Diese Beziehung gilt entlang jeder Dimension. Innerhalb eines Quaders werden also die Dimensionen des Merkmalsraums als unabhängig angenommen.

Die Parameter sind mit weitgehend problemunabhängigen Defaultwerten vorbesetzt, die nicht grundlos verändert werden sollten. Die Parameter sind nur beim Lernvorgang von Bedeutung. Sie haben keinen Einfluss auf das Verhalten von `enquire_class_box`.

Standardwerte: 'min_samples_for_split' = 80,

'split_error' = 0.1,

'prop_constant' = 0.25.

Parameter

- ▷ **`ClassifHandle`** (input_control) ..... class_box ⇝ *handle*
  Handle des Klassifikators.
- ▷ **`Flag`** (input_control) ..... string ⇝ *string*
  Name des gewünschten Parameters.
  Defaultwert: 'split_error'
  Wertevorschläge: `Flag` ∈ {'min_samples_for_split', 'split_error', 'prop_constant'}
- ▷ **`Value`** (input_control) ..... number ⇝ *real* / integer
  Wert des Parameters.
  Defaultwert: 0.1

Ergebnis

`set_class_box_param` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassifHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_box`, `enquire_class_box`

Nachfolger

`learn_class_box`, `test_sampset_box`, `write_class_box`, `close_class_box`, `clear_sampset`

Siehe auch

`enquire_class_box`, `get_class_box_param`, `learn_class_box`

Modul

Foundation

`test_sampset_box` (: : ClassifHandle, SampKey : Error)

Klassifizieren einer Menge von Vektoren.

`test_sampset_box` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

Im Unterschied zu `learn_sampset_box` findet kein Lernen statt. Typischerweise verwendet man `test_sampset_box`, um einen Klassifikator auf unabhängige Testdaten anzuwenden. `Error` gibt dann Auskunft über die Anwendbarkeit des Gelernten auf neue Beispiele.

Parameter

- ▷ **`ClassifHandle`** (input_control) ..... class_box ⇝ *handle*
  Handle des Klassifikators.
- ▷ **`SampKey`** (input_control) ..... feature_set ⇝ *handle*
  Schlüssel der Testdaten.
- ▷ **`Error`** (output_control) ..... real ⇝ *real*
  Fehler bei der Zuordnung.

Ergebnis

`test_sampset_box` liefert den Wert TRUE. Eine Fehlerbehandlung wird ausgelöst, falls der Schlüssel `SampKey` nicht existiert oder Probleme beim Öffnen der Datei auftreten.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_class_box`, `learn_class_box`, `set_class_box_param`

Nachfolger

`enquire_class_box`, `learn_class_box`, `write_class_box`, `close_class_box`, `clear_sampset`

Siehe auch

`enquire_class_box`, `learn_class_box`, `learn_sampset_box`, `read_sampset`

Modul

Foundation

write_class_box (: : ClassifHandle, FileName :)

Ausgeben des Klassifikators in eine Datei.

write_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die MLP, SVM, KNN oder GMM Operatoren verwenden.

write_class_box speichert den Klassifikator ClassifHandle in der Datei FileName ab. Der Klassifikator kann mit read_class_box wieder eingelesen werden. Die Dateiendung für den Box-Klassifikator ist in HALCON 'gbc'.

Achtung

Falls eine Datei mit diesem Namen bereits existiert, wird sie ohne Warnung überschrieben. Die Datei kann nicht editiert werden.

Parameter

- ▷ **ClassifHandle** (input_control) ..... class_box ⇝ *handle*
  Handle des Klassifikators.
- ▷ **FileName** (input_control) ..... filename.write ⇝ *string*
  Name der Datei, auf die die Daten geschrieben werden.
  Dateiendung: .gbc

Ergebnis

write_class_box liefert den Wert TRUE. Eine Fehlerbehandlung wird ausgelöst, falls die Datei FileName nicht geöffnet werden konnte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_class_box, enquire_class_box, learn_class_box, test_sampset_box

Nachfolger

close_class_box, clear_sampset

Siehe auch

create_class_box, read_class_box

Modul

Foundation

16.7 Matching

adapt_template (Image : : TemplateID :)

Anpassen eines Templates an die Größe eines Bildes.

adapt_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

adapt_template dient zur Anpassung eines Templates, das mit create_template erzeugt wurde, an die Größe eines Bildes. Vor der ersten Verwendung eines Templates mit Bildern einer anderen Größe kann adapt_template aufgerufen werden um Laufzeit beim ersten Aufruf des Matching Operators zu sparen. Wird adapt_template nicht explizit verwendet, erfolgt der Aufruf implizit beim ersten Matching mit einer anderen Bildgröße. Der Inhalt des Bildes ist irrelevant. Es wird nur die Breite von Image verwendet.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
Bild, das die Größe des späteren Matching vorgibt.

▷ **TemplateID** (input_control) . . . . . template ⇝ *handle*
Nummer des Templates.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `adapt_template` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TemplateID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_template`, `create_template_rot`, `read_template`

Nachfolger

`set_reference_template`, `best_match`, `fast_match`, `fast_match_mg`, `set_offset_template`, `best_match_mg`, `best_match_pre_mg`, `best_match_rot`, `best_match_rot_mg`

Modul

Matching

best_match (Image : : TemplateID, MaxError, SubPixel : Row, Column, Error)

Suche des besten Matching zwischen einem Template und einem Bild.

best_match ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

`best_match` führt ein Matching zwischen dem Template von `TemplateID` und `Image` durch. Dabei wird das Template so über die Punkte von `Image` geschoben, dass es immer vollständig innerhalb von `Image` liegt. `best_match` arbeitet ähnlich wie `fast_match` mit der Erweiterung, dass jedes mal wenn eine Position mit einem geringeren Matching-Fehler gefunden wurde, der Parameter `MaxError` intern entsprechend angepasst (d.h. verkleinert) wird, um die Rechenzeit zu verkürzen. Abhängig vom Parameter `SubPixel` wird die Position in Subpixelgenauigkeit ausgegeben. Das Matching-Kriterium („displaced frame difference“) ist wie folgt definiert:

$$error[row, col] = \frac{\sum_{u,v} |\texttt{Image}[row-u, col-v] - \texttt{TemplateID}[u,v]|}{area(TemplateID)}$$

Die Laufzeit des Verfahrens hängt von der Größe des Definitionsbereiches von `Image` ab. Es ist daher wichtig, den Definitionsbereich möglichst einzugrenzen, d.h. den Operator nur in einer möglichst eng umrissenen „region of interest“ anzuwenden. Der Parameter `MaxError` legt den maximalen Fehler fest, den die gesuchte Position haben darf. Um so kleiner dieser Wert ist, um so schneller läuft das Verfahren.

`Row` und `Column` liefern die Position des Best-Match, wobei `Error` die mittlere Abweichung der Grauwerte angibt.

Falls keine Position mit einem Fehler unter `MaxError` gefunden wurde, wird die Position $(0, 0)$ und ein Matching-Fehler von 255 für `Error` geliefert. In diesem Fall muss der Wert für `MaxError` größer gewählt werden.

Der maximale Fehler der Position (ohne Rauschen) beträgt im sub-Pixel Modus 0.1 Pixel. Der mittlere Fehler beträgt 0.03 Pixel.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte
Eingabebild, in dem das Pattern gefunden werden soll.

▷ **TemplateID** (input_control) . . . . . template ⇝ *handle*
Nummer des Templates.

▷ **MaxError** (input_control) . . . . . real ⇝ *real*
Maximale mittlere Differenz der Grauwerte.
Defaultwert: 20.0
Wertevorschläge: `MaxError` ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 17.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0}
Typischer Wertebereich: $0 \leq$ `MaxError` ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 3

▷ **SubPixel** (input_control) . . . . . string ⇝ *string*
Subpixel Genauigkeit falls *'true'*.
Defaultwert: 'false'
Werteliste: `SubPixel` ∈ {'true', 'false'}

▷ **Row** (output_control) . . . . . point.y(-array) ⇝ *real*
Zeilenposition des Best-Match.

▷ **Column** (output_control) . . . . . point.x(-array) ⇝ *real*
Spaltenposition des Best-Match.

▷ **Error** (output_control) . . . . . real(-array) ⇝ *real*
Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `best_match` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`create_template`, `read_template`, `set_offset_template`, `set_reference_template`, `adapt_template`, `draw_region`, `draw_rectangle1`, `reduce_domain`

Alternativen

`fast_match`, `fast_match_mg`, `best_match_mg`, `best_match_pre_mg`, `best_match_rot`, `best_match_rot_mg`, `exhaustive_match`, `exhaustive_match_mg`

Modul

Matching

best_match_mg (Image : : TemplateID, MaxError, SubPixel, NumLevels, WhichLevels : Row, Column, Error)

Suche nach dem besten Grauwert-Match in einer Pyramide.

best_match_mg ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

`best_match_mg` führt eine Suche nach dem besten Grauwert-Match in einer Pyramide durch. `best_match_mg` arbeitet analog zu `best_match`, es wird jedoch die Suche durch die Verwendung einer Auflösungspyramide beschleunigt. Als Eingabe dient ein Bild, eventuell mit eingeschränktem Definitionsbereich. Der

Grauwert-Match wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen der Grauwert-Match vollständig in das Bild passt. Das bedeutet, dass kein Match gefunden wird, wenn er aus dem Bild herausragt. Der Parameter `MaxError` gibt den maximalen Fehler vor, der bei dem Mustervergleich auftreten darf. Bei einem kleinen Wert beschleunigt sich der Operator, es kann jedoch vorkommen, dass das Muster nicht gefunden wird. Der Wert von `MaxError` muss gegenüber `best_match` größer gewählt werden, da der Matching-Fehler auf höheren Pyramidenebenen oft größer ist.

Der Parameter `SubPixel` legt fest, ob das Ergebnis mit subpixel Genauigkeit bestimmt werden soll. `NumLevels` bestimmt wie viele Pyramidenstufen für die Suche verwendet werden. Hat `NumLevels` den Wert *1*, dann ist das Verfahren identisch mit `best_match`, d.h. es wird nur auf den Originaldaten gerechnet. Für Werte größer als *1*, beginnt das Verfahren die Suche bei der geringsten Auflösung und sucht dort die Position mit dem geringsten Fehler. In der nächst höheren Auflösung wird dann die gefundene Position verfeinert. Dies wird bis zur höchsten Auflösung (Originaldaten) vorgesetzt (`WhichLevels` = *'all'*). Als alternatives Verfahren kann der Modus `WhichLevels` mit den Wert *'original'* verwendet werden. Hierbei wird auf allen Ebenen mit reduzierter Auflösung nicht nur der Punkt mit dem geringsten Fehler, sondern alle Punkte die unterhalb von `MaxError` liegen weiter untersucht. Dieses Verfahren ist langsamer, aber die Gefahr, dass der richtige Punkt nicht gefunden wird, ist geringer. Gegebenenfalls kann in diesem Modus eine geringere Auflösung, d.h. eine höhere Pyramidenstufe verwendet werden, was zu einer verbesserten Laufzeit führt. Neben den Modi *'all'* und *'original'* für `WhichLevels` kann die gewünschte Ebene bei der zwischen den Verfahren umgeschaltet werden soll auch explizit angegeben werden. Hier entspricht *0* dem Wert *'original'* und `NumLevels` - 1 entspricht dem Wert *'all'*. Ein Wert zwischen diesen beiden Extremen ist in den meisten Fällen ein guter Kompromiss zwischen Stabilität und Laufzeit. Ein größerer Wert für `WhichLevels` verkürzt die Laufzeit, während ein kleinerer Wert das Verhalten stabiler macht. Der Wert für `NumLevels` muss gleich oder kleiner sein als die Anzahl der Ebenen, die bei der Erzeugung des Templates verwendet wurde.

Die gefunden Position des Musters wird in `Row`, `Column` übergeben. Der hierbei aufgetretene Fehler steht in `Error`. Wenn kein Punkt unterhalb von `MaxError` gefunden wird hat `Error` den Wert *255* und `Row` und `Column` haben den Wert *0*. Der maximale Fehler der Position (ohne Rauschen) beträgt im sub-Pixel Modus 0.1 Pixel. Der mittlere Fehler beträgt 0.03 Pixel.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild, in dem das Pattern gefunden werden soll.
- ▷ **`TemplateID`** (input_control) ..... template ⇝ *handle*
  Nummer des Templates.
- ▷ **`MaxError`** (input_control) ..... real ⇝ *real*
  Maximale mittlere Differenz der Grauwerte.
  Defaultwert: 30.0
  Wertevorschläge: `MaxError` ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 17.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0}
  Typischer Wertebereich: 0 ≤ `MaxError` ≤ 255
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 3
- ▷ **`SubPixel`** (input_control) ..... string ⇝ *string*
  Subpixel Genauigkeit falls *'true'*.
  Defaultwert: 'false'
  Werteliste: `SubPixel` ∈ {'true', 'false'}
- ▷ **`NumLevels`** (input_control) ..... integer ⇝ *integer*
  Anzahl der verwendeten Auflösungsebenen.
  Defaultwert: 4
  Wertevorschläge: `NumLevels` ∈ {1, 2, 3, 4, 5, 6}
- ▷ **`WhichLevels`** (input_control) ..... integer ⇝ *integer* / string
  Auflösungsebene bis zu der die Methode „best match“ verwendet wird.
  Defaultwert: 2
  Wertevorschläge: `WhichLevels` ∈ {'all', 'original', 0, 1, 2, 3, 4, 5, 6}
- ▷ **`Row`** (output_control) ..... point.y ⇝ *real*
  Zeilenposition des Best-Match.
- ▷ **`Column`** (output_control) ..... point.x ⇝ *real*
  Spaltenposition des Best-Match.

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert best_match_mg den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

create_template, read_template, adapt_template, draw_region, draw_rectangle1, reduce_domain, set_reference_template, set_offset_template

Alternativen

fast_match, fast_match_mg, best_match, best_match_pre_mg, best_match_rot, best_match_rot_mg, exhaustive_match, exhaustive_match_mg

Modul

Matching

best_match_pre_mg (ImagePyramid : : TemplateID, MaxError, SubPixel, NumLevels, WhichLevels : Row, Column, Error)

Suche nach dem besten Grauwert-Match in einer vorherberechneten Pyramide.

best_match_pre_mg ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

best_match_pre_mg führt eine Suche nach dem besten Grauwert-Match in einer Pyramide durch. best_match_pre_mg arbeitet analog zu best_match_mg, mit dem Unterschied, dass die Berechnung der Pyramide vor dem Aufruf mit gen_gauss_pyramid durchgeführt wurde. Dies verkürzt die Laufzeit, falls das Matching mehrfach durchgeführt werden soll, oder die Pyramide auch bei anderen Operationen genötigt wird. Die Pyramide muss mit dem Verkleinerungsfaktor *0.5* und dem Modus *'constant'* erzeugt werden.

Parameter

▷ **ImagePyramid** (input_object) . . . . . singlechannelimage-array ⇝ *object :* byte
Bildpyramide, in dem das Pattern gefunden werden soll.

▷ **TemplateID** (input_control) . . . . . template ⇝ *handle*
Nummer des Templates.

▷ **MaxError** (input_control) . . . . . real ⇝ *real*
Maximale mittlere Differenz der Grauwerte.
Defaultwert: 30.0
Wertevorschläge: MaxError ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 17.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0}
Typischer Wertebereich: $0 \leq$ MaxError ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 3

▷ **SubPixel** (input_control) . . . . . string ⇝ *string*
Subpixel Genauigkeit falls *'true'*.
Defaultwert: 'false'
Werteliste: SubPixel ∈ {'true', 'false'}

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer*
Anzahl der verwendeten Auflösungsebenen.
Defaultwert: 3
Wertevorschläge: NumLevels ∈ {1, 2, 3, 4, 5, 6}

▷ **WhichLevels** (input_control) . . . . . integer ⇝ *integer* / string
Auflösungsebene bis zu der die Methode „best match“ verwendet wird.
Defaultwert: ’original’
Wertevorschläge: WhichLevels ∈ {’all’, ’original’, 0, 1, 2, 3, 4, 5, 6}

▷ **Row** (output_control) . . . . . point.y ⇝ *real*
Zeilenposition des Best-Match.

▷ **Column** (output_control) . . . . . point.x ⇝ *real*
Spaltenposition des Best-Match.

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert best_match_pre_mg den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_gauss_pyramid, create_template, read_template, adapt_template, draw_region, draw_rectangle1, reduce_domain, set_reference_template

Alternativen

fast_match, fast_match_mg, exhaustive_match, exhaustive_match_mg

Modul

Matching

best_match_rot (Image : : TemplateID, AngleStart, AngleExtend, MaxError, SubPixel : Row, Column, Angle, Error)

Suche des besten Matching zwischen einem Template und einem Bild mit Rotation.

best_match_rot ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

best_match_rot führt ein Matching zwischen dem Template von TemplateID und Image durch. Es arbeitet analog zu best_match mit der Erweiterung, dass das Muster gedreht vorliegen kann. Die Parameter AngleStart und AngleExtend legen die maximale Rotation des Musters fest: AngleStart gibt größte Rotation gegen den Uhrzeiger an und AngleExtend die größte Rotation im Uhrzeigersinn relativ zu diesem Winkel. Beide Werte müssen kleiner oder gleich den entsprechenden Werte des Template bei dessen Erzeugung sein (siehe create_template_rot). Als Erweiterung gegenüber best_match liefert best_match_rot zu zusätzlichen Ausgabeparameter Angle der die gefundenen Rotationswinkel des Musters angibt. Die Genauigkeit hängt von dem Parameter AngleStep von create_template_rot ab. Im Fall von SubPixel = *’true’* wird die Position und der Winkel mit „subpixel“ Genauigkeit bestimmt.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object : byte*
Eingabebild, in dem das Pattern gefunden werden soll.

▷ **TemplateID** (input_control) . . . . . template ⇝ *handle*
Nummer des Templates.

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinster auftretende Rotation des Musters.
Defaultwert: -0.39
Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtend** (input_control) . . . . . angle.rad ⇝ *real*
Maximale positive Abweichung von AngleStart.
Defaultwert: 0.79
Wertevorschläge: AngleExtend ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtend > 0

▷ **MaxError** (input_control) . . . . . real ⇝ *real*
Maximale mittlere Differenz der Grauwerte.
Defaultwert: 30.0
Wertevorschläge: MaxError ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 17.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0}
Typischer Wertebereich: 0 ≤ MaxError ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 3

▷ **SubPixel** (input_control) . . . . . string ⇝ *string*
Subpixel Genauigkeit falls *'true'*.
Defaultwert: 'false'
Werteliste: SubPixel ∈ {'true', 'false'}

▷ **Row** (output_control) . . . . . point.y(-array) ⇝ *real*
Zeilenposition des Best-Match.

▷ **Column** (output_control) . . . . . point.x(-array) ⇝ *real*
Spaltenposition des Best-Match.

▷ **Angle** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Rotationswinkel des Musters.

▷ **Error** (output_control) . . . . . real(-array) ⇝ *real*
Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert best_match_rot den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

create_template_rot, read_template, set_offset_template, set_reference_template, adapt_template, draw_region, draw_rectangle1, reduce_domain

Alternativen

best_match_rot_mg

Siehe auch

best_match, best_match_mg

Modul

Matching

best_match_rot_mg (Image : : TemplateID, AngleStart, AngleExtend, MaxError, SubPixel, NumLevels : Row, Column, Angle, Error)

Suche des besten Matching zwischen einem Template und einer Pyramide mit Rotation.

best_match_rot_mg ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

best_match_rot_mg führt ein Matching zwischen dem Template von TemplateID und Image durch. Es arbeitet analog zu best_match_mg mit der Erweiterung, dass das Muster wie bei best_match_rot gedreht

vorliegen kann. Die Parameter AngleStart und AngleExtend legen die maximale Rotation des Musters fest: AngleStart gibt größte Rotation gegen den Uhrzeiger an und AngleExtend die größte Rotation im Uhrzeigersinn relativ zu diesem Winkel. Beide Werte müssen kleiner oder gleich den entsprechenden Werte des Template bei dessen Erzeugung sein (siehe create_template_rot). Als Erweiterung gegenüber best_match_mg liefert best_match_rot_mg zu zusätzlichen Ausgabeparameter Angle der den gefundenen Rotationswinkel des Musters angibt.

Der Wert von MaxError muss gegenüber best_match_rot größer gewählt werden, da der Matching-Fehler auf höheren Pyramidenebenen oft größer ist.

Im Fall von SubPixel = *'true'* wird die Position und der Winkel mit „subpixel“ Genauigkeit bestimmt.

Der Wert von NumLevels muss kleiner oder gleich dem Wert sein, der bei der Erzeugung des Templates verwendet wurde.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild, in dem das Pattern gefunden werden soll.
- ▷ **TemplateID** (input_control) ..... template ⇝ *handle*
  Nummer des Templates.
- ▷ **AngleStart** (input_control) ..... angle.rad ⇝ *real*
  Kleinster auftretende Rotation des Musters.
  Defaultwert: -0.39
  Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **AngleExtend** (input_control) ..... angle.rad ⇝ *real*
  Maximale positive Abweichung von AngleStart.
  Defaultwert: 0.79
  Wertevorschläge: AngleExtend ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
  Restriktion: AngleExtend > 0
- ▷ **MaxError** (input_control) ..... real ⇝ *real*
  Maximale mittlere Differenz der Grauwerte.
  Defaultwert: 40.0
  Wertevorschläge: MaxError ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 17.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0}
  Typischer Wertebereich: 0 ≤ MaxError ≤ 255
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **SubPixel** (input_control) ..... string ⇝ *string*
  Subpixel Genauigkeit falls *'true'*.
  Defaultwert: 'false'
  Werteliste: SubPixel ∈ {'true', 'false'}
- ▷ **NumLevels** (input_control) ..... integer ⇝ *integer*
  Anzahl der verwendeten Auflösungsebenen.
  Defaultwert: 3
  Wertevorschläge: NumLevels ∈ {1, 2, 3, 4, 5, 6}
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenposition des Best-Match.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenposition des Best-Match.
- ▷ **Angle** (output_control) ..... angle.rad(-array) ⇝ *real*
  Rotationswinkel des Musters.
- ▷ **Error** (output_control) ..... real(-array) ⇝ *real*
  Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert best_match_rot_mg den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

create_template_rot, set_reference_template, set_offset_template, adapt_template, draw_region, draw_rectangle1, reduce_domain

Alternativen

best_match_rot, best_match_mg, find_ncc_model, find_ncc_models

Siehe auch

fast_match

Modul

Matching

clear_template (: : TemplateID :)

Freigabe des Speichers eines Templates.

clear_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

clear_template gibt den Speicher eines Templates, das mit create_template oder create_template_rot angelegt wurde, wieder frei. Das Template kann nach dem Aufruf nicht mehr verwendet werden. Der Wert von TemplateID ist ungültig. Die Nummer kann jedoch bei nachfolgenden Aufrufen von create_template oder create_template_rot wieder vergeben werden.

Parameter

▷ **TemplateID** (input_control) ..... template ⇝ *handle*
Nummer des Templates.

Ergebnis

Ist die Nummer des Templates gültig, dann liefert clear_template den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TemplateID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_template, create_template_rot, read_template, write_template

Modul

Matching

create_template (Template : : FirstError, NumLevel, Optimize, GrayValues : TemplateID)

Aufbereiten eines Musters für Template Matching. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

create_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

create_template bereitet ein Muster (Template), das als Bild übergeben wird, für das Template Matching vor. Dem Template wird nach der Transformation eine Nummer zugewiesen (TemplateID), die bei der weiteren Verarbeitung verwendet wird. Die Form und Größe von Template sind beliebig. Es ist jedoch zu beachten, dass das Matching nur in dem Teil des Bildes ausgeführt wird, in dem Template vollständig hineinpasst.

Das Template sollte so gewählt werden, dass keine Grauwerte des Hintergrundes enthalten sind. Hierbei kann ausgenutzt werden, dass die Form des Template nicht auf Rechtecke beschränkt ist. Zu Gewinnung der Templateregion können Segmentierungsoperatoren wie threshold oder die draw_* Operatoren verwendet werden. Soll die Position des Templates später mit sub-Pixel-Genauigkeit bestimmt werden, dann muss Template zusätzlich um ein Pixel kleiner sein als das eigentliche Muster. Die kann z.B. mit dem Operator erosion_circle realisiert werden.

Der Parameter NumLevel gibt die Anzahl der Pyramidenebenen an (NumLevel = 1 bedeutet, dass nur die Originalgrauwerte verwendet werden), die beim Matching maximal verwendet werden können. Die Anzahl kann bei Matching aber kleiner sein als dieser Wert. Falls das Template durch das Verkleinern zu klein wird, reduziert sich die maximale Anzahl von Pyramidenstufen automatisch (ohne Fehlermeldung!).

Der Parameter GrayValues legt fest, ob die Originalgrauwerte (*'original'* bzw. *'normalized'*) oder die Kantenamplitude (*'gradient'* bzw. *'sobel'*) verwendet werden soll. Bei *'original'* wird die Summe der Differenzen als Merkmal verwendet. Dieses Verfahren ist abhängig von der Beleuchtung. Bei *'normalized'* wird die normierte Summe der Differenzen verwendet. Dieses Merkmal ist unabhängig von additiven Grauwertänderungen. Es ist etwas langsamer und nicht ganz so stabil. Wenn auch multiplikative Grauwertänderungen auftreten, sollte das Matching mittels normalisierter Kreuzkorrelation (normalized cross correlation, NCC) verwendet werden (create_ncc_model). Falls die Grauwerte sich nicht ändern ist *'original'* vorzuziehen. Die Kantenamplitude ist eine weitere Methode unabhängig von der Beleuchtung zu sein. Der Nachteil ist die höhere Laufzeit und eine stärkere Abhängigkeit gegenüber Formänderungen. Der Modus *'gradient'* ist etwas schneller aber auch empfindlicher gegen Rauschen. Der maximale Fehler beim Matching muss bei der Kantenamplitude typischerweise größer gewählt werden. Als Alternative zu den Gradientenverfahren kann der Operator set_offset_template verwendet werden, falls die Beleuchtungsänderung bekannt ist.

Der Parameter Optimize legt fest ob das Template für eine schnellere Verarbeitung zusätzlich optimiert wird. Dies verlangsamt die Erzeugung des Templates, verkürzt aber das Matching. Zusätzlich wird durch die Optimierung das Matching stabiler, d.h. der Verlust eines korrekten Matches wird unwahrscheinlicher.

Als Referenzwert für das Matching wird der Schwerpunkt verwendet. D.h. wenn man das Template auf das Originalbild anwendet, wird der Schwerpunkt zurückgegeben. Diese Voreinstellung kann mit set_reference_template angepasst werden.

Im sub-Pixel Modus wird eine zusätzliche Positionskorrektur verwendet: Das Template wird auf die Originaldaten angewandt und die hierbei entstehende Abweichung der gefundenen Position vom Schwerpunkt wird als Korrekturvektor mit verwendet. Dies ist für Muster, die in einem kontrastreichen Kontext liegen von Bedeutung. Bei den meisten Mustern ist dieser Korrekturvektor Null.

Vor der Verwendung kann das Template, das bildformatunabhängig gespeichert ist, noch mit adapt_template auf die Größe eines konkreten Bildes angepasst werden.

Parameter

▷ **Template** (input_object) ..... singlechannelimage ⇝ *object* : byte
Eingabebild, dessen Definitionsbereich für das Pattern Matching aufbereitet wird.

▷ **FirstError** (input_control) ..... integer ⇝ *integer*
Noch nicht verwendet.
Defaultwert: 255
Werteliste: FirstError ∈ {255}

▷ **NumLevel** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 4
Werteliste: NumLevel ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Optimize** (input_control) ..... string ⇝ *string*
Art der Optimierung.
Defaultwert: 'sort'
Werteliste: Optimize ∈ {'none', 'sort'}

▷ **GrayValues** (input_control) ..... string ⇝ *string*
Art der Grauwerte.
Defaultwert: 'original'
Werteliste: GrayValues ∈ {'original', 'normalized', 'gradient', 'sobel'}

▷ **TemplateID** (output_control) ..... template ⇝ *handle*
Nummer des Templates.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_template den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

draw_region, reduce_domain, threshold

Nachfolger

adapt_template, set_reference_template, clear_template, write_template, set_offset_template, best_match, best_match_mg, fast_match, fast_match_mg

Alternativen

create_ncc_model, create_template_rot, read_template

Modul

Matching

create_template_rot (Template : : NumLevel, AngleStart, AngleExtend, AngleStep, Optimize, GrayValues : TemplateID)

Aufbereiten eines Musters für Template Matching mit Rotation.

create_template_rot ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

create_template_rot bereitet ein Muster, das als Bild übergeben wird, für das Template Matching vor. Als Erweiterung gegenüber create_template kann das Matching später mit rotierten Mustern ausgeführt werden. Die Parameter AngleStart und AngleExtend legen die maximale Rotation des Musters fest: AngleStart gibt die größte Rotation gegen den Uhrzeiger an und AngleExtend die größte Rotation im Uhrzeigersinn relativ zu diesem Winkel. AngleExtend muss folglich immer kleiner als 2π sein. Mit dem Parameter AngleStep wird die maximale Winkelauflösung (auf der untersten Pyramidenebene) festgelegt.

Es ist zu beachten, dass alle möglichen Rotationen bei der Erzeugung des Templates berechnet werden um Laufzeit bei Matching zu sparen. Dies führt zu einer entsprechend hohen Laufzeit von create_template_rot und einem hohen Speicherbedarf des erzeugten Templates. Der Speicherbedarf hängt von AngleExtend und AngleStep ab. Die Anzahl der Pyramidenstufen kann dagegen vernachlässigt werden. Falls A die Fläche von Template ist, dann ist der Speicherbedarf M in Byte etwa:

$$M = \frac{A * 12 * \texttt{AngleExtend}}{\texttt{AngleStep}}$$

Dem Template wird nach der Transformation eine Nummer zugewiesen (TemplateID), die bei der weiteren Verarbeitung verwendet wird.

Die Beschreibung der weiteren Parameter ist bei create_template zu finden.

Achtung

Es ist zu beachten, dass für jede Rotation Muster angelegt werden. Dies erhöht bei einer feinen Auflösung den Speicherbedarf entsprechend.

Parameter

- ▷ **Template** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
  Eingabebild, dessen Definitionsbereich für das Pattern Matching aufbereitet wird.
- ▷ **NumLevel** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl von Pyramidenebenen.
  Defaultwert: 4
  Werteliste: `NumLevel` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
  Kleinster auftretende Rotation des Musters.
  Defaultwert: -0.39
  Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **AngleExtend** (input_control) . . . . . angle.rad ⇝ *real*
  Maximale positive Abweichung von `AngleStart`.
  Defaultwert: 0.79
  Wertevorschläge: `AngleExtend` ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
  Restriktion: AngleExtend > 0
- ▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real*
  Schrittweite (Winkelgenauigkeit) des Matchings.
  Defaultwert: 0.0982
  Wertevorschläge: `AngleStep` ∈ {0.3927, 0.1963, 0.0982, 0.0491, 0.0245}
  Restriktion: AngleStep > 0
- ▷ **Optimize** (input_control) . . . . . string ⇝ *string*
  Art der Optimierung.
  Defaultwert: 'sort'
  Werteliste: `Optimize` ∈ {'none', 'sort'}
- ▷ **GrayValues** (input_control) . . . . . string ⇝ *string*
  Art der Grauwerte.
  Defaultwert: 'original'
  Werteliste: `GrayValues` ∈ {'original', 'normalized', 'gradient', 'sobel'}
- ▷ **TemplateID** (output_control) . . . . . template ⇝ *handle*
  Nummer des Templates.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_template_rot` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`draw_region`, `reduce_domain`, `threshold`

Nachfolger

`best_match_rot`, `best_match_rot_mg`, `adapt_template`, `set_reference_template`, `clear_template`, `set_offset_template`, `write_template`

Alternativen

`create_ncc_model`, `create_template`

Modul

Matching

deserialize_template (: : SerializedItemHandle : TemplateID)

Deserialisieren eines serialisierten Templates.

deserialize_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

deserialize_template deserialisiert ein Template, welches mit serialize_template serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Template wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Template mit dem Handle TemplateID gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **TemplateID** (output_control) ..... template ⇝ *handle*
Handle des Templates.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_template den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_template

Nachfolger

adapt_template, set_reference_template, set_offset_template, best_match, fast_match, best_match_rot

Modul

Matching

fast_match (Image : Matches : TemplateID, MaxError :)

Suche nach allen guten Matches eines Templates und eines Bilds.

fast_match ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

fast_match führt ein Matching zwischen dem Template von TemplateID und Image durch. Dabei wird das Template so über die Punkte von Image geschoben, dass es immer vollständig innerhalb von Image liegt. Das Matching-Kriterium („displaced frame difference“) ist wie folgt definiert:

$$error[row, col] = \frac{\sum_{u,v} |\texttt{Image}[row-u, col-v] - \texttt{TemplateID}[u,v]|}{area(TemplateID)}$$

Die Laufzeit des Verfahrens hängt von der Größe des Definitionsbereiches von Image ab. Es ist daher wichtig, den Definitionsbereich möglichst einzugrenzen, d.h. den Operator nur in einer möglichst eng umrissenen „region of interest“ anzuwenden. Der Parameter MaxError legt den maximalen Fehler fest, den die gesuchten Positionen haben dürfen. Um so kleiner dieser Wert ist, um so schneller läuft das Verfahren.

Alle Punkte, bei denen der Fehler des Matching kleiner als MaxError ist, werden in der Ausgaberegion Matches übergeben.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object* : byte
Eingabebild, in dem das Pattern gefunden werden soll.

▷ **Matches** (output_object) . . . . . region(-array) ⇝ *object*
Alle Punkte, bei denen der Fehler unter der Schwelle liegt.

▷ **TemplateID** (input_control) . . . . . template ⇝ *handle*
Nummer des Templates.

▷ **MaxError** (input_control) . . . . . real ⇝ *real*
Maximale mittlere Differenz der Grauwerte.
Defaultwert: 20.0
Wertevorschläge: MaxError ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 17.0, 20.0, 30.0}
Typischer Wertebereich: 0 ≤ MaxError ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert fast_match den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

create_template, read_template, adapt_template, draw_region, draw_rectangle1, reduce_domain

Nachfolger

connection, best_match

Alternativen

best_match, best_match_mg, fast_match_mg, exhaustive_match, exhaustive_match_mg

Modul

Matching

fast_match_mg (Image : Matches : TemplateID, MaxError, NumLevel :)

Suche nach allen guten Grauwert-Matches in einer Pyramide.

fast_match_mg ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

fast_match_mg führt analog zu fast_match ein Matching zwischen dem Template von TemplateID und Image durch. Im Gegensatz zu fast_match beginnt die Suche nach guten Matches aber in verkleinerten Bildern (Pyramide). Die Anzahl der Ebenen der Pyramide werden mit NumLevel festgelegt. Dabei bedeutet der Wert 1, dass keine Pyramide verwendet wird. fast_match_mg ist in diesem Fall identisch mit fast_match. Bei dem Wert 2 beginnt die Suche bei dem Bild mit halber Kantenlänge. Für alle Punkte, die in dem verkleinerten Bild einen geringen Fehler hatten (kleiner als MaxError) wird die Suche dann bei den entsprechenden Positionen im Originalbild (Image) verfeinert. Optional kann bei dem Parameter NumLevel ein zweiter Wert übergeben werden. Dieser legt die Nummer der Ebene fest bei der mit dem Matching aufgehört werden soll. Es wird dann die bis dahin gefunden Region auf die Originalebene vergrößert. Die Werte sind größer oder gleich Null. Dabei entspricht Null der untersten Ebene und führt somit zu keiner Veränderung des Verfahrens. Bei der Wert 1 bricht das Matching eine Ebene oberhalb ab. Hierdurch wird Rechenzeit eingespart. Dies ist insbesondere dann sinnvoll, wenn es nicht auf eine genaue Lokalisation ankommt.

Die Laufzeit hängt dabei sehr von MaxError ab: um so größer der Schwellenwert ist um so länger dauert das Matching, da mehr Punkt weiter untersucht werden müssen. Falls MaxError jedoch zu klein ist, wird das gesuchte Muster nicht gefunden. Der Wert ist also für jede Anwendung durch Tests zu optimieren.

NumLevel gibt die Anzahl von Ebenen der Pyramide, einschließlich dem Ausgangsbild an.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild, in dem das Pattern gefunden werden soll.
- ▷ **Matches** (output_object) ..... region(-array) ⇝ *object*
  Alle Punkte, bei denen der Fehler unter der Schwelle liegt.
- ▷ **TemplateID** (input_control) ..... template ⇝ *handle*
  Nummer des Templates.
- ▷ **MaxError** (input_control) ..... real ⇝ *real*
  Maximale mittlere Differenz der Grauwerte.
  Defaultwert: 30.0
  Wertevorschläge: MaxError ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 17.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0}
  Typischer Wertebereich: $0 \leq$ MaxError ≤ 255
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 3
- ▷ **NumLevel** (input_control) ..... integer(-array) ⇝ *integer*
  Anzahl Ebenen in der Pyramide.
  Defaultwert: 3
  Wertevorschläge: NumLevel ∈ {1, 2, 3, 4, 5, 6, 7, 8}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert fast_match_mg den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

create_template, read_template, adapt_template, draw_region, draw_rectangle1, reduce_domain

Alternativen

best_match, best_match_mg, fast_match, exhaustive_match, exhaustive_match_mg

Modul

Matching

read_template (: : FileName : TemplateID)

Einlesen eines Templates von Datei.

read_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

read_template liest ein Matching Template von Datei das mit write_template geschrieben wurde. Die Dateiendung für ein Template ist in HALCON 'gvt'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: .gvt

▷ **TemplateID** (output_control) . . . . . template ⇝ *handle*
Nummer des Templates.

Ergebnis

Ist der Dateiname korrekt, dann liefert read_template den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

adapt_template, set_reference_template, set_offset_template, best_match, fast_match, best_match_rot

Modul

Matching

serialize_template (: : TemplateID : SerializedItemHandle)

Serialisieren eines Templates.

serialize_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

serialize_template serialisiert die Daten eines Templates (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_template in eine Datei geschrieben werden. Das Template wird in dem Handle TemplateID übergeben. Das serialisierte Template wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_template wieder deserialisiert werden.

Parameter

▷ **TemplateID** (input_control) . . . . . template ⇝ *handle*
Handle des OCV-Tools.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Templates.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_template den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_template, create_template_rot

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_template

Modul

Matching

set_offset_template (: : TemplateID, GrayOffset :)

Grauwertoffset für Template.

set_offset_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

set_offset_template addiert einen Offset zu den Grauwerten des Templates um eine Grauwertänderung im Bild auszugleichen. Der Parameter GrayOffset gibt die Änderung gegenüber den Grauwerten des ursprünglichen Pattern an, wie sie bei dem Aufruf von create_template gesetzt waren. Der Wert von GrayOffset bezieht sich auf das Bild in dem das Matching ausgeführt wird: Bei einem helleren Bild ist ein positiver Wert, bei einem dunkleren Bild ein negativer Wert zu übergeben. set_offset_template ist bei jeder Grauwertänderung aufzurufen. Die Grauwerte können z.B. in einem Referenzbereich des Bildes mit intensity oder min_max_gray bestimmt werden.

Parameter

▷ **TemplateID** (input_control) . . . . . template ⇝ *handle*
Nummer des Templates.

▷ **GrayOffset** (input_control) . . . . . number ⇝ *integer*
Offset der Grauwert.
Defaultwert: 0
Wertevorschläge: GrayOffset ∈ {-10, -5, -2, -1, 0, 1, 2, 5, 10}
Typischer Wertebereich: -255 ≤ GrayOffset ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_offset_template den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TemplateID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_template, adapt_template, read_template

Nachfolger

best_match, best_match_mg, best_match_rot, fast_match, fast_match_mg

Modul

Matching

set_reference_template (: : TemplateID, Row, Column :)

Referenzposition für ein Matching-Template setzen.

set_reference_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

set_reference_template definiert die Referenzposition eines Templates neu. Als Default nach create_template oder create_template_rot wird der Ursprung des Bild $(0, 0)$ als Referenz verwendet, d.h. bei einer Verschiebung von Null wird der Schwerpunkt des Templates als Ergebnis geliefert. Mit set_reference_template kann nun ein beliebiger Referenzpunkt definiert werden. Verwendet man z.B. den Schwerpunkt als Referenz, dann wird bei einer Verschiebung von Null der Vektor $(0, 0)$ geliefert.

Parameter

- ▷ **TemplateID** (input_control) ..... template ⇝ *handle*
  Nummer des Templates.
- ▷ **Row** (input_control) ..... point.y ⇝ *real*
  Referenzposition des Templates (Zeile).
- ▷ **Column** (input_control) ..... point.x ⇝ *real*
  Referenzposition des Templates (Spalte).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_reference_template den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TemplateID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_template, create_template_rot, read_template, adapt_template

Nachfolger

best_match, best_match_mg, best_match_rot, fast_match, fast_match_mg

Modul

Matching

write_template (: : TemplateID, FileName :)

Schreiben eines Templates auf Datei.

write_template ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Form-basierten oder NCC-basierten Operatoren verwenden.

write_template schreibt ein Matching Template auf Datei das mit read_template wieder gelesen werden kann. Die Dateiendung für ein Template ist in HALCON 'gvt'.

Parameter

▷ **TemplateID** (input_control) ..... template ⇝ *handle*
Nummer des Templates.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei.
Dateiendung: .gvt

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert write_template den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_template, create_template_rot

Modul

Matching

16.8 Morphologie

closing_golay (Region : RegionClosing : GolayElement, Rotation :)

Ausführen einer Closing-Operation mit Golay-Alphabet als strukturierendes Element.

closing_golay ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

closing_golay ist als Hintereinanderschaltung von Minkowski-Addition und Minkowski-Subtraktion definiert. Dabei wird zuerst die Minkowski-Addition der Eingaberegion (Region) mit dem durch GolayElement und Rotation ausgewählten strukturierenden Element des Golay-Alphabets durchgeführt. Mit dem Ergebnis der Minkowski-Addition und dem um 180° gedrehten strukturierenden Element, wird dann eine Minkowski-Subtraktion durchgeführt.

Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (Rotation) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Durch closing_golay werden Löcher, die kleiner sind als das strukturierende Element, geschlossen und die Regionenränder geglättet. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt.

Achtung

Es ist zu beachten, dass nicht bei jedem Golayelement alle Werte von Rotation zulässig sind. Bei einigen Werten für Rotation entsteht die identische Abbildung.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen,die verarbeitet werden sollen.

▷ **RegionClosing** (output_object) ..... region(-array) ⇝ *object*
Regionen, auf die die Closing-Operation angewandt wurde.

▷ **GolayElement** (input_control) ..... string ⇝ *string*
Strukturierendes Element aus dem Golay-Alphabet.
Defaultwert: 'h'
Werteliste: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

▷ **Rotation** (input_control) . . . . . integer ⇝ *integer*
Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
Defaultwert: 0
Werteliste: `Rotation` ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `closing_golay` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`closing`

Siehe auch

`erosion_golay`, `dilation_golay`, `opening_golay`, `hit_or_miss_golay`, `thinning_golay`, `thickening_golay`, `golay_elements`

Modul

Foundation

dilation_golay (Region : RegionDilation : GolayElement, Iterations, Rotation :)

Ausdehnen von Regionen (Golay).

`dilation_golay` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`dilation_golay` berechnet die Dilatation der Eingaberegionen mit einem ausgewählten strukturierenden Element aus dem Golay-Alphabet, das durch `GolayElement` bestimmt ist. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer `Rotation` gibt dabei an, welche Rotation, bzw. ob die Vordergrund- (gerade) oder die Hintergrundvariante (ungerade Werte) des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt. Das strukturierende Element wird dabei über die zu verarbeitende Region (`Region`) geschoben. Für alle Positionen des strukturierenden Elementes, an denen sie sich mit dem Objekt schneidet, wird der Bezugspunkt (relativ zum strukturierenden

Element) in die Ausgaberegion aufgenommen. Es wird also die Vereinigungsmenge über alle Verschiebungen des strukturierenden Elementes über die Region gebildet.

Der Parameter `Iterations` bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die Iteration n wird die Ergebnisregion der Iteration $(n-1)$ verwendet.

Achtung

Es ist zu beachten, dass nicht bei jedem Golayelement alle Werte von `Rotation` zulässig sind. Bei einigen Werten für `Rotation` entsteht die identische Abbildung.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **`RegionDilation`** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis der Dilation-Operation.
- ▷ **`GolayElement`** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert: 'h'
  Werteliste: `GolayElement` ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert: 1
  Wertevorschläge: `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich: $1 \leq$ Iterations (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **`Rotation`** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
  Defaultwert: 0
  Werteliste: `Rotation` ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(3 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation_golay` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`dilation1`, `dilation2`, `dilation_seq`

Siehe auch

erosion_golay, opening_golay, closing_golay, hit_or_miss_golay, thinning_golay, thickening_golay, golay_elements

Modul

Foundation

dilation_seq (Region : RegionDilation : GolayElement, Iterations :)

Ausdehnen von Regionen (sequentiell).

dilation_seq ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

dilation_seq berechnet die sequentielle Dilatation der Eingaberegionen mit dem durch GolayElement ausgewählten strukturierenden Element aus dem Golay-Alphabet. Dazu führt dilation_seq den Operator dilation_golay mit allen Rotationen des strukturierenden Elements so oft durch, wie der Parameter Iterations angibt. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'd', 'c', 'f', 'h', 'k'.

Für die Skelettbildung werden meistens die Elemente 'l' und 'm' verwendet. Es werden nur die „Vordergrund Elemente" des Golayalphabets (gerade Rotationsnummern) verwendet. Die Elemente 'i' und 'e' erzeugen hier die identische Abbildung, d.h. sie verändern die Eingaberegion nicht. Die Elemente 'l', 'm' und 'f2' sind im Vordergrund identisch. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionDilation** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis der Dilation-Operation.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert: 'h'
  Werteliste: GolayElement ∈ {'l', 'd', 'c', 'f', 'h', 'k'}
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert: 1
  Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich: 1 ≤ Iterations (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\texttt{Iterations} \cdot 20 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion dilation_seq den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

dilation1, dilation2, dilation_golay

Siehe auch

erosion_seq, hit_or_miss_seq, thinning_seq

Modul

Foundation

erosion_golay (Region : RegionErosion : GolayElement, Iterations, Rotation :)

Verdünnen von Regionen (Golay).

erosion_golay ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

Dieser Operator berechnet die Erosion der Eingaberegionen mit dem strukturierenden Element aus dem Golay-Alphabet, das durch GolayElement ausgewählt wird. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (Rotation) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt. erosion_golay ist eine Regionenoperation. Das strukturierende Element wird dabei über die zu verarbeitende Region Region geschoben. Für alle Positionen des strukturierenden Elementes, an denen sie sich mit dem Objekt völlig deckt, wird der Bezugspunkt relativ zum strukturierenden Element gesetzt. Es wird also die Schnittmenge über alle Verschiebungen des strukturierenden Elementes über die Region gebildet.

Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet.

Achtung

Es ist zu beachten, dass nicht bei jedem Golayelement alle Werte von Rotation möglich sind. Bei einigen Werten für Rotation entsteht die identische Abbildung.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **RegionErosion** (output_object) ..... region(-array) ⇝ *object*
Das Ergebnis der Erosion.

▷ **GolayElement** (input_control) ..... string ⇝ *string*
Strukturierendes Element aus dem Golay-Alphabet.
Defaultwert: 'h'
Werteliste: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert: 1
Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich: 1 ≤ Iterations (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Rotation** (input_control) . . . . . integer ⇝ *integer*
Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
Defaultwert: 0
Werteliste: Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(3 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion erosion_golay den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, watersheds, class_ndim_norm

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

erosion_seq, erosion1, erosion2

Siehe auch

dilation_golay, opening_golay, closing_golay, hit_or_miss_golay, thinning_golay, thickening_golay, golay_elements

Modul

Foundation

erosion_seq (Region : RegionErosion : GolayElement, Iterations :)

Verdünnen von Regionen (sequentiell).

erosion_seq ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

erosion_seq berechnet die sequentielle Erosion der Eingaberegionen mit dem durch GolayElement ausgewählten strukturierenden Element aus dem Golay-Alphabet. Dazu führt erosion_seq den Operator erosion_golay mit allen Rotationen des strukturierenden Elements so oft durch, wie der Parameter Iterations angibt. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten

'l', 'd', 'c', 'f', 'h', 'k'.

Es werden nur die „Vordergrund Elemente“ des Golayalphabets (gerade Rotationsnummern) verwendet. Die Elemente 'i' und 'e' erzeugen hier die identische Abbildung, d.h. sie verändern die Eingaberegion nicht. Die Elemente 'l', 'm' und 'f2' sind im Vordergrund identisch. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt. Für die Skelettbildung werden meistens die Elemente 'l' und 'm' verwendet.

Parameter

▷ **Region** (input_object) . . . . . region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **RegionErosion** (output_object) . . . . . region(-array) ⇝ *object*
Das Ergebnis der Erosion.

▷ **GolayElement** (input_control) . . . . . string ⇝ *string*
Strukturierendes Element aus dem Golay-Alphabet.
Defaultwert: 'h'
Werteliste: `GolayElement` ∈ {'l', 'd', 'c', 'f', 'h', 'k'}

▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert: 1
Wertevorschläge: `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich: 1 ≤ Iterations (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\texttt{Iterations} \cdot 20 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `erosion_seq` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`

Nachfolger

`connection`, `reduce_domain`, `select_shape`, `area_center`

Alternativen

`erosion_golay`, `erosion1`, `erosion2`

Siehe auch

`dilation_seq`, `hit_or_miss_seq`, `thinning_seq`

Modul

Foundation

fitting (Region, StructElements : RegionFitted : :)

Ausführen eines Closings nach einem Opening mit mehreren strukturierenden Elementen.

`fitting` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

fitting führt die beiden Grundoperationen opening und closing nacheinander aus. Die acht strukturierenden Elemente, die typischerweise zum Einsatz kommen, können mit der Funktion gen_struct_elements erzeugt werden. Jedoch ist auch der Einsatz einzelner, eigens erzeugter Elemente sinnvoll. Sei(en) R die Eingaberegion(en) und M_i bezeichne die strukturierenden Elemente. Sei weiterhin P das Zwischenergebnis nach dem opening und Q das Endergebnis, dann lässt sich die Funktion formal folgendermaßen schreiben:

$$P = \bigcup_{i=1}^{n} (R \circ M_i)$$
$$Q = \bigcap_{i=1}^{n} (P \bullet M_i)$$

Regionen, die größer sind als die strukturierenden Elemente bleiben erhalten, während kleine Löcher beim Closing geschlossen werden.

Parameter

▷ **Region** (input_object) . . . . . region(-array) ⇝ *object*
Regionen die bearbeitet werden.

▷ **StructElements** (input_object) . . . . . region(-array) ⇝ *object*
Strukturierende Elemente.

▷ **RegionFitted** (output_object) . . . . . region(-array) ⇝ *object*
Regionen, die strukturierenden Elemente enthalten.

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion fitting den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_struct_elements, gen_region_points

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

opening, closing, connection, select_shape

Modul

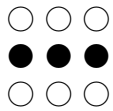
Foundation

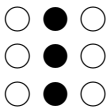
gen_struct_elements (: StructElements : Type, Row, Column :)

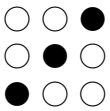
Erzeugen von strukturierenden Elementen.

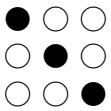
gen_struct_elements ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

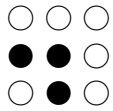
gen_struct_elements dient zur Erzeugung von acht strukturierenden Elementen, die üblicherweise für die Funktion fitting Verwendung finden. Die Standardbelegung *'noise'* für den Parameter Type erzeugt Elemente, die insbesonders für die Eliminierung von Rauschen geeignet sind.

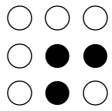

M_1

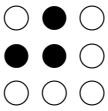

M_2

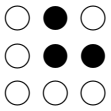

M_3


M_4


M_5


M_6


M_7


M_8

Parameter

▷ **StructElements** (output_object) ..... region(-array) ⇝ *object*
Erzeugte strukturierende Elemente.

▷ **Type** (input_control) ..... string ⇝ *string*
Art der strukturierende Elemente.
Defaultwert: 'noise'
Werteliste: Type ∈ {'noise'}

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Bezugspunktes.
Defaultwert: 1
Wertevorschläge: Row ∈ {0, 1, 10, 50, 100, 200, 300, 400}
(lin)

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Bezugspunktes.
Defaultwert: 1
Wertevorschläge: Column ∈ {0, 1, 10, 50, 100, 200, 300, 400}
(lin)

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion gen_struct_elements den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

fitting, hit_or_miss, opening, closing, erosion2, dilation2

Siehe auch

golay_elements

Modul

Foundation

golay_elements (: StructElement1, StructElement2 : GolayElement, Rotation, Row, Column :)

Erzeugen der strukturierenden Elemente des Golay-Alphabets.

`golay_elements` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`golay_elements` erzeugt die strukturierenden Elemente aus dem Golay-Alphabet. Der Parameter `GolayElement` legt den Namen, `Rotation` die Rotation fest. Die strukturierenden Elemente sind für `hit_or_miss` ausgelegt: In `StructElement1` wird das strukturierende Element für den Vordergrund in `StructElement2` das für den Hintergrund übergeben. `Row` und `Column` legen den Bezugspunkt fest.

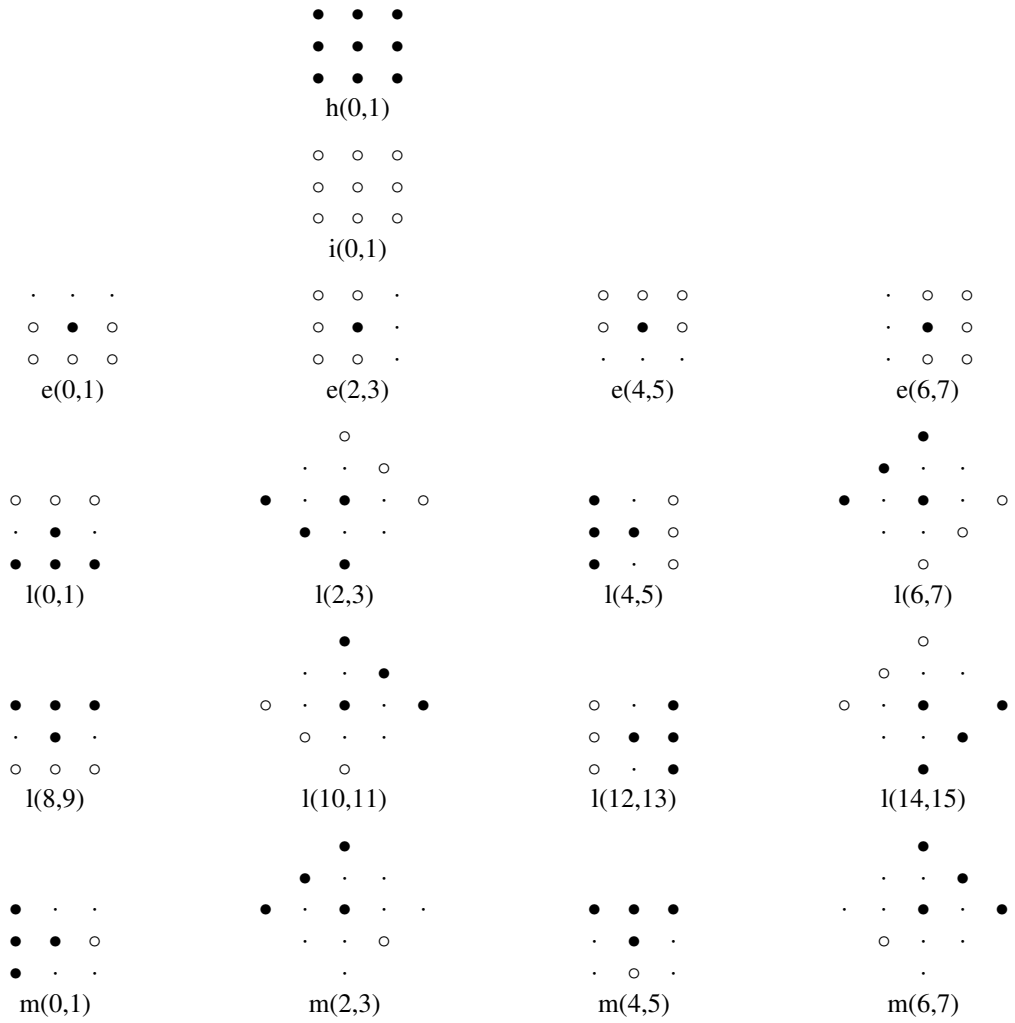
Die Rotationen sind von 0 bis 15 durchnummeriert. Das heißt aber nicht, dass es 16 verschiedene Rotationen gibt: Die geraden Werte bezeichnen die Rotationen der Vordergrundelemente, die ungeraden die der Hintergrundelemente.

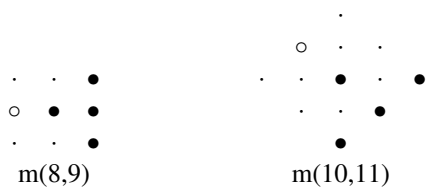
Bei `golay_elements` werden nur die geraden Werte angegeben und legen den Wert für `StructElement1` fest. Die nächstgrößere (ungerade) Zahl wird intern berechnet und bestimmt den Wert für `StructElement2`. Für die Elemente 'h' und 'i' gibt es keine Rotationen, folglich sind nur 0 und 1 als „Rotationen" möglich (bei `golay_elements` folglich nur 0). Das Element 'e' hat nur 4 Rotationen, folglich liegen die gültigen Werte zwischen 0 und 7 (bei `golay_elements` ergeben sich die Werte 0, 2, 4 und 6).

Die Grafiken zeigen die Elemente des Golay-Alphabets mit allen realisierten Rotationen. Die verwendeten Zeichen haben folgende Bedeutung:

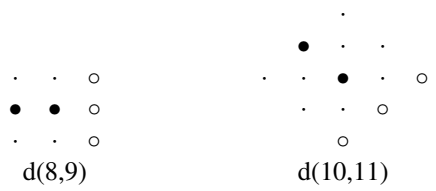
- ● Vordergrundpixel
- ○ Hintergrundpixel
- · Pixel mit beliebigem Wert

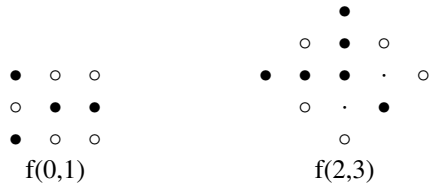
Die Namen der Elemente und ihre Rotationsnummern stehen jeweils unter der Zeichnung. Die Elemente mit gerader Nummer enthalten die Vordergrundpixel, die Elemente mit ungerader Rotationsnummer die Hintergrundpixel.

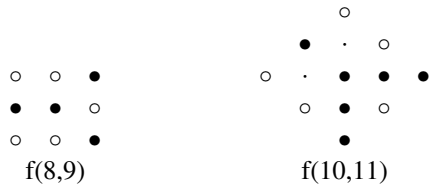



m(8,9) m(10,11) m(12,13) m(14,15)

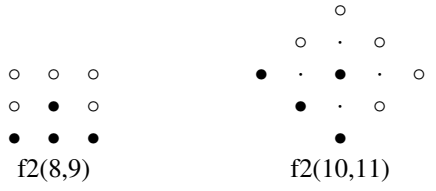

d(0,1) d(2,3) d(4,5) d(6,7)

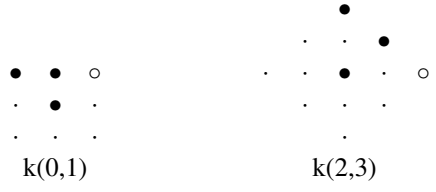

d(8,9) d(10,11) d(12,13) d(14,15)

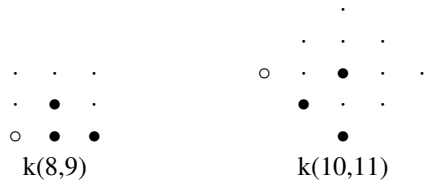

f(0,1) f(2,3) f(4,5) f(6,7)

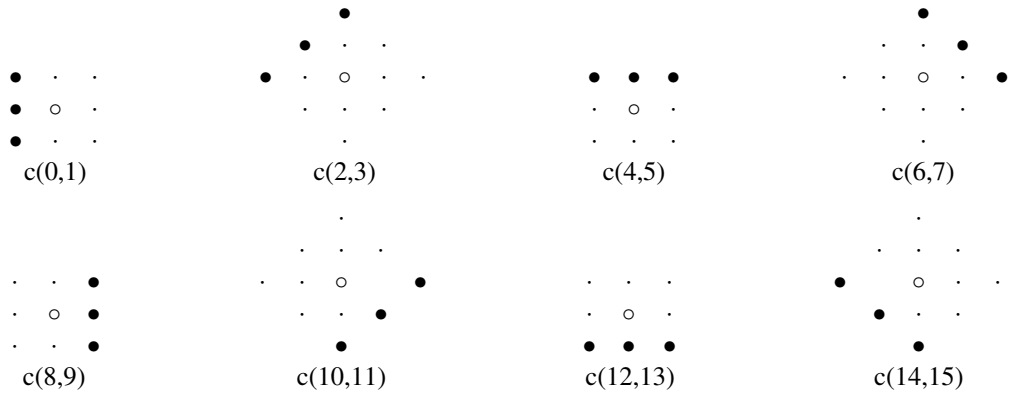

f(8,9) f(10,11) f(12,13) f(14,15)


f2(0,1) f2(2,3) f2(4,5) f2(6,7)


f2(8,9) f2(10,11) f2(12,13) f2(14,15)


k(0,1) k(2,3) k(4,5) k(6,7)


k(8,9) k(10,11) k(12,13) k(14,15)



Parameter

▷ **StructElement1** (output_object) . . . . . region ⇝ *object*
Strukturierendes Element für den Vordergrund.

▷ **StructElement2** (output_object) . . . . . region ⇝ *object*
Strukturierendes Element für den Hintergrund.

▷ **GolayElement** (input_control) . . . . . string ⇝ *string*
Name des strukturierenden Elementes.
Defaultwert: 'l'
Werteliste: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

▷ **Rotation** (input_control) . . . . . integer ⇝ *integer*
Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
Defaultwert: 0
Werteliste: Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14}

▷ **Row** (input_control) . . . . . point.y ⇝ *integer*
Zeile des Bezugspunktes.
Defaultwert: 16
Wertevorschläge: Row ∈ {0, 16, 32, 128, 256}
Typischer Wertebereich: 0 ≤ Row ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Column** (input_control) . . . . . point.x ⇝ *integer*
Spalte des Bezugspunktes.
Defaultwert: 16
Wertevorschläge: Column ∈ {0, 16, 32, 128, 256}
Typischer Wertebereich: 0 ≤ Column ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Ergebnis

Bei korrekter Parametrisierung liefert der Operator golay_elements den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

hit_or_miss

Alternativen

gen_region_points, gen_struct_elements, gen_region_polygon_filled

Siehe auch

dilation_golay, erosion_golay, opening_golay, closing_golay, hit_or_miss_golay, thickening_golay

Literatur

J. Serra: „Image Analysis and Mathematical Morphology“. Volume I. Academic Press, 1982

Modul

Foundation

hit_or_miss_golay (Region : RegionHitMiss : GolayElement, Rotation :)

Ausführen der Hit-or-Miss-Operation für Regionen (Golay).

hit_or_miss_golay ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

hit_or_miss_golay berechnet die Hit-or-Miss-Transformation der Eingaberegionen (abgestützt auf das Golay-Alphabet). Dazu wird zunächst eine Erosion der Regionen mit dem durch GolayElement ausgewählten strukturierenden Element des Golay-Alphabets durchgeführt. Im zweiten Schritt folgt eine Erosion der Komplemente der Regionen mit der Hintergrund-Maske des strukturierenden Elements. Aus den beiden Zwischenergebnissen wird die Schnittmenge gebildet. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e','i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (Rotation) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Hit-or-Miss-Transformation wählt genau die Punkte aus, deren Umgebung die Bedingungen des gewählten Golay-Elements erfüllt.

Achtung

Es ist zu beachten, dass nicht bei jedem Golayelement alle Rotationen möglich sind.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionHitMiss** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis der Hit-or-Miss-Operation.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert: 'h'
  Werteliste: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Rotation** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
  Defaultwert: 0
  Werteliste: Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion hit_or_miss_golay den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

hit_or_miss_seq, hit_or_miss

Siehe auch

erosion_golay, dilation_golay, opening_golay, closing_golay, thinning_golay, thickening_golay, golay_elements

Modul

Foundation

hit_or_miss_seq (Region : RegionHitMiss : GolayElement :)

Ausführen der Hit-or-Miss-Operation für Regionen (sequentiell).

hit_or_miss_seq ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

hit_or_miss_seq berechnet die Hit-or-Miss-Transformation einer Region mit allen rotierten Versionen des durch GolayElement gewählten strukturierenden Elements aus dem Golay-Alphabet. Als Ergebnis wird die Vereinigungsmenge der Zwischenergebnisse aus den einzelnen Rotationen geliefert. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionHitMiss** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis der Hit-or-Miss-Operation.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert: 'h'
  Werteliste: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

Komplexität

Sei F die Fläche einer Eingaberegion und R die Anzahl der Rotationen, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(R \cdot 6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion hit_or_miss_seq den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

hit_or_miss_golay, hit_or_miss

Siehe auch

thinning_seq, thickening_seq

Modul

Foundation

morph_hat (Region, StructElement : RegionMorphHat : :)

Vereinigen von bottom_hat und top_hat.

morph_hat ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

morph_hat vereinigt die Flächen, die bei der opening-Operation wegfallen mit den Flächen, die bei der closing-Operation hinzukommen. Dies entspricht also der Vereinigung von top_hat und bottom_hat. Die Position von StructElement ist ohne Bedeutung.

Das strukturierende Element (StructElement) kann mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Achtung

Die Regionen werden einzeln bearbeitet.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **StructElement** (input_object) ..... region ⇝ *object*
Strukturierendes Element (lageinvariant).

▷ **RegionMorphHat** (output_object) ..... region(-array) ⇝ *object*
Vereinigung von Top-Hat und Bottom-Hat.

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"

main()
{
  cout << "Reproduction of 'dilation_circle ()'" << endl;
  cout << "First = original image " << endl;
  cout << "Red   = after segmentation " << endl;
  cout << "Blue  = after erosion " << endl;

  HByteImage img("monkey");
  HWindow    w;
```

```
  HRegion      circ   = HRegion::GenCircle (10, 10, 1.5);
  HRegionArray regs   = (img >= 128).Connection();
  HRegionArray tophat = regs.TopHat (circ);
  HRegionArray bothat = regs.BottomHat (circ);
  HRegionArray unionX = tophat.Union2 (bothat);

                        img.Display (w);      w.Click ();
  w.SetColor ("red");   regs.Display (w);     w.Click ();
  w.SetColor ("blue");  tophat.Display (w);   w.Click ();
  w.SetColor ("green"); bothat.Display (w);   w.Click ();
  w.SetColor ("white"); unionX.Display (w);   w.Click ();

  return(0);
}
```

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `morph_hat` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_region_polygon_filled`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`top_hat`, `bottom_hat`, `union2`

Siehe auch

`opening`, `closing`

Modul

Foundation

morph_skeleton (Region : RegionSkeleton : :)

Berechnen des morphologischen Skeletts einer Region.

`morph_skeleton` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`morph_skeleton` berechnet das Skelett der Eingaberegionen (`Region`) mit Hilfe morphologischer Transformationen. Die Berechnung liefert ein nicht zusammenhängendes Skelett (Lücken in den Diagonalen) mit einer Breite von ein oder zwei Pixeln. Zur Berechnung wird das Golay-Element 'h' verwendet. Hieraus ergibt sich die Verwendung der 8-ter Nachbarschaft. Dies entspricht der Maximumsnorm.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **RegionSkeleton** (output_object) ..... region(-array) ⇝ *object*
Das Ergebnis der Skelettierung.

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `morph_skeleton` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Nachfolger

`skeleton`, `reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`skeleton`

Modul

Foundation

morph_skiz (Region : RegionSkiz : Iterations1, Iterations2 :)

Verdünnen von Regionen.

`morph_skiz` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`morph_skiz` führt zuerst ein sequentielles Thinning (`thinning_seq`) mit dem Element 'l' des Golay-Alphabets durch. Die Anzahl der Durchläufe bestimmt der Parameter `Iterations1`. Mit dem Ergebnis wird dann ein sequentielles Thinning mit dem Element 'e' des Golay-Alphabets durchgeführt. Die Anzahl der Durchläufe bestimmt hier der Parameter `Iterations2`. Durch die Skiz-Operation wird eine Art Skelett gebildet und die Äste dann verkürzt. Wird die Skiz-Operation auf die komplementäre Region angewandt, so werden die Regionen durch das entstandene Skelett voneinander getrennt.

Werden bei `Iterations1` oder `Iterations2` sehr große Werte oder 'maximal' übergeben, dann bricht die Verarbeitung ab, wenn keine Veränderung mehr auftritt.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **RegionSkiz** (output_object) ..... region(-array) ⇝ *object*
Das Ergebnis des Skiz-Operators.

▷ **Iterations1** (input_control) . . . . . integer ⇝ *integer* / string
Iterationen für das sequentielle Thinning mit dem Element 'l'des Golay-Alphabets.
Defaultwert: 100
Wertevorschläge: Iterations1 ∈ {'maximal', 0, 1, 2, 3, 5, 7, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200, 300, 400}
Typischer Wertebereich: 0 ≤ Iterations1 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Iterations2** (input_control) . . . . . integer ⇝ *integer* / string
Iterationen für das sequentielle Thinning mit dem Element 'e' des Golay-Alphabets.
Defaultwert: 1
Wertevorschläge: Iterations2 ∈ {'maximal', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich: 0 ≤ Iterations2 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O((\texttt{Iterations1} + \texttt{Iterations2}) \cdot 3 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion morph_skiz den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

pruning, reduce_domain, select_shape, area_center, connection, background_seg, complement

Alternativen

skeleton, thinning_seq, morph_skeleton, interjacent

Siehe auch

thinning, hit_or_miss_seq, difference

Modul

Foundation

opening_golay (Region : RegionOpening : GolayElement, Rotation :)

Auftrennen von Lücken mit einem Golay-Element.

opening_golay ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

opening_golay ist als Hintereinanderschaltung von Minkowski-Subtraktion und Minkowski-Addition. Dabei wird zuerst die Minkowski-Subtraktion der Eingaberegionen (Region) mit dem durch GolayElement und Rotation ausgewählten strukturierenden Element aus dem Golay-Alphabet durchgeführt. Mit dem Ergebnis der Minkowski-Subtraktion und dem um 180° gedrehten strukturierenden Element, wird dann eine Minkowski-Addition durchgeführt.

Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (Rotation) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt.

Durch opening_golay werden Regionen, die kleiner sind als das strukturierende Element, entfernt und die Objektränder geglättet.

Achtung

Es ist zu beachten, dass nicht bei jedem Golayelement alle Werte von Rotation zulässig sind. Bei einigen Werten für Rotation entsteht die identische Abbildung.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionOpening** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis des Opening-Operators.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert: 'h'
  Werteliste: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Rotation** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
  Defaultwert: 0
  Werteliste: Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion opening_golay den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

opening_seg, opening

Siehe auch

erosion_golay, dilation_golay, closing_golay, hit_or_miss_golay, thinning_golay, thickening_golay, golay_elements

Modul

Foundation

opening_seg (Region, StructElement : RegionOpening : :)

Trennen von sich überlappenden Regionen.

opening_seg ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

Die opening_seg-Operation ist als die Hintereinanderschaltung von erosion1, connection und dilation1 definiert (siehe Beispiel). Es wird dabei nur eine Iteration bei erosion1 und dilation1 ausgeführt.

opening_seg dient zum Trennen von sich überlappenden Regionen, deren Überlappungsbereich kleiner als StructElement ist. Es ist zu beachten, dass sich die Ergebnisregionen überlappen können ohne dabei zu verschmelzen (siehe expand_region). opening_seg verwendet als Bezugspunkt den Schwerpunkt des strukturierenden Elementes.

Das strukturierende Element StructElement kann mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element (lageinvariant).
- ▷ **RegionOpening** (output_object) ..... region-array ⇝ *object*
  Das Ergebnis des Opening-Operators.

Beispiel

```
* Simulation of opening_seg
* opening_seg(Region,StructElement,RegionOpening):
    erosion1(Region,StructElement,H1,1)
    connection(H1,H2)
    dilation1(H2,StructElement,RegionOpening,1)
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \sqrt{\sqrt{F1}}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion opening_seg den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_region_polygon_filled

Nachfolger

expand_region, reduce_domain, select_shape, area_center, connection

Alternativen

erosion1, connection, dilation1

Modul

Foundation

```
thickening ( Region,  StructElement1,
    StructElement2 : RegionThick : Row,  Column,  Iterations : )
```

Vergrößern einer Region um das Ergebnis von Hit-or-Miss.

thickening ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

thickening berechnet das Thickening der Eingaberegionen mit morphologischen Operationen. Der Operator führt dabei zuerst eine Hit-or-Miss-Transformation (vgl. hit_or_miss) durch und fügt die dabei gefundenen Punkte zur Eingaberegion hinzu. Der Parameter Iterations bestimmt die Anzahl der Iterationen, die durchgeführt werden.

Für die Wahl der strukturierenden Elemente StructElement1 und StructElement2 sowie für Row und Column gelten dieselben Empfehlungen wie bei hit_or_miss.

Die strukturierenden Elemente (StructElement1 und StructElement2) können z.B. mit dem Operator golay_elements erzeugt werden.

Achtung

Wenn der Bezugspunkt von StructElement1 zum Vordergrund gehört, ändert sich die Region nicht.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement1** (input_object) ..... region ⇝ *object*
  Strukturierendes Element für den Vordergrund
- ▷ **StructElement2** (input_object) ..... region ⇝ *object*
  Strukturierendes Element für den Hintergrund
- ▷ **RegionThick** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis des Thickening-Operators.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunkts.
  Defaultwert: 16
  Wertevorschläge: Row ∈ {0, 2, 4, 8, 16, 32, 128}
  Typischer Wertebereich: 0 ≤ Row ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spalte des Bezugspunkts.
Defaultwert: 16
Wertevorschläge: Column ∈ {0, 2, 4, 8, 16, 32, 128}
Typischer Wertebereich: 0 ≤ Column ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Anzahl der Iterationsschritte.
Defaultwert: 1
Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50, 70, 100, 200, 400}
Typischer Wertebereich: 1 ≤ Iterations (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, $F1$ die Fläche des strukturierenden Elementes 1 und $F2$ die Fläche der des strukturierenden Elementes 2, dann ist die Laufzeitkomplexität für ein Objekt:

$$O\left(\texttt{Iterations} \cdot \sqrt{F} \cdot \left(\sqrt{F1} + \sqrt{F2}\right)\right) \quad .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion thickening den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

golay_elements, threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

thickening_golay, thickening_seq

Siehe auch

hit_or_miss

Modul

Foundation

thickening_golay (Region : RegionThick : GolayElement, Rotation :)

Vergrößern einer Region um das Ergebnis von Hit-or-Miss (Golay).

thickening_golay ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

thickening_golay berechnet das Thickening der Eingaberegionen mit morphologischen Operationen (abgestützt auf das Golay-Alphabet). Der Operator führt dazu eine Hit-or-Miss-Transformation (vgl. hit_or_miss_golay) durch und fügt die dabei gefundenen Punkte zur Eingaberegion hinzu. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (Rotation) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind unter golay_elements beschrieben.

Achtung

Es ist zu beachten, dass nicht bei jedem Golayelement alle Rotationen möglich sind.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionThick** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis des Thickening-Operators.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert: 'h'
  Werteliste: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Rotation** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
  Defaultwert: 0
  Werteliste: Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion thickening_golay den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

thickening, thickening_seq

Siehe auch

erosion_golay, hit_or_miss_golay

Modul

Foundation

thickening_seq (Region : RegionThick : GolayElement, Iterations :)

Vergrößern einer Region um das Ergebnis von Hit-or-Miss (sequentiell).

thickening_seq ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

thickening_seq berechnet das sequentielle Thickening der Eingaberegionen mit dem durch GolayElement ausgewählten strukturierenden Element aus dem Golay-Alphabet. Dazu führt thickening_seq den Operator thickening_golay mit allen Rotationen des strukturierneden Elements so oft durch, wie der Parameter Iterations angibt. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt. Bei allen Elementen außer 'c' werden Vorder- und Hintergrundanteil vertauscht, damit sie Einfluss auf den äußeren Rand der Region nehmen. Mit dem Element 'c' kann eine konvexe Hülle erzeugt werden, wenn genügend Iterationen durchgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionThick** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis des Thickening-Operators.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert: 'h'
  Werteliste: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationsschritte.
  Defaultwert: 1
  Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50, 70, 100, 200}
  Typischer Wertebereich: 1 ≤ Iterations (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(Iterations \cdot 6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion thickening_seq den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

thickening_golay, thickening

Siehe auch

erosion_golay, thinning_seq

Modul

Foundation

thinning (Region, StructElement1, StructElement2 : RegionThin : Row, Column, Iterations :)

Verdünnen einer Region um das Ergebnis von Hit-or-Miss.

thinning ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

thinning berechnet die Verdünnung der Eingaberegionen mit morphologischen Operationen. Dazu führt der Operator eine Hit-or-Miss-Transformation (vgl. hit_or_miss) durch und entfernt die dabei gefundenen Punkte aus den Eingaberegionen. Der Parameter Iterations bestimmt die Anzahl der Iterationen, die durchgeführt werden.

Für die Wahl der strukturierende Elemente StructElement1 und StructElement2 sowie für Row und Column gelten dieselben Empfehlungen wie bei hit_or_miss.

Strukturierende Elemente (StructElement1, StructElement2) können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement1** (input_object) ..... region ⇝ *object*
  Strukturierendes Element für den Vordergrund
- ▷ **StructElement2** (input_object) ..... region ⇝ *object*
  Strukturierendes Element für den Hintergrund
- ▷ **RegionThin** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis des Thinning-Operators.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunkts.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ Row ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunkts.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ Column ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert: 1
  Wertevorschläge: Iterations $\in$ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich: $1 \leq$ Iterations (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, $F1$ die Fläche des strukturierenden Elementes 1 und $F2$ die Fläche der des strukturierenden Elementes 2, dann ist die Laufzeitkomplexität für ein Objekt:

$$O\left(\texttt{Iterations} \cdot \sqrt{F} \cdot \left(\sqrt{F1} + \sqrt{F2}\right)\right) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `thinning` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_region_polygon_filled`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`thinning_golay`, `thinning_seq`

Siehe auch

`hit_or_miss`

Modul

Foundation

`thinning_golay` (Region : RegionThin : GolayElement, Rotation :)

Verdünnen einer Region um das Ergebnis von Hit-or-Miss (Golay).

`thinning_golay` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`thinning_golay` berechnet die Verdünnung der Eingaberegionen mit morphologischen Operationen (abgestützt auf das Golay-Alphabet). Dazu führt der Operator eine Hit-or-Miss-Transformation (vgl. `hit_or_miss_golay`) aus und entfernt die dabei gefundenen Punkte aus den Eingaberegionen. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (`Rotation`) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt.

Achtung

Es ist zu beachten, dass nicht bei jedem Golayelement alle Rotationen möglich sind.

Parameter

▷ **Region** (input_object) . . . . . region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **RegionThin** (output_object) . . . . . region(-array) ⇝ *object*
Das Ergebnis des Thinning-Operators.

▷ **GolayElement** (input_control) . . . . . string ⇝ *string*
Strukturierendes Element aus dem Golay-Alphabet.
Defaultwert: 'h'
Werteliste: `GolayElement` ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

▷ **Rotation** (input_control) . . . . . integer ⇝ *integer*
Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
Defaultwert: 0
Werteliste: `Rotation` ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `thinning_golay` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`thinning_seq`, `thinning`

Siehe auch

`erosion_golay`, `hit_or_miss_golay`

Modul

Foundation

thinning_seq (Region : RegionThin : GolayElement, Iterations :)

Sequentielles Verdünnen einer Region um das Ergebnis von Hit-or-Miss.

thinning_seq ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`thinning_seq` berechnet das sequentielle Thinning der Eingaberegionen mit dem durch `GolayElement` ausgewählten strukturierenden Element aus dem Golay-Alphabet. Dazu führt `thinning_seq` den Operator `thinning_golay` für die Eingaberegionen mit allen Rotationen des strukturierenden Elements so oft durch, wie der Parameter `Iterations` angibt. Wird `Iterations` groß genug gewählt, erzeugt der Operator mit den

Elementen ’l’ bzw. ’m’ das Skelett der Regionen. Bei dem Element ’c’ werden Vorder- und Hintergrundanteil vertauscht, damit es den inneren Rand beeinflusst. Wird bei `Iterations` ein sehr großer Werte oder ’maximal’ übergeben, dann bricht die Verarbeitung ab, wenn keine Veränderung mehr auftritt. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

’l’ Skelett, ähnlich wie bei `skeleton`. Dieses strukturierende Element wird auch bei `morph_skiz` verwendet.

’m’ Ein Skelett mit vielen „Haaren“ und mehrfachen (parallelen) Ästen.

’d’ Skelett ohne mehrfach Äste aber mit vielen Lücken. Ähnlich zu `morph_skeleton`.

’c’ Gleichmäßige Erosion der Region.

’e’ Ein Pixel dicke Linien werden verkürzt. Dieses strukturierende Element wird auch bei `morph_skiz` verwendet.

’i’ Isolierte Punkte werden entfernt. (Nur `Iterations` = 1 sinnvoll).

’f’ Verzweigungspunkte (y-junctions) werden eliminiert. (Nur `Iterations` = 1 sinnvoll).

’f2’ Es werden ein Pixel lange Äste und Ecken entfernt. (Nur `Iterations` = 1 sinnvoll).

’h’ Eine Art von innerem Rand, der aber dicker als bei `boundary` ist, wird erzeugt, (Nur `Iterations` = 1 sinnvoll).

’k’ Verzweigungspunkte werden eliminiert und dabei auch neue erzeugt.

Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt.

Parameter

▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **`RegionThin`** (output_object) ..... region(-array) ⇝ *object*
Das Ergebnis des Thinning-Operators.

▷ **`GolayElement`** (input_control) ..... string ⇝ *string*
Strukturierendes Element aus dem Golay-Alphabet.
Defaultwert: ’l’
Werteliste: `GolayElement` ∈ {’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’}

▷ **`Iterations`** (input_control) ..... integer ⇝ *integer* / string
Anzahl der Iterationsschritte. Bei ’f’, ’f2’, ’h’ und ’i’ ist nur der Wert 1 sinnvoll.
Defaultwert: 20
Wertevorschläge: `Iterations` ∈ {’maximal’, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200}
Typischer Wertebereich: 1 ≤ Iterations (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(Iterations \cdot 6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `thinning_seq` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Nachfolger

pruning, reduce_domain, select_shape, area_center, connection, complement

Alternativen

skeleton, morph_skiz, expand_region

Siehe auch

hit_or_miss_seq, erosion_golay, difference, thinning_golay, thinning, thickening_seq

Modul

Foundation

16.9 OCR

close_ocr (: : OcrHandle :)

Freigeben des Speichers eines OCR-Klassifikators.

close_ocr ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

close_ocr gibt den Speicher des Klassifikators mit der Nummer OcrHandle frei. Damit werden alle zugehörigen Daten gelöscht. Diese können gegebenenfalls mit write_ocr vorher gesichert werden. Die Nummer OcrHandle ist nach dem Aufruf ungültig. Sie kann aber später vom System für neue Klassifikatoren wieder verwendet werden.

Achtung

Alle Daten des Klassifikators werden im Hauptspeicher (nicht auf der Platte) gelöscht.

Parameter

▷ **OcrHandle** (input_control) ..... ocr_box ⇝ *handle*
ID des zu löschenden OCR-Klassifikators.

Ergebnis

Falls OcrHandle gültig ist, liefert close_ocr den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OcrHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

write_ocr_trainf

Nachfolger

`read_ocr`

Modul

OCR/OCV

create_ocr_class_box (: : WidthPattern, HeightPattern, Interpolation, Features, Character : OcrHandle)

Erzeugen eines neues OCR-Klassifikators.

create_ocr_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

create_ocr_class_box erzeugt einen neuen OCR-Klassifikator. Zur Beschreibung des Klassifikators siehe Operator learn_class_box. Dieser Klassifikator muss dann mit traind_ocr_class_box oder trainf_ocr_class_box trainiert werden.

Die Parameter WidthPattern und HeightPattern geben die Größe der Eingabeschicht des Klassifikators an. Diese Größe wird bei den Merkmalen *'projection_horizontal'*, *'projection_vertical'*, *'pixel'*, *'pixel_invar'* und *'pixel_binary'* verwendet, um das zu erkennende Zeichen auf eine Standardgröße zu transformieren. Um so größer die Standardgröße ist, um so mehr Zeichen können (mit Grauwertmerkmalen) unterschieden werden. Dabei steigt aber auch die Zeit für das Training (und die Anzahl der Lernstichproben) und die Zeit für die Erkennung an. Der Parameter Interpolation gibt den Interpolationsmodus für die Anpassung der Zeichen im Bild auf die Größe im Klassifikator an. Dieser Parameter ist bei dem Operator affine_trans_image beschrieben. Der Wert *0* erzeugt dieselbe Interpolation wie *'nearest_neighbor'* in affine_trans_image, d.h. es erfolgt keine Interpolation. Für *1* erhält man dasselbe Verhalten wie *'constant'* in affine_trans_image, d.h. es erfolgt eine ungewichtete Interpolation zwischen den Grauwerten. Der Wert *2* erzeugt dieselbe Interpolation wie *'weighted'*, d.h. es erfolgt eine gewichtete Interpolation zwischen den Grauwerten. Der Parameter Interpolation muss so gewählt werden, dass bei der Skalierung der Zeichen auf die Standardgröße keine Aliasing-Effekte auftreten. In der Praxis bedeutet dies, dass Interpolation = *1* gewählt werden sollte, außer die Zeichen werden stark verkleinert. In diesem Fall sollte Interpolation = *2* gewählt werden. Interpolation = *0* sollte nur gewählt werden, falls die Zeichen nicht skaliert werden.

Der Parameter Character legt alle zu erkennenden Zeichen fest. Im Normalfall bestehen die übergebenen Strings aus einem Zeichen (z.B. Alphabet). Es können aber auch Strings mit beliebiger Länge gelernt werden. Die Anzahl unterscheidbarer Zeichen (Anzahl Strings in Character) ist auf *2048* beschränkt.

Mit dem Parameter Features können neben den Grauwerten zusätzliche Merkmale zur Erkennung der Buchstaben ausgewählt werden.

Es stehen folgende Merkmale zur Verfügung

'default' *'ratio'* und *'pixel_invar'* werden ausgewählt.

'ratio' Seitenverhältnis des Zeichens.

'width' Breite des Zeichens (nicht skalierungsinvariant).

'height' Höhe des Zeichens(nicht skalierungsinvariant).

'zoom_factor' Größenunterschied zwischen aktuellem Zeichen und den Werten von WidthPattern und HeightPattern (nicht skalierungsinvariant).

'foreground' Relativer Anteil der Vordergrund-Pixel.

'foreground_grid_9' Relative Anteil der Vordergrund-Pixel in einem 3×3 Raster innerhalb des umschließenden Rechtecks des Zeichens.

'foreground_grid_16' Relative Anteil der Vordergrund-Pixel in einem 4×4 Raster innerhalb des umschließenden Rechtecks des Zeichens.

'anisometry' Formmerkmal Anisometry.

'compactness' Formmerkmal Compactness.

'convexity' Formmerkmal Convexity.

'moments_region_2nd_invar' Normierte 2. geometrischen Momente der Region. Siehe auch moments_region_2nd_invar.

'moments_region_2nd_rel_invar' Normierte 2. relativen geometrischen Momente der Region. Siehe auch `moments_region_2nd_rel_invar`.

'moments_region_3rd_invar' Normierte 3. geometrischen Momente der Region. Siehe auch `moments_region_3rd_invar`.

'moments_central' Normierte zentrale geometrischen Momente der Region. Siehe auch `moments_region_central`.

'phi' Sinus und Kosinus der Orientierung (Winkel) des Zeichens.

'num_connect' Anzahl der Zusammenhangskomponenten.

'num_holes' Anzahl der Hohlflächen.

'projection_horizontal' Horizontal Projektion der Grauwert.

'projection_horizontal_invar' Horizontal Projektion der Grauwerte die maximal skaliert sind.

'projection_vertical' Vertikale Projektion der Grauwert.

'projection_vertical_invar' Vertikale Projektion der Grauwerte die maximal skaliert sind.

'cooc' Werte der binären Cooccurrence Matrix.

'moments_gray_plane' Normierte Grauwertmomente und die Winkel der Grauwertebene.

'num_runs' Anzahl der Sehnen der Region normiert auf die Höhe.

'chord_histo' Häufigkeit der Sehnen pro Zeile (nicht skalierungsinvariant).

'pixel' Grauwert der Zeichens.

'pixel_invar' Grauwerte der Zeichens mit automatischer maximaler Grauwertspreizung.

'pixel_binary' Region des Zeichens als Binärbild auf eine Größe von `WidthPattern` × `HeightPattern` skaliert.

'gradient_8dir' Die Gradienten des Zeichens werden berechnet. Die Gradientenrichtungen werden in 8 Richtungen diskretisiert. Das Amplitudenbild wird entsprechend der diskretisierten Richtungen in 8 Kanäle aufgeteilt. In jedem der 8 Kanäle werden auf einem 5×5 Gitter Stichproben entnommen. Diese Stichproben bilden die Merkmale.

Parameter

▷ **`WidthPattern`** (input_control) . . . . . integer ⇝ *integer*
Breite des Eingabenetzes.
Defaultwert: 8
Wertevorschläge: `WidthPattern` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich: $1 \leq$ `WidthPattern` ≤ 100

▷ **`HeightPattern`** (input_control) . . . . . integer ⇝ *integer*
Höhe des Eingabenetzes.
Defaultwert: 10
Wertevorschläge: `HeightPattern` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich: $1 \leq$ `HeightPattern` ≤ 100

▷ **`Interpolation`** (input_control) . . . . . integer ⇝ *integer*
Interpolationsmodus bei der Skalierung der Zeichen.
Defaultwert: 1
Werteliste: `Interpolation` ∈ {0, 1, 2}

▷ **`Features`** (input_control) . . . . . string(-array) ⇝ *string*
Zusätzliche Merkmale.
Defaultwert: 'default'
Werteliste: `Features` ∈ {'default', 'zoom_factor', 'ratio', 'width', 'height', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'anisometry', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'phi', 'num_connect', 'num_holes', 'projection_horizontal', 'projection_vertical', 'projection_horizontal_invar', 'projection_vertical_invar', 'chord_histo', 'num_runs', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'cooc', 'moments_gray_plane'}

▷ **`Character`** (input_control) . . . . . string-array ⇝ *string*
Alle Zeichen des Zeichensatzes.
Defaultwert: ['a','b','c']

▷ **OcrHandle** (output_control) . . . . . ocr_box ⇝ *handle*
ID des erzeugten OCR-Klassifikators.

Ergebnis

Sind die Parameter korrekt, dann liefert create_ocr_class_box den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

reset_obj_db

Nachfolger

traind_ocr_class_box, trainf_ocr_class_box, info_ocr_class_box, write_ocr, ocr_change_char

Alternativen

create_ocr_class_mlp, create_ocr_class_svm

Siehe auch

affine_trans_image, ocr_change_char, moments_region_2nd_invar, moments_region_2nd_rel_invar, moments_region_3rd_invar, moments_region_central

Modul

OCR/OCV

create_text_model (: : : TextModel)

Erzeugen eines Textmodells.

create_text_model ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen create_text_model_reader verwenden.

create_text_model erzeugt das neue Textmodell TextModel, welches den Text beschreibt, der von find_text segmentiert werden soll. Parameter des Textmodells werden mit set_text_model_param gesetzt und mit get_text_model_param abgefragt.

Das Erzeugen eines Textmodells mit create_text_model ist gleichbedeutend mit einem Aufruf von create_text_model_reader mit Mode = *'manual'*.

Parameter

▷ **TextModel** (output_control) . . . . . text_model ⇝ *handle*
Neu erzeugtes Textmodell.

Ergebnis

create_text_model gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Modul

OCR/OCV

deserialize_ocr (: : SerializedItemHandle : OcrHandle)

Deserialisieren eines serialisierten OCR-Klassifikators.

deserialize_ocr ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

deserialize_ocr deserialisiert einen OCR-Klassifikator, welcher mit serialize_ocr serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Der serialisierte OCR-Klassifikator wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten OCR-Klassifikator mit dem Handle OcrHandle gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **OcrHandle** (output_control) ..... ocr_box ⇝ *handle*
ID des eingelesenen OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_ocr den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

reset_obj_db, fread_serialized_item, receive_serialized_item, serialize_ocr

Nachfolger

do_ocr_multi, do_ocr_single, traind_ocr_class_box, trainf_ocr_class_box

Siehe auch

serialize_ocr, do_ocr_multi, traind_ocr_class_box, trainf_ocr_class_box

Modul

OCR/OCV

do_ocr_multi (Character, Image : : OcrHandle : Class, Confidence)

Klassifizieren von Zeichen.

do_ocr_multi ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

do_ocr_multi ordnet jedem Character (Buchstabe) eine Klasse zu. Dabei werden (bei Graustufenmerkmalen) alle Grauwerte aus den umschließenden Rechtecken der Regionen verwendet. Die Grauwerte werden aus dem Parameter Image entnommen. Für jeden Buchstaben wird die zugehörige Klasse in Class und ein Konfidenzwert in Confidence zurückgegeben. Der Konfidenzwert beschreibt die Ähnlichkeit zwischen dem übergebenen und dem zugeordneten Zeichen.

Parameter

- ▷ **Character** (input_object) . . . . . region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte für Zeichen.
- ▷ **OcrHandle** (input_control) . . . . . ocr_box ⇝ *handle*
  ID des OCR-Klassifikators.
- ▷ **Class** (output_control) . . . . . string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
  Parameteranzahl: Class == Character
- ▷ **Confidence** (output_control) . . . . . real(-array) ⇝ *real*
  Konfidenzwerte der Zeichen.
  Parameteranzahl: Confidence == Character

Ergebnis

Sind die Eingabeparameter korrekt besetzt, dann liefert `do_ocr_multi` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`traind_ocr_class_box`, `trainf_ocr_class_box`, `read_ocr`, `connection`, `sort_region`

Alternativen

`do_ocr_single`

Siehe auch

`write_ocr`

Modul

OCR/OCV

do_ocr_single (Character, Image : : OcrHandle : Classes, Confidences)

Klassifizieren von einem Zeichen.

`do_ocr_single` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

`do_ocr_single` ordnet dem `Character` (Buchstabe) Klassen zu. Dabei werden (bei Graustufenmerkmalen) alle Grauwerte aus den umschließenden Rechtecken der Regionen verwendet. Diese werden aus dem Parameter `Image` entnommen. Für jeden Buchstaben werden die beiden Klassen mit den höchsten Konfidenzen in `Classes` zurückgegeben. Die zugehörigen Konfidenzen werden in `Confidences` übergeben. Der Konfidenzwert beschreibt die Ähnlichkeit zwischen dem übergebenen und dem zugeordneten Zeichen.

Parameter

- ▷ **Character** (input_object) . . . . . region ⇝ *object*
  Zu erkennendes Zeichen.
- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte für Zeichen.
- ▷ **OcrHandle** (input_control) . . . . . ocr_box ⇝ *handle*
  ID des OCR-Klassifikator.
- ▷ **Classes** (output_control) . . . . . string-array ⇝ *string*
  Klassen (Name) der Zeichen.
  Parameteranzahl: 2

▷ **Confidences** (output_control) . . . . . real-array ⇝ *real*
Konfidenzwerte der Zeichen.
Parameteranzahl: 2

Ergebnis

Sind die Eingabeparameter korrekt besetzt, dann liefert `do_ocr_single` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`traind_ocr_class_box`, `trainf_ocr_class_box`, `read_ocr`, `connection`, `sort_region`

Alternativen

`do_ocr_multi`

Siehe auch

`write_ocr`

Modul

OCR/OCV

info_ocr_class_box (: : OcrHandle : WidthPattern, HeightPattern, Interpolation, WidthMaxChar, HeightMaxChar, Features, Characters)

Abfragen von Informationen über einen OCR-Klassifikator.

`info_ocr_class_box` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

`info_ocr_class_box` gibt alle einige Informationen über einen OCR-Klassifikator aus. Die Parameter entsprechen denen von `create_ocr_class_box`. Die Parameter `WidthMaxChar` und `HeightMaxChar` geben die Ausdehnung des größten trainierten Zeichens an. Diese Werte können zur Steuerung der Segmentierung verwendet werden.

Parameter

▷ **OcrHandle** (input_control) . . . . . ocr_box ⇝ *handle*
ID des OCR-Klassifikators.
▷ **WidthPattern** (output_control) . . . . . integer ⇝ *integer*
Breite der skalierten Zeichen.
▷ **HeightPattern** (output_control) . . . . . integer ⇝ *integer*
Höhe der skalierten Zeichen.
▷ **Interpolation** (output_control) . . . . . integer ⇝ *integer*
Interpolationsmodus beim Skalieren der Zeichen.
▷ **WidthMaxChar** (output_control) . . . . . integer ⇝ *integer*
Breite des größten trainierten Zeichens.
▷ **HeightMaxChar** (output_control) . . . . . integer ⇝ *integer*
Höhe des größten trainierten Zeichens.
▷ **Features** (output_control) . . . . . string-array ⇝ *string*
Verwendete Merkmale.
▷ **Characters** (output_control) . . . . . string-array ⇝ *string*
Alle Zeichen des Zeichensatzes.

Ergebnis

`info_ocr_class_box` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_ocr, create_ocr_class_box

Nachfolger

write_ocr

Modul

OCR/OCV

ocr_change_char (: : OcrHandle, Character :)

Festlegen einer neue Umsetzungstabelle für Zeichen.

ocr_change_char ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

ocr_change_char setzt bei dem Klassifikator eine neue Zuordnungstabelle für die Zeichen. Die Anzahl der Strings in Character muss genauso groß sein wie bei dem Netz OcrHandle. ocr_change_char kann folgendermaßen verwendet werden um einen Zeichensatz zu erweitern: Beim Erzeugen eines Netzes (create_ocr_class_box) werden mehr Zeichen angegeben als zunächst benötigt. Es bleiben die letzten n Zeichen zunächst unbenutzt. Wenn man mehr Zeichen benötigt, dann werden mit ocr_change_char diese unbenutzten Zeichen besetzt und danach trainiert.

Parameter

▷ **OcrHandle** (input_control) . . . . . ocr_box ⇝ *handle*
ID des zu ändernden OCR-Netzes.

▷ **Character** (input_control) . . . . . string-array ⇝ *string*
Neue Zeichenzuordnung.
Defaultwert: ['a','b','c']

Ergebnis

Stimmt die Anzahl der Zeichen in Character mit denen des Netzes überein, dann liefert ocr_change_char den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OcrHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

read_ocr

Nachfolger

do_ocr_multi_class_mlp, do_ocr_single_class_mlp

Modul

OCR/OCV

ocr_get_features (Character : : OcrHandle : FeatureVector)

Abfragen der Merkmale eines Zeichens.

ocr_get_features ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

ocr_get_features bestimmt die Merkmale für das übergebene Zeichen. Die Art und Anzahl der Merkmale werden durch den Klassifikator OcrHandle festgelegt. Es sind die gleichen Werte, die bei Operatoren wie traind_ocr_class_box oder trainf_ocr_class_box intern verwendet werden.

Parameter

▷ **Character** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Zu trainierende Zeichen.

▷ **OcrHandle** (input_control) . . . . . ocr_box ⇝ *handle*
ID des gewünschten OCR-Klassifikators.

▷ **FeatureVector** (output_control) . . . . . real-array ⇝ *real*
Merkmalswerte.

Ergebnis

Sind die Parameter korrekt, dann liefert ocr_get_features den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_ocr, reduce_domain, threshold, connection

Siehe auch

trainf_ocr_class_box, traind_ocr_class_box

Modul

OCR/OCV

read_ocr (: : FileName : OcrHandle)

Einlesen eines OCR-Klassifikators aus einer Datei.

read_ocr ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

read_ocr liest einen OCR-Klassifikator aus der Datei FileName. Die Datei wird dabei im Verzeichnis ($HALCONROOT/ocr/) und im aktuellen Verzeichnis gesucht. Sind bereits zu viele Klassifikatoren geladen, erfolgt eine Fehlermeldung.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei mit dem OCR-Klassifikator.
Defaultwert: 'testnet'
Dateiendung: .obc, .fnt

▷ **OcrHandle** (output_control) . . . . . ocr_box ⇝ *handle*
ID des eingelesenen OCR-Klassifikators.

Ergebnis

Ist die angegebene Datei vorhanden und der Format korrekt, dann liefert read_ocr den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

reset_obj_db

Nachfolger

do_ocr_multi, do_ocr_single, traind_ocr_class_box, trainf_ocr_class_box

Siehe auch

write_ocr, do_ocr_multi, traind_ocr_class_box, trainf_ocr_class_box

Modul

OCR/OCV

serialize_ocr (: : OcrHandle : SerializedItemHandle)

Serialisieren eines OCR-Klassifikators.

serialize_ocr ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

serialize_ocr serialisiert einen OCR-Klassifikator (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_ocr in eine Datei geschrieben werden. Der OCR-Klassifikator wird in dem Handle OcrHandle übergeben. Der serialisierte OCR-Klassifikator wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_ocr wieder deserialisiert werden.

Parameter

▷ **OcrHandle** (input_control) ..... ocr_box ⇝ *handle*
ID des OCR-Klassifikators.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_ocr den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

traind_ocr_class_box, trainf_ocr_class_box

Nachfolger

do_ocr_multi, do_ocr_single, fwrite_serialized_item, send_serialized_item, deserialize_ocr

Siehe auch

deserialize_ocr, do_ocr_multi, traind_ocr_class_box, trainf_ocr_class_box

Modul

OCR/OCV

testd_ocr_class_box (Character, Image : : OcrHandle, Class : Confidence)

Testen eines OCR-Klassifikators.

testd_ocr_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

testd_ocr_class_box testet die Güte der Klassenzugehörigkeit von Zeichen. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in Character muss der zugehörige Name (Klasse) Class übergeben werden. Die Grauwerte übergibt man in Image. Confidence gibt Auskunft über die Zugehörigkeit zu einer (beliebig wählbaren) Klasse.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu testende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte für die Zeichen.
- ▷ **OcrHandle** (input_control) ..... ocr_box ⇝ *handle*
  ID des gewünschten OCR-Klassifikators.
- ▷ **Class** (input_control) ..... string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
  Defaultwert: 'a'
- ▷ **Confidence** (output_control) ..... real(-array) ⇝ *real*
  Konfidenz mit der das Zeichen der Klasse zugehört.

Ergebnis

Sind die Parameter korrekt, dann liefert testd_ocr_class_box den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_ocr, trainf_ocr_class_box, traind_ocr_class_box

Modul

OCR/OCV

traind_ocr_class_box (Character, Image : : OcrHandle, Class : AvgConfidence)

Trainieren eines OCR-Klassifikators durch Eingabe von Regionen.

traind_ocr_class_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

traind_ocr_class_box trainiert den Klassifikator direkt über die eingegebenen Regionen in einem Bild. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in Character muss der zugehörige Name (Klasse) Class übergeben werden. Die Grauwerte übergibt man in Image. AvgConfidence gibt Auskunft über den Erfolg des Trainings: Er enthält die mittlere Konfidenz der trainierten Zeichen bei einer Reklassifikation. Dabei werden falsch klassifizierte Zeichen mit Konfidenz 0 bewertet.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu trainierende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte für die Zeichen.
- ▷ **OcrHandle** (input_control) ..... ocr_box ⇝ *handle*
  ID des gewünschten OCR-Klassifikators.
- ▷ **Class** (input_control) ..... string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
  Defaultwert: 'a'
- ▷ **AvgConfidence** (output_control) ..... real ⇝ *real*
  Mittlere Konfidenz bei einer Reklassifikation der trainierten Zeichen.

Ergebnis

Sind die Parameter korrekt, dann liefert `traind_ocr_class_box` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OcrHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ocr_class_box`, `read_ocr`

Nachfolger

`write_ocr`, `do_ocr_multi`, `do_ocr_single`

Alternativen

`trainf_ocr_class_box`

Modul

OCR/OCV

trainf_ocr_class_box (: : OcrHandle, TrainingFile : AvgConfidence)

Trainieren eines OCR-Klassifikators mit einer Trainingsdatei.

`trainf_ocr_class_box` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

`trainf_ocr_class_box` trainiert den Klassifikator `OcrHandle` über die angegebenen Trainingsdateien. Es können beliebig viele Dateien angegeben werden. `AvgConfidence` gibt Auskunft über den Erfolg des Trainings: Er enthält die mittlere Konfidenz der trainierten Zeichen bei einer Kontroll-Klassifikation. Dabei werden falsch klassifizierte Zeichen mit Konfidenz 0 bewertet. Es ist zu beachten, dass Trainingszeichen, die keine entsprechende Klasse im Klassifikator `OcrHandle` haben, verworfen werden.

Achtung

Die Namen der Buchstaben in der Datei müssen zu dem Netz passen.

Parameter

▷ **OcrHandle** (input_control) . . . . . ocr_box ⇝ *handle*
ID des gewünschten OCR-Netzes.

▷ **TrainingFile** (input_control) . . . . . filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: 'train_ocr'
Dateiendung: `.trf`, `.otr`

▷ **AvgConfidence** (output_control) . . . . . real ⇝ *real*
Mittlere Konfidenz bei einer Reklassifikation der trainierten Zeichen.

Ergebnis

Ist der Dateiname korrekt und die Daten passen zu dem Netz, dann liefert `trainf_ocr_class_box` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OcrHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ocr_class_box`, `read_ocr`

Nachfolger

`traind_ocr_class_box`, `write_ocr`, `do_ocr_multi`, `do_ocr_single`

Alternativen

`traind_ocr_class_box`

Modul

OCR/OCV

write_ocr (: : OcrHandle, FileName :)

Schreiben eines OCR-Klassifikators in eine Datei.

`write_ocr` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die auf MLP, SVM oder CNN-basierten Operatoren verwenden.

`write_ocr` schreibt den OCR-Klassifikator `OcrHandle` in die Datei `FileName`. Da die Daten des Klassifikators nach Beendigung des Programms verloren gehen, muss man nach dem Training die Daten sichern, wenn man sie in einem späteren Programmlauf wieder verwenden will. Die Daten können später wieder mit `read_ocr` eingelesen werden. Die Extension wird automatisch an `FileName` angehängt.

Achtung

Die Ausgabedatei `FileName` muss ohne Extension angegeben werden.

Parameter

▷ **OcrHandle** (input_control) . . . . . ocr_box ⇝ *handle*
ID des OCR-Klassifikators.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei für den OCR-Klassifikator (ohne Extension).
Defaultwert: 'my_ocr'
Dateiendung: `.obc`

Ergebnis

Ist OcrHandle gültig und kann die angegebene Datei geschrieben werden, dann liefert write_ocr den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

traind_ocr_class_box, trainf_ocr_class_box

Nachfolger

do_ocr_multi, do_ocr_single

Siehe auch

read_ocr, do_ocr_multi, traind_ocr_class_box, trainf_ocr_class_box

Modul

OCR/OCV

16.10 Regionen

get_region_chain (Region : : : Row, Column, Chain)

Kontur von einem Objekt als Kettencode.

get_region_chain ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

get_region_chain gibt die Kontur einer Region aus. Eine Kontur ist eine Folge von Punkten, die die Umrandung der Region beschreiben. Die Kontur „liegt auf“ der Region. Sie beginnt bei der kleinsten Zeilennummer; in dieser Zeile bei dem Punkt mit dem größten Spaltenindex. Der Umlauf erfolgt im Uhrzeigersinn. Hohlflächen der Region werden ignoriert. Der Richtungscode (Kettencode) ist wie folgt definiert:

$$\begin{array}{ccc} 3 & 2 & 1 \\ 4 & * & 0 \\ 5 & 6 & 7 \end{array}$$

get_region_chain liefert den Code in Form eines Tupels ab. Bei einer leeren Region sind die Parameter Row und Column gleich Null und Chain ist das leere Tupel.

Achtung

Hohlflächen der Region werden ignoriert. Es darf nur eine Region übergeben werden und die muss genau eine Zusammenhangskomponente haben.

Parameter

▷ **Region** (input_object) ..... region ⇝ *object*
Zu transformierende Region.

▷ **Row** (output_control) ..... chain.begin.y ⇝ *integer*
Zeile des Startpunktes.

▷ **Column** (output_control) ..... chain.begin.x ⇝ *integer*
Spalte des Startpunktes.

▷ **Chain** (output_control) ..... chain.code-array ⇝ *integer*
Richtungscode der Kontur (ab Startpunkt).
Typischer Wertebereich: $0 \leq$ Chain ≤ 7

Ergebnis

get_region_chain liefert normalerweise den Wert TRUE. Wird mehr als eine Zusammenhangskomponente übergeben, wird ein Exception ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei leerer Region (die Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

sobel_amp, threshold, skeleton, edges_image, gen_rectangle1, gen_circle

Nachfolger

approx_chain, approx_chain_simple

Siehe auch

copy_obj, get_region_contour, get_region_polygon

Modul

Foundation

hamming_change_region (InputRegion : OutputRegion : Width, Height, Distance :)

Erzeugen einer Region mit vorgegebenem Hamming-Abstand.

hamming_change_region ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

hamming_change_region verändert die Regionen in dem linken oberen Bildausschnitt (der Größe Width x Height) so, dass die entstehenden Regionen zu den Eingabedaten den Hamming-Abstand Distance haben. Dazu werden Distance Punkte zur jeweiligen Eingaberegion hinzugefügt bzw. aus ihr entfernt.

Achtung

Wenn Width und Height zu groß gewählt werden, benötigt die erzeugte Region viel Speicherplatz.

Parameter

▷ **InputRegion** (input_object) ..... region(-array) ⇝ *object*
Zu modifizierenden Regionen.

▷ **OutputRegion** (output_object) ..... region(-array) ⇝ *object*
Regionen, die den angegebenen Hamming-Abstand haben.

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Horizontale Ausdehnung des zu ändernden Bereichs.
Defaultwert: 100
Wertevorschläge: Width ∈ {64, 128, 256, 512}
Typischer Wertebereich: 1 ≤ Width ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width > 0

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Vertikale Ausdehnung des zu ändernden Bereichs.
Defaultwert: 100
Wertevorschläge: Height ∈ {64, 128, 256, 512}
Typischer Wertebereich: 1 ≤ Height ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height > 0

▷ **Distance** (input_control) . . . . . integer ⇝ *integer*
Hamming-Abstand zwischen den alten und den neuen Regionen.
Defaultwert: 1000
Wertevorschläge: Distance ∈ {100, 500, 1000, 5000, 10000}
Typischer Wertebereich: 0 ≤ Distance ≤ 10000 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Distance >= 0 && Distance < Width * Height

Komplexität

Speicherplatzkomplexität der erzeugten Region in Byte (worst case): $O(2 * \texttt{Width} * \texttt{Height})$.

Ergebnis

hamming_change_region liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

connection, regiongrowing, pouring, class_ndim_norm

Nachfolger

select_shape

Siehe auch

hamming_distance

Modul

Foundation

interjacent (Region : RegionInterjacent : Mode :)

Unterteilen der Bildebene mittels vorgegebener Regionen.

interjacent ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

interjacent unterteilt die Bildebene mittels der Regionen in Region. Als Ergebnis wird eine Ausgaberegion mit den ermittelten Trennlinien zurückgeliefert. Es stehen dafür drei verschiedene Modi zur Verfügung:

'medial_axis' Dieser Modus wird für Eingaberegionen verwendet, die sich nicht berühren. Die Prozedur findet dann Trennlinien zwischen den Regionen, die den Hintergrund „gerecht“ zwischen den Eingaberegionen aufteilt. Ihre Anwendung entspricht der Ausführung folgender Befehlssequenz:

Beispiel:

```
complement('full',Region,Tmp)
skeleton(Tmp,Result)
```

'border' Für einander nicht berührende Eingaberegionen entspricht die Prozedur in diesem Modus boundary (Region,Result), ersetzt also jeder Eingaberegion durch ihre Kontur. Einander berührende Eingaberegionen werden zu einer Region zusammengefasst. Die zugehörige Ausgaberegion umfasst dann deren Kontur und die (ein Pixel breite) Trennlinie zwischen den ursprünglichen Regionen. interjacent führt in diesem Fall also folgende Einzelschritte durch:

Beispiel:

```
boundary(Region,Tmp1,'inner')
union1(Tmp1,Tmp2)
skeleton(Tmp2,Result)
```

'mixed' In diesem Modus verhält sich `interjacent` für einander nicht berührende Eingaberegionen wie im Modus *'medial_axis'*. Kommt es zu einer Berührung, werden zwar wiederum Trennlinien zwischen den Eingaberegionen als Ausgaberegionen erzeugt, nun aber inklusive der „Berührungslinie" der Regionen. Einander berührende Regionen werden also (durch eine Linie in der Ausgaberegion) „getrennt". Dies entspricht der Anweisungssequenz:

Beispiel:

```
erosion1(Region,Mask,Tmp1,1)
union1(Tmp1,Tmp2)
complement(full,Tmp2,Tmp3)
skeleton(Tmp3,Result)
```

wobei Mask folgende „Kreuzmaske" bezeichnet:



Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegionen, zwischen deren Regionen Trennlinien gefunden werden sollen.
- ▷ **RegionInterjacent** (output_object) ..... region ⇝ *object*
  Eine Ausgaberegion mit den ermittelten Trennlinien als Region.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Gewünschter Modus.
  Defaultwert: 'mixed'
  Werteliste: `Mode` ∈ {'medial_axis', 'border', 'mixed'}

Beispiel

```
read_image(Image,'forest_air1')
mean_image(Image,Mean,31,31)
dyn_threshold(Image,Mean,Seg,20,'light')
interjacent(Seg,Graph,'medial_axis')
dev_display(Graph)
```

Ergebnis

`interjacent` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<'2 (H_MSG_TRUE)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `connection`, `regiongrowing`, `pouring`

Nachfolger

`select_shape`, `disp_region`

Siehe auch

`expand_region`, `junctions_skeleton`, `boundary`

Modul

Foundation

16.11 Segmentation

bin_threshold (Image : Region : :)

Segmentieren eines Bildes mit automatisch gewählter Schwelle.

bin_threshold ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den Operator binary_threshold verwenden.

bin_threshold segmentiert ein einkanaliges Bild mittels eines automatisch bestimmten Schwellenwertes. Als erstes wird dabei das relative Histogramm über den Grauwerten ermittelt. Dann werden im Histogramm relevante Minima gesucht, die als Schwellen für die Schwellenwertsegmentierung dienen. Um die Anzahl der Minima zu reduzieren, wird das Histogramm wie bei auto_threshold mit einer Gaußmaske geglättet. Die Maske wird dabei so lange vergrößert, bis nur noch ein Minimum vorhanden ist. Ausgewählt werden die Grauwerte von Null bis zu diesem Minimum oder für real Bilder den kleinsten Wert zu dem minimalen Wert. Das Verfahren ist z.B. für die Segmentierung von dunklen Buchstaben auf hellem Papier geeignet.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **Region** (output_object) ..... region(-array) ⇝ *object*
  Dunkle Bereiche des Bildes.

Beispiel

```
read_image (Image, 'letters')
bin_threshold (Image, Seg)
connection (Seg, Connected)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Modul

Foundation

class_ndim_box (MultiChannelImage : Regions : ClassifHandle :)

Klassifizieren von Bildpunkten durch Hyperquader.

class_ndim_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

class_ndim_box klassifiziert die Bildpunkte von mehrkanaligen Bildern. Es wird hierzu der Klassifikator ClassifHandle verwendet, der mit create_class_box erzeugt wurde. Trainiert werden kann der Klassifikator mit learn_ndim_box oder wie bei create_class_box beschrieben. Siehe auch class_ndim_norm.

MultiChannelImage ist ein mehrkanaliges Bild, dessen Bildpunkte klassifiziert werden sollen. D.h. es werden die Bildpunkte ausgewählt, die dem Klassifikationskriterium genügen.

Parameter

▷ **MultiChannelImage** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / int4 / real
Mehrkanaliges Eingabebild.

▷ **Regions** (output_object) ..... region-array ⇝ *object*
Ergebnis der Klassifikation.

▷ **ClassifHandle** (input_control) ..... class_box ⇝ *handle*
Handle des Klassifikators.

Beispiel

```
read_image(Image,'montery')
dev_display(Image)
dev_set_color('green')
disp_message (WindowHandle, 'Draw learning region  ',\
              'window', 12, 12, 'black', 'true')
draw_region(Reg1,WindowHandle)
reduce_domain(Image,Reg1,Foreground)
dev_set_color('red')
disp_message (WindowHandle, 'Draw background region',\
              'window', 12, 12, 'black', 'true')
draw_region(Reg2,WindowHandle)
reduce_domain(Image,Reg2,Background)
disp_message (WindowHandle, 'Training...           ',\
              'window', 12, 12, 'black', 'true')
create_class_box(ClassifHandle)
learn_ndim_box(Foreground,Background,Image,ClassifHandle)
disp_message (WindowHandle, 'Classification        ',\
              'window', 12, 12, 'black', 'true')
class_ndim_box(Image,Res,ClassifHandle)
dev_set_draw('fill')
dev_display (Res)
close_class_box(ClassifHandle)
```

Komplexität

Sei N die Anzahl der Hyperquader und A die Fläche der Eingaberegion(en), dann ist die Laufzeitkomplexität $O(N, A)$.

Ergebnis

`class_ndim_box` liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_class_box`, `learn_class_box`, `median_image`, `compose2`, `compose3`, `compose4`, `compose5`, `compose6`, `compose7`

Alternativen

`class_ndim_norm`, `class_2dim_sup`, `class_2dim_unsup`

Siehe auch

`descript_class_box`

Modul

Foundation

expand_line (Image : RegionExpand : Coordinate, ExpandType, RowColumn, Threshold :)

Expandieren ausgehend von einer Linie.

expand_line ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

expand_line erzeugt eine Region durch Expansion, ausgehend von einer Linie (Zeile oder Spalte). Die Expansion wird beendet, sobald der betrachtete Grauwert um mehr als Threshold vom Mittelwert entlang der Linie (ExpandType = *'mean'*) oder vom zuletzt hinzugefügten Grauwert (ExpandType = *'gradient'*) abweicht.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild.
- ▷ **RegionExpand** (output_object) ..... region(-array) ⇝ *object*
  Gefundenes Segmente.
- ▷ **Coordinate** (input_control) ..... integer ⇝ *integer*
  Koordinate der Zeile oder Spalte.
  Defaultwert: 256
  Wertevorschläge: Coordinate ∈ {16, 64, 128, 200, 256, 300, 400, 511}
  Restriktion: Coordinate >= 0
- ▷ **ExpandType** (input_control) ..... string ⇝ *string*
  Kriterium für Abbruch.
  Defaultwert: 'gradient'
  Werteliste: ExpandType ∈ {'gradient', 'mean'}
- ▷ **RowColumn** (input_control) ..... string ⇝ *string*
  Segmentierungsmodus (Zeile oder Spalte).
  Defaultwert: 'row'
  Werteliste: RowColumn ∈ {'row', 'column'}
- ▷ **Threshold** (input_control) ..... number ⇝ *real* / integer
  Schwellenwert für Ausdehnung.
  Defaultwert: 3.0
  Wertevorschläge: Threshold ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 13.0, 17.0, 20.0, 30.0}
  Typischer Wertebereich: 1.0 ≤ Threshold ≤ 255.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0
  Restriktion: Threshold >= 0.0 && Threshold <= 255.0

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  HImage   image (argv[1]),
           gauss;
  HWindow  win;

  win.SetDraw ("margin");
  win.SetColored (12);

  image.Display (win);

  gauss = image.GaussImage (5);
```

```
  HRegionArray reg = gauss.ExpandLine (100, "mean", "row", 5.0);

  reg.Display (win);
  win.Click ();

  return (0);
}
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

binomial_filter, gauss_filter, smooth_image, anisotropic_diffusion, median_image, affine_trans_image, rotate_image

Nachfolger

intersection, opening, closing

Alternativen

regiongrowing_mean, expand_gray, expand_gray_ref

Modul

Foundation

learn_ndim_box (Foreground, Background, MultiChannelImage : : ClassifHandle :)

Trainieren eines Klassifikators mit mehrkanaligen Bildern.

learn_ndim_box ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

learn_ndim_box trainiert den Klassifikator ClassifHandle mit den Bildpunkten des mehrkanaligen Eingabebildes der Region, die durch Foreground angegeben wird. Die Punkte in Background sollen von dem Klassifikator zurückgewiesen werden. Der so trainierte Klassifikator kann für class_ndim_box zur Segmentierung von Bildern verwendet werden. Foreground soll gefunden werden, Background sind die Bildteile, die nicht gefunden werden sollen.

Beim Trainingsvorgang wird jedes Pixel einmal trainiert. Für Bildpunkte aus Foreground wird die Klasse „0“, für Background die Klasse „1“ verwendet. Es wird abwechselnd mit einem Bildpunkt aus Foreground und einem aus Background trainiert. Falls eine Region kleiner ist, wird zyklisch von vorne begonnen, bis die andere abgearbeitet ist. learn_ndim_box akzeptiert später bei der Segmentierung nur die Bildpunkte, die der Klasse „0“ zugeordnet werden.

Aus Anwendersicht besteht der wesentliche Unterschied zwischen learn_ndim_norm und learn_ndim_box darin, dass bei letzterem die Background-Klasse den Klassifikationsprozess selbst mitsteuert. Hier wird eine Trennfläche zwischen Foreground- und Background-Klasse erzeugt, so dass keine Punkte im Merkmalsraum falsch klassifiziert werden. Bei learn_ndim_norm dagegen ist eine Überlappung von Foreground- und Background-Klasse erlaubt. Dies schlägt sich dann im Rückgabewert Quality nieder: Je größer die Überlappung, desto kleiner die Güte der Klassifikation.

Achtung

Alle Kanäle müssen von dem gleichen Typ sein.

Parameter

▷ **Foreground** (input_object) ..... region(-array) ⇝ *object*
Zu trainierende Vordergrundpunkte.

▷ **Background** (input_object) ..... region(-array) ⇝ *object*
Zu trainierende Hintergrundpunkte (Rückweisungsklasse).

▷ **MultiChannelImage** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / int4 / real
Mehrkanaliges Bild.

▷ **ClassifHandle** (input_control) ..... class_box ⇝ *handle*
Handle des Klassifikators.

Komplexität

Sei N die Anzahl der erzeugten Hyperquader und A die Fläche der größeren Eingaberegion. Dann ist die Laufzeitkomplexität $O(N * A)$.

Ergebnis

`learn_ndim_box` liefert den Wert TRUE, falls die Parameter korrekt sind und ein Klassifikator aktiv ist. Für das Verhalten bzgl. der Eingabebilder sind die Flags *'no_object_result'* und *'empty_region_result'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ClassifHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_box`, `draw_region`

Nachfolger

`class_ndim_box`, `descript_class_box`

Alternativen

`learn_class_box`, `learn_ndim_norm`

Modul

Foundation

16.12 Steuerung

ifelse (: : Condition :)

Bedingte Anweisung mit Alternative.

`ifelse` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Bitte nur noch den Operator `if` verwenden.

`ifelse` stellt eine bedingte Anweisung mit Alternative dar. Falls die Bedingung wahr ist (also ungleich *0*), werden alle Aufrufe bis zum nächsten Operator `else` ausgeführt. Ist die Bedingung falsch (also *0*), wird der Programmteil zwischen `else` und `endif` ausgeführt. Es ist zu beachten, dass der Operator `ifelse` als `if` im Programmeditor dargestellt wird.

Parameter

▷ **Condition** (input_control) ..... integer ⇝ *integer*
Bedingung für die if-Anweisung.
Defaultwert: 1

Ergebnis

Enthält die Bedingung einen korrekten Ausdruck, dann liefert `ifelse` (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`if`

Siehe auch

`else`, `elseif`, `for`, `while`, `repeat`, `until`

Modul

Foundation

16.13 Tools

```
approx_chain ( : : Row, Column, MinWidthCoord, MaxWidthCoord,
    ThreshStart, ThreshEnd, ThreshStep, MinWidthSmooth,
    MaxWidthSmooth, MinWidthCurve, MaxWidthCurve, Weight1, Weight2,
    Weight3 : ArcCenterRow, ArcCenterCol, ArcAngle, ArcBeginRow,
    ArcBeginCol, LineBeginRow, LineBeginCol, LineEndRow, LineEndCol,
    Order )
```

Approximation einer Kontur durch Bögen und Linien.

`approx_chain` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

Die Koordinaten einer Kurve werden durch eine Folge von Linien und Kreisbögen approximiert. Das Verfahren durchläuft dabei für bestimmte Parameter Wertebereiche. In der Parameterliste der Funktion werden die Grenzen dieser Bereiche explizit angegeben (MinWidthCoord ... MaxWidthCoord, ThreshStart ... ThreshEnd, MinWidthSmooth ... MaxWidthSmooth, MinWidthCurve ... MaxWidthCurve). Zusätzlich muss für den Parameterbereich des Schwellenwertes für spitze Ecken auch die Schrittweite angegeben werden (ThreshStep). Durch Einengung der durchlaufenen Bereiche kann die Laufzeit der Berechnung verkürzt, das Ergebnis aber eventuell verschlechtert werden.

Durch die Parameter Weight1, Weight2 und Weight3 kann angegeben werden, ob die gewünschte Gewichtung mehr auf Genauigkeit der Approximation oder Erhalt möglichst vieler großer oder möglichst weniger kleiner Segmente liegt. Für (Weight1,Weight2,Weight3) erzeugt also (1,0,0) eine sehr genaue Approximation (0,1,1) eine Approximation mit möglichst wenigen und großen Segmenten.

Das Ergebnis des Verfahren wird getrennt nach Bögen und Linien zurückgegeben. Ist man an der Reihenfolge der Segmente interessiert, so können die einzelnen Ergebniselemente sukzessive aus den Ergebnistupeln ausgelesen werden, wobei dem Rückgabeparameter Order die Reihenfolge entnommen werden kann (0: nächstes Element ist nächstes Liniensegment, 1: nächstes Element ist nächstes Bogensegment).

Achtung

Konturen, die eventuell nur aus einem Segment bestehen können, sollten auch mit einem Schwellenwertmaximum (ThreshEnd) > 1.0 untersucht werden, da ansonsten auf jeden Fall mindestens ein „Eckpunkt“ bestimmt wird.

Parameter

- ▷ **`Row`** (input_control) . . . . . point.y-array ⇝ *integer*
  Zeilenwert der Kontur.
  Defaultwert: 32
- ▷ **`Column`** (input_control) . . . . . point.x-array ⇝ *integer*
  Spaltenwert der Kontur.
  Defaultwert: 32
- ▷ **`MinWidthCoord`** (input_control) . . . . . real ⇝ *real*
  Minimale Breite des Gaußoperators zur Koordinatenglättung (> 0.4).
  Defaultwert: 0.5
  Wertevorschläge: `MinWidthCoord` $\in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\}$
  Typischer Wertebereich: $0.4 \leq$ `MinWidthCoord` ≤ 3.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.1

▷ **MaxWidthCoord** (input_control) . . . . . real ⇝ *real*
Maximale Breite des Gaußoperators zur Koordinatenglättung.
Defaultwert: 2.4
Wertevorschläge: MaxWidthCoord ∈ {0.5, 0.7, 1.0, 1.2, 1.5, 1.7}
Typischer Wertebereich: 0.4 ≤ MaxWidthCoord ≤ 3.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1

▷ **ThreshStart** (input_control) . . . . . real ⇝ *real*
Minimaler Schwellenwert der Krümmung für die Akzeptanz einer Ecke (relativ zur größten auftretenden Krümmung).
Defaultwert: 0.3
Wertevorschläge: ThreshStart ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8}
Typischer Wertebereich: 0.1 ≤ ThreshStart ≤ 0.9 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1

▷ **ThreshEnd** (input_control) . . . . . real ⇝ *real*
Maximaler Schwellenwert der Krümmung für die Akzeptanz einer Ecke (relativ zur größten auftretenden Krümmung).
Defaultwert: 0.9
Wertevorschläge: ThreshEnd ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8}
Typischer Wertebereich: 0.1 ≤ ThreshEnd ≤ 0.9 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1

▷ **ThreshStep** (input_control) . . . . . real ⇝ *real*
Schrittweite für den Schwellenwertanstieg.
Defaultwert: 0.2
Wertevorschläge: ThreshStep ∈ {0.3, 0.4, 0.5}
Typischer Wertebereich: 0.1 ≤ ThreshStep ≤ 0.9 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1

▷ **MinWidthSmooth** (input_control) . . . . . real ⇝ *real*
Minimale Breite des Gaußoperators zur Glättung der Krümmungsfunktion (> 0.4).
Defaultwert: 0.5
Wertevorschläge: MinWidthSmooth ∈ {0.5, 0.7, 1.0, 1.2, 1.5, 1.7}
Typischer Wertebereich: 0.4 ≤ MinWidthSmooth ≤ 3.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1

▷ **MaxWidthSmooth** (input_control) . . . . . real ⇝ *real*
Maximale Breite des Gaußoperators zur Glättung der Krümmungsfunktion.
Defaultwert: 2.4
Wertevorschläge: MaxWidthSmooth ∈ {0.5, 0.7, 1.0, 1.2, 1.5, 1.7}
Typischer Wertebereich: 0.4 ≤ MaxWidthSmooth ≤ 3.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1

▷ **MinWidthCurve** (input_control) . . . . . integer ⇝ *integer*
Minimale Breite des Kurvenbereichs zur Krümmungsbestimmung (> 0).
Defaultwert: 2
Wertevorschläge: MinWidthCurve ∈ {2, 5, 7}
Typischer Wertebereich: 1 ≤ MinWidthCurve ≤ 12 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2

▷ **MaxWidthCurve** (input_control) . . . . . integer ⇝ *integer*
Maximale Breite des Kurvenbereichs zur Krümmungsbestimmung.
Defaultwert: 12
Wertevorschläge: MaxWidthCurve ∈ {2, 5, 7}
Typischer Wertebereich: 1 ≤ MaxWidthCurve ≤ 20 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2

▷ **Weight1** (input_control) . . . . . real ⇝ *real*
Gewichtungsfaktor für die Approximationsgenauigkeit.
Defaultwert: 1.0
Wertevorschläge: `Weight1` ∈ {0.0, 0.5, 1.0}
Typischer Wertebereich: 0.0 ≤ `Weight1` ≤ 1.0 (lin)
Minimale Schrittweite: 0.1
Empfohlene Schrittweite: 0.5

▷ **Weight2** (input_control) . . . . . real ⇝ *real*
Gewichtungsfaktor für große Segmente.
Defaultwert: 1.0
Wertevorschläge: `Weight2` ∈ {0.0, 0.5, 1.0}
Typischer Wertebereich: 0.0 ≤ `Weight2` ≤ 1.0 (lin)
Minimale Schrittweite: 0.1
Empfohlene Schrittweite: 0.5

▷ **Weight3** (input_control) . . . . . real ⇝ *real*
Gewichtungsfaktor für kleine Segmente.
Defaultwert: 1.0
Wertevorschläge: `Weight3` ∈ {0.0, 0.5, 1.0}
Typischer Wertebereich: 0.0 ≤ `Weight3` ≤ 1.0 (lin)
Minimale Schrittweite: 0.1
Empfohlene Schrittweite: 0.5

▷ **ArcCenterRow** (output_control) . . . . . arc.center.y-array ⇝ *integer*
Zeile des Mittelpunkts eines Bogens.

▷ **ArcCenterCol** (output_control) . . . . . arc.center.x-array ⇝ *integer*
Spalte des Mittelpunkts eines Bogens.

▷ **ArcAngle** (output_control) . . . . . arc.angle.rad-array ⇝ *real*
Winkel eines Bogens.

▷ **ArcBeginRow** (output_control) . . . . . arc.begin.y-array ⇝ *integer*
Zeile des Startpunkts eines Bogens.

▷ **ArcBeginCol** (output_control) . . . . . arc.begin.x-array ⇝ *integer*
Spalte des Startpunkts eines Bogens.

▷ **LineBeginRow** (output_control) . . . . . line.begin.y-array ⇝ *integer*
Zeile des Startpunkts einer Linie.

▷ **LineBeginCol** (output_control) . . . . . line.begin.x-array ⇝ *integer*
Spalte des Startpunkts einer Linie.

▷ **LineEndRow** (output_control) . . . . . line.end.y-array ⇝ *integer*
Zeile des Endpunkts einer Linie.

▷ **LineEndCol** (output_control) . . . . . line.end.x-array ⇝ *integer*
Spalte des Endpunkts einer Linie.

▷ **Order** (output_control) . . . . . integer-array ⇝ *integer*
Reihenfolge von Linien- (Wert 0) und Bogensegmenten (Wert1).

Beispiel

```
/* read edge image  */
read_image(&Image,"fig1_kan");
/* construct edge region  */
hysteresis_threshold(Image,&RK1,64,255,40,1);
connection(RK1,&Rand);
/* fetch chain code  */
T_get_region_contour(Rand,&Rows,&Columns);
firstline = get_i(Tline,0);
firstcol = get_i(Tcol,0);
/* approximation with lines and circular arcs */
set_d(t1,0.4,0);
set_d(t2,2.4,0);

set_d(t3,0.3,0);
```

```
set_d(t4,0.9,0);

set_d(t5,0.2,0);

set_d(t6,0.4,0);
set_d(t7,2.4,0);

set_i(t8,2,0);
set_i(t9,12,0);

set_d(t10,1.0,0);
set_d(t11,1.0,0);
set_d(t12,1.0,0);

T_approx_chain(Rows,Columns,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,
                &Bzl,&Bzc,&Br,&Bwl,&Bwc,&Ll0,&Lc0,&Ll1,&Lc1,&order);
nob = length_tuple(Bzl);
nol = length_tuple(Ll0);
/* draw lines and arcs */
set_i(WindowHandleTuple,WindowHandle,0);
set_line_width(WindowHandle,4);
if (nob>0) T_disp_arc(Bzl,Bzc,Br,Bwl,Bwc);
set_line_width(WindowHandle,1);
if (nol>0) T_disp_line(WindowHandleTuple,Ll0,Lc0,Ll1,Lc1);
```

Ergebnis

`approx_chain` liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`sobel_amp`, `edges_image`, `get_region_contour`, `threshold`, `hysteresis_threshold`

Nachfolger

`set_line_width`, `disp_arc`, `disp_line`

Alternativen

`get_region_polygon`, `approx_chain_simple`

Siehe auch

`get_region_chain`, `smallest_circle`, `disp_circle`, `disp_line`

Modul

Foundation

approx_chain_simple (: : Row, Column : ArcCenterRow, ArcCenterCol, ArcAngle, ArcBeginRow, ArcBeginCol, LineBeginRow, LineBeginCol, LineEndRow, LineEndCol, Order)

Approximation einer Kontur durch Bögen und Linien.

`approx_chain_simple` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

Die Koordinaten einer Kurve werden durch eine Folge von Linien und Kreisbögen approximiert.

Das Ergebnis des Verfahren wird getrennt nach Bögen und Linien zurückgegeben. Ist man an der Reihenfolge der Segmente interessiert, so können die einzelnen Ergebniselemente sukzessive aus den Ergebnistupeln ausgelesen

werden, wobei dem Rückgabeparameter Order die Reihenfolge entnommen werden kann (0: nächstes Element ist nächstes Liniensegment, 1: nächstes Element ist nächstes Bogensegment).

`approx_chain_simple` verhält sich ähnlich wie `approx_chain` nur bei `approx_chain_simple` sind die fehlende Parameter intern wie folgt belegt: MinWidthCoord = 1.0, MaxWidthCoord = 3.0, ThreshStart = 0.5, ThreshEnd = 0.9, ThreshStep = 0.3, MinWidthSmooth = 1.0, MaxWidthSmooth = 3.0, MinWidthCurve = 2, MaxWidthCurve = 9, Weight1 = 1.0, Weight2 = 1.0, Weight3 = 1.0.

Parameter

- ▷ **Row** (input_control) . . . . . point.y-array ⇝ *integer*
  Zeilenwert der Kontur.
  Defaultwert: 32
- ▷ **Column** (input_control) . . . . . point.x-array ⇝ *integer*
  Spaltenwert der Kontur.
  Defaultwert: 32
- ▷ **ArcCenterRow** (output_control) . . . . . arc.center.y-array ⇝ *integer*
  Zeile des Mittelpunkts eines Bogens.
- ▷ **ArcCenterCol** (output_control) . . . . . arc.center.x-array ⇝ *integer*
  Spalte des Mittelpunkts eines Bogens.
- ▷ **ArcAngle** (output_control) . . . . . arc.angle.rad-array ⇝ *real*
  Winkel eines Bogens.
- ▷ **ArcBeginRow** (output_control) . . . . . arc.begin.y-array ⇝ *integer*
  Zeile des Startpunkts eines Bogens.
- ▷ **ArcBeginCol** (output_control) . . . . . arc.begin.x-array ⇝ *integer*
  Spalte des Startpunkts eines Bogens.
- ▷ **LineBeginRow** (output_control) . . . . . line.begin.y-array ⇝ *integer*
  Zeile des Startpunkts einer Linie.
- ▷ **LineBeginCol** (output_control) . . . . . line.begin.x-array ⇝ *integer*
  Spalte des Startpunkts einer Linie.
- ▷ **LineEndRow** (output_control) . . . . . line.end.y-array ⇝ *integer*
  Zeile des Endpunkts einer Linie.
- ▷ **LineEndCol** (output_control) . . . . . line.end.x-array ⇝ *integer*
  Spalte des Endpunkts einer Linie.
- ▷ **Order** (output_control) . . . . . integer-array ⇝ *integer*
  Reihenfolge von Linien- (Wert 0) und Bogensegmenten (Wert1).

Beispiel

```
/* read edge image */
read_image(&Image,"fig1_kan");
/* construct edge region */
hysteresis_threshold(Image,&RK1,64,255,40,1);
connection(RK1,&Rand);
/* fetch chain code */
T_get_region_contour(Rand,&Rows,&Columns);
firstline = get_i(Tline,0);
firstcol = get_i(Tcol,0);
/* approximation with lines and circular arcs */
T_approx_chain_simple(Rows,Columns,
                      &Bzl,&Bzc,&Br,&Bwl,&Bwc,&Ll0,&Lc0,&Ll1,&Lc1,&order);
nob = length_tuple(Bzl);
nol = length_tuple(Ll0);
/* draw lines and arcs */
set_i(WindowHandleTuple,WindowHandle,0);
set_line_width(WindowHandle,4);
if (nob>0) T_disp_arc(Bzl,Bzc,Br,Bwl,Bwc);
set_line_width(WindowHandle,1);
if (nol>0) T_disp_line(WindowHandleTuple,Ll0,Lc0,Ll1,Lc1);
```

Ergebnis

`approx_chain_simple` liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`sobel_amp`, `edges_image`, `get_region_contour`, `threshold`, `hysteresis_threshold`

Nachfolger

`set_line_width`, `disp_arc`, `disp_line`

Alternativen

`get_region_polygon`, `approx_chain`

Siehe auch

`get_region_chain`, `smallest_circle`, `disp_circle`, `disp_line`

Modul

Foundation

`clear_all_bar_code_models` (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Barcode-Modelle und Freigabe des verwendeten Speichers.

`clear_all_bar_code_models` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_bar_code_models` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_bar_code_models` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_bar_code_models` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_bar_code_models` liefert den Wert TRUE, falls alle von HALCON verwalteten Barcode-Modelle korrekt freigegeben wurden. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Bar Code

`clear_all_barriers` (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Zerstören aller Barrieren-Synchronisationsobjekte.

clear_all_barriers ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_barriers nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_barriers existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_barriers *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_barriers liefert den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_calib_data (: : :)

Freigeben des Speichers aller im Speicher angelegten Kalibrierdatenmodelle.

clear_all_calib_data ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_calib_data nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_calib_data existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_calib_data *darf nicht* in Applikationen verwendet werden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Calibration

clear_all_camera_setup_models (: : :)

Freigeben des Speichers aller im Speicher angelegten Kameraaufbaumodelle.

clear_all_camera_setup_models ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_camera_setup_models nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_camera_setup_models existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_camera_setup_models *darf nicht* in Applikationen verwendet werden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Calibration

clear_all_class_gmm (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Gaußschen Mischverteilungsmodelle.

clear_all_class_gmm ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_class_gmm nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_class_gmm existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_class_gmm *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_class_gmm liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_class_knn (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löscht alle k-NN-Klassifikatoren.

clear_all_class_knn ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_class_knn nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_class_knn existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_class_knn *darf nicht* in Applikationen verwendet werden.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert clear_all_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

Foundation

clear_all_class_lut (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Look-Up-Table Klassifikatoren.

clear_all_class_lut ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_class_lut nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_class_lut existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_class_lut *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_class_lut liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_class_mlp (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller mehrschichtiger Perzeptrons.

clear_all_class_mlp ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_class_mlp nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_class_mlp existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_class_mlp *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_class_mlp liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_class_svm (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Support-Vektor-Maschines.

clear_all_class_svm ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_class_svm nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_class_svm existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_class_svm *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_class_svm liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_class_train_data (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löscht alle Trainingsdaten.

clear_all_class_train_data ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_class_train_data nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_class_train_data wird für die „reset program“-Funktionalität in HDevelop verwendet.

Ergebnis

clear_all_class_train_data gibt den Wert TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_color_trans_luts (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löscht alle Look-Up-Tabellen für die Farbraumtransformation.

clear_all_color_trans_luts ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten

clear_all_color_trans_luts nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

Nach dem Ausführen des Operators clear_all_color_trans_luts werden alle Handles der Look-Up-Tabellen ungültig. clear_all_color_trans_luts existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_color_trans_luts *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_color_trans_luts liefert den Wert TRUE, falls alle Look-Up-Tabellen erfolgreich gelöscht wurden. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_component_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller Komponentenmodelle.

clear_all_component_models ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_component_models nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_component_models existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_component_models *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_component_models liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

clear_all_conditions (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Zerstören aller Condition-Synchronisationsobjekte.

clear_all_conditions ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_conditions nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_conditions` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_conditions` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_conditions` liefert den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_data_code_2d_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller 2D-Datacode-Modelle und Freigabe des verwendeten Speichers.

`clear_all_data_code_2d_models` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_data_code_2d_models` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_data_code_2d_models` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_data_code_2d_models` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_data_code_2d_models` liefert den Wert TRUE, falls alle von HALCON verwalteten 2D-Datacode-Modelle korrekt freigegeben wurden. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Data Code

clear_all_deformable_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller Formmodelle.

`clear_all_deformable_models` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_deformable_models` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_deformable_models` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_deformable_models` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_deformable_models` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

clear_all_descriptor_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigeben des Speichers aller im Speicher angelegten Deskriptormodelle.

`clear_all_descriptor_models` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_descriptor_models` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Ergebnis

Ist der Operator erfolgreich dann liefert `clear_descriptor_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

clear_all_events (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Event-Synchronisationsobjekte.

`clear_all_events` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_events` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_events` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_events` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_events` gibt den Wert TRUE zurück. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_lexica (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Lexika.

clear_all_lexica ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_lexica nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_lexica existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_lexica *darf nicht* in Applikationen verwendet werden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

OCR/OCV

clear_all_matrices (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löscht alle Matrizen aus dem Speicher.

clear_all_matrices ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_matrices nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_matrices existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_matrices *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_matrices gibt immer den Wert TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_metrology_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Messmodelle und Freigabe des verwendeten Speichers.

clear_all_metrology_models ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_metrology_models nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_metrology_models existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_metrology_models *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_metrology_models liefert den Wert TRUE, falls alle von HALCON verwalteten Messmodelle korrekt freigegeben wurden. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

2D Metrology

clear_all_mutexes (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Mutex-Synchronisationsobjekte.

clear_all_mutexes ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_mutexes nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_mutexes existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_mutexes *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_mutexes liefert den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_ncc_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller NCC-Modelle.

clear_all_ncc_models ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_ncc_models nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_ncc_models existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_ncc_models *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_ncc_models liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

clear_all_object_model_3d (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigeben des Speichers aller 3D-Objektmodelle.

clear_all_object_model_3d ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_object_model_3d nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_object_model_3d existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_object_model_3d *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_object_model_3d liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

3D Metrology

clear_all_ocr_class_knn (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller OCR-Klassifikatoren.

clear_all_ocr_class_knn ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_ocr_class_knn nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_ocr_class_knn` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_ocr_class_knn` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_ocr_class_knn` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

OCR/OCV

clear_all_ocr_class_mlp (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller OCR-Klassifikatoren.

`clear_all_ocr_class_mlp` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_ocr_class_mlp` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_ocr_class_mlp` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_ocr_class_mlp` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_ocr_class_mlp` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

OCR/OCV

clear_all_ocr_class_svm (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller auf SVM basierenden OCR-Klassifikatoren.

`clear_all_ocr_class_svm` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_ocr_class_svm` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_ocr_class_svm` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_ocr_class_svm` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_ocr_class_svm` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

OCR/OCV

clear_all_sample_identifiers (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigeben des Speichers aller Sample-Identifikatoren.

`clear_all_sample_identifiers` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_sample_identifiers` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_sample_identifiers` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_sample_identifiers` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_sample_identifiers` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

clear_all_scattered_data_interpolators (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Interpolatoren für verstreute Daten.

`clear_all_scattered_data_interpolators` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_scattered_data_interpolators` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_scattered_data_interpolators` existiert ausschließlich, um die

Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_scattered_data_interpolators` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_scattered_data_interpolators` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_serialized_items (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller aktuell vorhandenen serialisierten Elemente.

`clear_all_serialized_items` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_serialized_items` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_serialized_items` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_serialized_items` *darf nicht* in Applikationen verwendet werden.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `clear_all_serialized_items` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_shape_model_3d (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller 3D-Formmodelle.

`clear_all_shape_model_3d` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `close_all_framegrabbers` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_shape_model_3d` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_shape_model_3d` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_shape_model_3d` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

3D Metrology

clear_all_shape_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller Formmodelle.

`clear_all_shape_models` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_shape_models` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_shape_models` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_shape_models` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_shape_models` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

clear_all_sheet_of_light_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Lichtschnitt-Modelle und Freigabe des verwendeten Speichers.

`clear_all_sheet_of_light_models` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_sheet_of_light_models` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_sheet_of_light_models` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_sheet_of_light_models` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_sheet_of_light_models` liefert den Wert TRUE, falls alle von HALCON verwalteten Lichtschnitt-Modelle korrekt freigegeben wurden. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

3D Metrology

clear_all_stereo_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigeben des Speichers aller im Speicher angelegten Stereomodelle.

clear_all_stereo_models ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_stereo_models nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_stereo_models existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_stereo_models *darf nicht* in Applikationen verwendet werden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

3D Metrology

clear_all_surface_matching_results (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller Ergebnisse des oberflächenbasierten Matchings.

clear_all_surface_matching_results ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_surface_matching_results nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_surface_matching_results existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_surface_matching_results *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_surface_matching_results liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

3D Metrology

clear_all_surface_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller Oberflächenmodelle.

clear_all_surface_models ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_surface_models nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_surface_models existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_surface_models *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_surface_models liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

3D Metrology

clear_all_templates (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller Templates.

clear_all_templates ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_templates nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_templates existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_templates *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_templates liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

clear_all_text_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigeben aller Textmodelle.

clear_all_text_models ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_text_models nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_text_models existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_text_models *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_text_models gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

OCR/OCV

clear_all_text_results (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigeben aller Ergebnisse der Textsegmentierung.

clear_all_text_results ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_text_results nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

clear_all_text_results existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. clear_all_text_results *darf nicht* in Applikationen verwendet werden.

Ergebnis

clear_all_text_results gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

clear_all_training_components (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller Komponenten-Trainingsergebnisse.

clear_all_training_components ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten clear_all_training_components nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_training_components` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_training_components` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_training_components` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

clear_all_variation_models (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigabe des Speichers aller Variationsmodelle.

`clear_all_variation_models` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `clear_all_variation_models` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`clear_all_variation_models` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `clear_all_variation_models` *darf nicht* in Applikationen verwendet werden.

Ergebnis

`clear_all_variation_models` liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Matching

close_all_bg_esti (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löscht alle Hintergrundschätzer-Datensätze.

`close_all_bg_esti` ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten `close_all_bg_esti` nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`close_all_bg_esti` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `close_all_bg_esti` *darf nicht* in Applikationen verwendet werden.

Ergebnis

Gelingt es, die Hintergrundschätzer zu schließen, liefert `close_all_bg_esti` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

close_all_class_box (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Klassifikatoren.

close_all_class_box ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten close_all_class_box nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`close_all_class_box` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `close_all_class_box` *darf nicht* in Applikationen verwendet werden.

Ergebnis

Gelingt es, die Klassifikatoren zu schließen, liefert `close_all_class_box` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

close_all_files (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Schließt alle offenen Dateien.

close_all_files ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten close_all_files nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

`close_all_files` existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. `close_all_files` *darf nicht* in Applikationen verwendet werden.

Ergebnis

Gelingt es, die Dateien zu schließen, liefert `close_all_files` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

close_all_framegrabbers (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Schließen aller Bildeinzugsgeräte.

close_all_framegrabbers ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt Neue Applikationen sollten close_all_framegrabbers nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

close_all_framegrabbers existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. close_all_framegrabbers *darf nicht* in Applikationen verwendet werden.

Aus Sicht einer Anwendung mit mehreren Threads bilden alle Bildeinzugsoperatoren (wie in open_framegrabber aufgelistet) eine eigene Gruppe innerhalb der info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers exklusiv ausgeführt werden.

D.h., info_framegrabber, open_framegrabber, close_framegrabber und close_all_framegrabbers verhindern die gleichzeitige Ausführung anderer Operatoren innerhalb der Gruppe, laufen aber mit allen nicht exklusiven Operatoren außerhalb der Gruppe parallel.

Ergebnis

Gelingt es, alle Bildeinzugsgeräte zu schließen, liefert close_all_framegrabbers den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

close_all_measures (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller Messobjekte.

close_all_measures ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten close_all_measures nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

close_all_measures existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. close_all_measures *darf nicht* in Applikationen verwendet werden.

Ergebnis

close_all_measures liefert immer TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

1D Metrology

close_all_ocrs (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Löschen aller OCR-Klassifikatoren.

close_all_ocrs ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten close_all_ocrs nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

close_all_ocrs existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. close_all_ocrs *darf nicht* in Applikationen verwendet werden.

Ergebnis

Gelingt es, die OCR-Klassifikatoren zu schließen, liefert close_all_ocrs den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

OCR/OCV

close_all_ocvs (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Freigeben aller OCV-Tools.

close_all_ocvs ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten close_all_ocvs nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein.

.

Achtung

close_all_ocvs existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. close_all_ocvs *darf nicht* in Applikationen verwendet werden.

Ergebnis

close_all_ocvs liefert immer den Wert TRUE

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

OCR/OCV

close_all_serials (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Schließen aller seriellen Geräte.

close_all_serials ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten close_all_serials nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein

.

Achtung

close_all_serials existiert ausschließlich, um die Funktionalität zum Zurücksetzen des Programmes in HDevelop zu implementieren. close_all_serials *darf nicht* in Applikationen verwendet werden.

Ergebnis

close_all_serials liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

close_all_sockets (: : :)

Dieser Operator ist funktionslos. Er hatte die folgende Funktion: Schließen aller Socket-Verbindungen.

close_all_sockets ist obsolet und funktionslos und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten close_all_sockets nicht verwenden. Der Operator wird in zukünftigen HALCON Versionen nicht mehr enthalten sein

.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

filter_kalman (: : Dimension, Model, Measurement,
PredictionIn : PredictionOut, Estimate)

Schätzung des aktuellen Zustands eines Systems mittels Kalman-Filterung.

filter_kalman ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

filter_kalman liefert eine Schätzung des aktuellen Zustands (oder auch eine Prädiktion des künftigen Zustandes) eines diskreten, stochastisch gestörten, linearen Systems. Kalman-Filter werden im Bereich der Bildverarbeitung insbesondere erfolgreich bei der Bildfolgenanalyse eingesetzt (Hintergrund-Erkennung, Fahrbahnverfolgung mittels Linienverfolgung oder mittels Regionen-Analyse etc.). Im folgenden wird zunächst eine kurze Einführung in die Theorie der Kalman-Filter gegeben. Im Anschluss daran wird dann die Routine filter_kalman selbst beschrieben.

KALMAN-FILTER: Ein diskretes, stochastisch gestörtes, lineares System wird durch folgende Kenngrößen charakterisiert:

- Status $x(t)$: Beschreibt den momentanen Zustand des Systems (Geschwindigkeiten, Temperaturen,...)
- Stellgröße $u(t)$: Eingaben von außen in das System
- Messwerte $y(t)$: Messwerte, die durch Beobachtung des Systems gewonnen werden. Sie spiegeln den Systemzustand (oder zumindest Teile davon) wider.
- Eine Ausgabefunktion, die die Abhängigkeit der Messwerte vom Status beschreibt.
- Eine Übergangsfunktion, die angibt, wie sich der Status in Abhängigkeit von der Zeit, seinem momentanen Wert und den Stellgrößen verändert.

Die Ausgabefunktion und die Übergangsfunktion sind linear. Ihre Anwendung kann daher als Matrix-Multiplikation geschrieben werden.

Die Übergangsfunktion wird durch die Übergangsmatrix $A(t)$ und die Stellmatrix $G(t)$ beschrieben, die Ausgangsfunktion durch die Messmatrix $C(t)$. $A(t)$ charakerisiert dabei die Abhängigkeit des neuen vom alten Systemstatus, $G(t)$ die Abhängigkeit von den Stellgrößen. In der Praxis ist es selten möglich (oder zumindest zu aufwendig), ein reales System und sein Verhalten in sich geschlossen exakt zu beschreiben. Insbesondere beschränkt man sich in der Regel auf eine relativ kleine Zahl von Variablen zur Darstellung des Systemverhaltens. Daraus resultiert ein Fehler, der sogenannte Systemfehler (der auch Systemstörung genannt wird) $v(t)$. Auch die Ausgabefunktion ist im allgemeinen nicht exakt. Jede Messung ist fehlerbehaftet. Die Messfehler seien mit $w(t)$ bezeichnet. Damit ergeben sich folgende Systemgleichungen:

$$\begin{aligned} x(t+1) &= A(t)x(t) + G(t)u(t) + v(t) \\ y(t) &= c(t)x(t) + w(t) \end{aligned}$$

Die System- und Messfehler $v(t)$ und $w(t)$ sind unbekannt. Sie werden bei den im Zusammenhang mit Kalman-Filterung betrachteten Systemen als gaußverteilte Zufallsvektoren betrachtet (daher die Bezeichnung „stochastisch gestörte Systeme“). Somit wird das System berechenbar, wenn die zu $v(t)$ und $w(t)$ gehörigen Erwartungswerte und Kovarianzmatrizen bekannt sind.

Die Schätzung des Systemzustandes erfolgt wie in der Gauß-Markov- Schätzung. Der Kalman-Filter ist jedoch ein rekursiver Algorithmus, der sich nur auf die aktuellen Messwerte $y(t)$ und den letzten Status $x(t)$ stützt. Letzterer enthält implizit das Wissen um früher bereits angefallenen Messwerte.

Für den Startwert $x(0)$ ist ein geeigneter Schätzwert x_0 anzugeben, der als Erwartungswert einer Zufallsvariable für $x(0)$ aufgefasst wird. Deren Fehler habe die Erwartung 0 und die Kovarianzmatrix P_0, die ebenfalls anzugeben ist. Zum Zeitpunkt t sei der Erwartungswert der Störungen $v(t)$ und $w(t)$ jeweils 0 und ihre Kovarianzmatrizen seien $Q(t)$ und $R(t)$. $x(t)$, $v(t)$ und $w(t)$ werden meist als unkorreliert angenommen (es sind auch beliebige Rauschprozesse modellierbar - die Erstellung der benötigten Matrizen durch den Anwender ist dann allerdings deutlich aufwendiger). An die gesuchten Schätzwerte x^t werden folgende Bedingungen gestellt:

- Die Schätzwerte x^t hängen linear vom tatsächlichen Wert $x(t)$ und der Messwertfolge $y(0), y(1), \cdots, y(t)$ ab.
- x^t sei erwartungstreu, d.h. $Ex^t = Ex(t)$.
- Als Gütekriterium erfülle x^t das Kriterium der minimalen Varianz, d.h. die Varianz des Schätzfehlers, definiert als $x(t) - x^t$, sei minimal.

Der Kalman-Filter führt nach der Initialisierung

$$\hat{x}(0) = x_0, \hat{P}(0) = P_0$$

zu jedem Zeitpunkt t folgende Berechnungsschritte aus:

$$\begin{aligned} (K-III) \qquad K(t) &= \frac{\hat{P}(t)C'(t)}{C(t)\hat{P}(t)C'(t)+R(t)} \\ (K-IV) \qquad x^t &= \hat{x}(t) + K(t)(y(t) - C(t)\hat{x}(t)) \\ (K-V) \qquad \tilde{P}(t) &= \hat{P}(t) - K(t)C(t)\hat{P}(t) \\ (K-I) \qquad \hat{x}(t+1) &= A(t)x^t + G(t)u(t) \\ (K-II) \qquad \hat{P}(t+1) &= A(t)\tilde{P}(t)A'(t) + Q(t) \end{aligned}$$

Dabei ist $\tilde{P}(t)$ die Kovarianzmatrix des Schätzfehlers, $\hat{x}(t)$ der Extrapolations- bzw. Prädiktionswert des Zustands, $\hat{P}(t)$ die Kovarianzen des Prädiktionsfehlers $\hat{x} - x$, K die Verstärkungsmatrix (das sogenannte Kalman-gain) und

X' die Transponierte einer Matrix X. Hingewiesen sei an dieser Stelle auch auf die Möglichkeit der Vorhersage (Prädiktion) des Folgezustandes durch Gleichung (K-I). Dies ist z.B. in der Bildverarbeitung mitunter sehr nützlich, um „regions of interest“ im Folgebild zu bestimmen.

Wie oben erwähnt, ist die Modellierung beliebiger Rauschprozesse mitunter sehr aufwendig. Ist beispielsweise das System- und das Messrauschen korreliert mit entsprechender Kovarianzmatrix L, sind die Gleichungen für das Kalman-gain und die Fehler-Kovarianzmatrix zu modifizieren:

$$\begin{array}{lll} (K-III) & K(t) & = \frac{\hat{P}(t)C'(t)+L(t)}{C(t)\hat{P}(t)+C(t)l(t)+L'C'(t)+R(t)} \\ (K-V) & \tilde{P}(t) & = (\hat{P}(t)-K(t)C(t)\hat{P}(t))\hat{P}(t)-K(t)L(t) \end{array}$$

Es ist nun Aufgabe des Benutzers, die linearen Systemgleichungen (K-I) bis (K-V) - in Abhängigkeit vom konkreten Problem - aufzustellen. Er muss also ein der Problemlösung zugrundeliegendes mathematisches Modell entwickeln. Statistische Größen, die die Ungenauigkeiten seines Systems und die die zu erwartenden Messfehler beschreiben, müssen dabei geschätzt werden, wenn sie sich nicht exakt berechnen lassen. Im einzelnen sind also folgende Schritte nötig:

1. Aufstellen eines mathematischen Modells
2. Auswahl charakteristischer Zustandsvariablen
3. Aufstellen der Gleichungen, die die Änderung dieser Zustandsvariablen beschreiben und Linearisierung derselben (Matrizen A und G)
4. Aufstellung der Gleichungen, die die Abhängigkeit der Messwerte des Systems von den Zustandsvariablen beschreiben und Linearisierung derselben (Matrix C)
5. Aufstellung oder Schätzung statistischer Abhängigkeiten zwischen den Systemstörungen (Matrix Q)
6. Aufstellung oder Schätzung statistischer Abhängigkeiten zwischen den Messfehlern (Matrix R)
7. Initialisierung des Anfangszustandes

Die Initialisierung des Systems (Punkt 7) verlangt dabei, wie oben erwähnt, die Angabe einer Schätzung x_0 des Systemzustandes zum Zeitpunkt 0 und der zugehörigen Kovarianzmatrix P_0. Ist der exakte Anfangszustand nicht bekannt, empfiehlt es sich, die Komponenten des Vektors x_0 auf die Mittelwerte der jeweiligen Wertebereiche zu setzen und hohe Werte in P_0 einzutragen (in der Größenordnung der Quadrate der Wertebereiche). Nach einigen Iterationen (wenn die Zahl der insgesamt akkumulierten Messwerte größer geworden ist als die Zahl der Systemgrößen), erhält man auch hier brauchbare Werte. Ist umgekehrt der Anfangszustand exakt bekannt, sind alle Einträge von P_0 auf Null zu setzen - P_0 beschreibt ja die Kovarianzen des Fehlers zwischen dem Schätzwert x_0 und dem tatsächlichen Wert $x(0)$.

DIE FILTER-ROUTINE:

Eine Kalman-Filterung hängt von einer Reihe von Daten ab, die sich in vier Gruppen gliedern lassen:

Modellparameter: Übergangsmatrix A, Steuermatrix G mit der Stellgröße u und Messmatrix C)

Modellstochastik: Systemfehler-Kovarianzmatrix Q, Systemfehler-Messfehler- Kovarianzmatrix L und Messfehler-Kovarianzmatrix R)

Messvektor: y

Systemvergangenheit: Extrapolationsvektor $\hat{x}$ und Extrapolationsfehler-Kovarianzmatrix $\hat{P}$

Viele Systeme kommen dabei ohne Eingaben „von außen“ und damit ohne G und u aus. Außerdem sind im Normalfall System- und Messfehler unkorreliert (L entfällt).

Konkret werden die benötigten Daten der Routine durch folgende Parameter übergeben:

Dimension: Dieser Parameter enthält die Dimensionen von Status-, Mess- und Stellvektor. `Dimension` ist somit ein Vektor *[n,m,p]*, wobei n die Anzahl der Zustandsvariablen, *'m'* die Anzahl der Messwerte und *'p'* die Anzahl der Stellglieder ist. Für ein System ohne deterministische Steuerung (d.h. ohne Einfluss „von außen“) ist folglich *[n,m,0]* zu übergeben.

Model: Dieser Parameter enthält hintereinandergehängt die zeilenweise linearisierten Matrizen (Vektoren) *'A,C,Q,G,u'* und (gegebenenfalls) *'L'*. `Model` ist also ein Vektor der Länge $n \times n + n \times m + n \times n + n \times p + p[+n \times m]$. Der letzte Summand entfällt, falls System- und Messfehler unkorreliert sind, d.h. kein *'L'* anzugeben ist.

Measurement: Dieser Parameter enthält hintereinandergehängt die zeilenweise linearisierte Matrix *'R'* und den Messvektor *'y'*. `Measurement` ist also ein Vektor der Dimension $m \times m + m$.

PredictionIn / PredictionOut: Diese Parameter enthalten hintereinandergehängt die zeilenweise linearisierte Matrix $\hat{P}$ (die Extrapolationsfehler-Kovarianzmatrix) und den Extrapolationsvektor $\hat{x}$, sind also Vektoren der Länge $n \times n + n$. `PredictionIn` ist ein Eingabeparameter, der $\hat{P}(t)$ und $\hat{x}(t)$ zum aktuellen Zeitpunkt t enthalten muss. In `PredictionOut` liefert die Routine dann die entsprechenden Vorhersagen $\hat{P}(t+1)$ und $\hat{x}(t+1)$.

Estimate: In Diesem Parameter liefert die Routine hintereinandergehängt die zeilenweise linearisierte Matrix $\tilde{P}$ (die Schätzfehler-Kovarianzmatrix) und den geschätzten Zustand $\tilde{x}$. `Estimate` ist daher ein Vektor der Länge $n \times n + n$.

Zu beachten ist dabei, dass Kovarianzmatrizen ($Q, R, \hat{P}, \tilde{P}$) naturgemäß symmetrisch sein müssen.

Parameter

▷ **Dimension** (input_control) . . . . . integer-array ⇝ *integer*
Die Dimensionen von Status-, Mess- und Stellvektor.
Defaultwert: [3,1,0]
Typischer Wertebereich: $0 \leq$ `Dimension` ≤ 30

▷ **Model** (input_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierten Matrizen A, C, Q, eventuell G und u und gegebenenfalls L.
Defaultwert: [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
Typischer Wertebereich: $0.0 \leq$ `Model` ≤ 10000.0

▷ **Measurement** (input_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix R und der Messvektor y.
Defaultwert: [1.2,1.0]
Typischer Wertebereich: $0.0 \leq$ `Measurement` ≤ 10000.0

▷ **PredictionIn** (input_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix $\hat{P}$ (die Extrapolationsfehler- kovarianzen) und der Extrapolationsvektor $\hat{x}$.
Defaultwert: [0.0,0.0,0.0,0.0,180.5,0.0,0.0,0.0,0.0,100.0,0.0,0.0,100.0,0.0]
Typischer Wertebereich: $0.0 \leq$ `PredictionIn` ≤ 10000.0

▷ **PredictionOut** (output_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix P∗ (die Extrapolationsfehler- kovarianzen) und der Extrapolationsvektor $\hat{x}$

▷ **Estimate** (output_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix $\tilde{P}$ (die Schätzfehlerkovarianzen) und der geschätzte Zustand $\tilde{x}$.

Beispiel

```
* Typical procedure:
* To initialize the variables, which describe the model, e.g., with
read_kalman('kalman.init',Dim,Mod,Meas,Pred)

* Generation of the first measurements (typical of the first image of an
* image series) with an appropriate problem-specific procedure (there is a
* fictitious procedure extract_features in example):
* extract_features(Image1,Meas,Meas1)

* first Kalman-Filtering:
filter_kalman(Dim,Mod,Meas1,Pred,Pred1,Est1)

* To use the estimate value (if need be the prediction too)
* with a problem-specific procedure (here use_est):
* use_est(Est1)

* To get the next measurements (e.g. from the next image):
```

```
* extract_next_features(Image2,Meas1,Meas2)

* if need be Update of the model parameter (a constant model)
* second Kalman-Filtering:
filter_kalman(Dim,Mod,Meas2,Pred1,Pred2,Est2)
* use_est(Est2)
* extract_next_features(Image3,Meas2,Meas3)
* etc.
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `filter_kalman` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_kalman`

Nachfolger

`update_kalman`

Siehe auch

`read_kalman`, `update_kalman`

Literatur

W.Hartinger: „Entwurf eines anwendungsunabhängigen Kalman-Filters mit Untersuchungen im Bereich der Bildfolgenanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

R.E.Kalman: „A New Approach to Linear Filtering and Prediction Problems“; Transactions ASME, Ser.D: Journal of Basic Engineering; Vol. 82, S.34-45; 1960.

R.E.Kalman, P.l.Falb, M.A.Arbib: „Topics in Mathematical System Theory“; McGraw-Hill Book Company, New York; 1969.

K-P. Karmann, A.von Brandt: „Moving Object Recognition Using an Adaptive Background Memory“; Time-Varying Image Processing and Moving Object Recognition 2 (ed.: V. Cappellini), Proc. of the 3rd Interantional Workshop, Florence, Italy, May, 29th - 31st, 1989; Elsevier, Amsterdam; 1990.

Modul

Foundation

intersection_ll (: : RowA1, ColumnA1, RowA2, ColumnA2, RowB1, ColumnB1, RowB2, ColumnB2 : Row, Column, IsParallel)

Berechnen des Schnittpunktes zwischen zwei Geraden.

`intersection_ll` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen den Operator `intersection_lines` aus dem Kapitel Tools / Geometrie verwenden.

`intersection_ll` berechnet die Koordinaten des Schnittpunktes zwischen zwei Geraden. Als Eingabe werden je zwei Punkte auf den Geraden erwartet (`RowA1`,`ColumnA1`, `RowA2`,`ColumnA2`) und (`RowB1`,`ColumnB1`, `RowB2`,`ColumnB2`). Das Ergebnis wird dann in `Row` und `Column` übergeben. Falls die Geraden parallel zueinander sind, sind die Werte von `Row` und `Column` undefiniert und `IsParallel` hat den Wert 1. Andernfalls liefert `IsParallel` den Wert 0.

Achtung

Falls die Geraden parallel zueinander sind, dann sind die Werte von `Row` und `Column` undefiniert.

Parameter

- ▷ **RowA1** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der ersten Geraden.
- ▷ **ColumnA1** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der ersten Geraden.
- ▷ **RowA2** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der ersten Geraden.
- ▷ **ColumnA2** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der ersten Geraden.
- ▷ **RowB1** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der zweiten Geraden.
- ▷ **ColumnB1** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate es ersten Punktes der zweiten Geraden.
- ▷ **RowB2** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der zweiten Geraden.
- ▷ **ColumnB2** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der zweiten Geraden.
- ▷ **Row** (output_control) . . . . . point.y(-array) ⇝ *real*
  Zeilenkoordinate des Schnittpunktes.
- ▷ **Column** (output_control) . . . . . point.x(-array) ⇝ *real*
  Spaltenkoordinate des Schnittpunktes.
- ▷ **IsParallel** (output_control) . . . . . number(-array) ⇝ *integer*
  Sind die zwei Geraden parallel?

Beispiel

```
dev_set_color ('black')
RowLine1 := 350
ColLine1 := 250
RowLine2 := 300
ColLine2 := 300
Rows := 300
Columns := 50
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
n := 0
for Rows := 40 to 200 by 4
  dev_set_color ('red')
  disp_line (WindowHandle, Rows, Columns, Rows+n, Columns+n)
  intersection_ll (Rows, Columns, Rows+n, Columns+n, RowLine1, ColLine1, \
                   RowLine2, ColLine2, Row, Column, IsParallel)
  dev_set_color ('blue')
  disp_line (WindowHandle, Row, Column-2, Row, Column+2)
  disp_line (WindowHandle, Row-2, Column, Row+2, Column)
  n := n+8
endfor
```

Ergebnis

`intersection_ll` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

partition_lines (: : RowBeginIn, ColBeginIn, RowEndIn, ColEndIn, Feature, Operation, Min, Max : RowBeginOut, ColBeginOut, RowEndOut, ColEndOut, FailRowBOut, FailColBOut, FailRowEOut, FailColEOut)

Partitionierung von Linien gemäß verschiedener Kriterien.

partition_lines ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

partition_lines teilt Linien anhand verschiedener Kriterien in zwei Mengen auf. Für jede Eingabelinie werden die angegebenen Merkmale (Feature) berechnet. Wenn jedes (Operation = *'and'*) oder mindestens eines (Operation = *'or'*) der so berechneten Merkmale in den vorgegebenen Grenzen (Min,Max)l liegt, wird die Linie in die erste Menge (Parameter RowBeginOut bis ColEndOut) übernommen, andernfalls in die zweite Menge (Parameter FailRowBOut bis FailColEOut). Mit den Defaultwerten *'min'* beziehungsweise *'max'* der Parameter Min und Max werden diese Grenzen nach unten beziehungsweise oben offen gelassen.

Bedingung: $Min_i \leq Feature_i(Linie) \leq Max_i$

Mögliche Werte für Feature:

'length' (euklidische) Länge der Linie

'row' Zeilenindex des Schwerpunkts

'column' Spaltenindex des Schwerpunkts

'phi' Orientierung der Linie ($-\frac{\pi}{2} < \varphi \leq \frac{\pi}{2}$)

Achtung

Wird nur ein Merkmal verwendet, dann ist der Wert von Operation bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie übergeben werden.

Parameter

- ▷ **RowBeginIn** (input_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **ColBeginIn** (input_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **RowEndIn** (input_control) ..... line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **ColEndIn** (input_control) ..... line.end.x-array ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **Feature** (input_control) ..... string(-array) ⇝ *string*
  Gewünschte Merkmale.
  Werteliste: Feature ∈ {'length', 'row', 'column', 'phi'}
- ▷ **Operation** (input_control) ..... string ⇝ *string*
  Gewünschte Verknüpfung.
  Werteliste: Operation ∈ {'and', 'or'}
- ▷ **Min** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Untere Grenzen der Merkmale oder 'min'.
  Defaultwert: 'min'
- ▷ **Max** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Obere Grenzen der Merkmale oder 'max'.
  Defaultwert: 'max'
- ▷ **RowBeginOut** (output_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Linien, die die Bedingung(en) erfüllen.
- ▷ **ColBeginOut** (output_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Linien, die die Bedingung(en) erfüllen.
- ▷ **RowEndOut** (output_control) ..... line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Linien, die die Bedingung(en) erfüllen.
- ▷ **ColEndOut** (output_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Linien, die die Bedingung(en) erfüllen.

▷ **FailRowBOut** (output_control) ..... line.begin.y-array ⇝ *integer*
Zeilenkoordinaten der Anfangspunkte der Linien, die die Bedingung(en) nicht erfüllen.

▷ **FailColBOut** (output_control) ..... line.begin.x-array ⇝ *integer*
Spaltenkoordinaten der Anfangspunkte der Linien, die die Bedingung(en) nicht erfüllen.

▷ **FailRowEOut** (output_control) ..... line.end.y-array ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Linien, die die Bedingung(en) nicht erfüllen.

▷ **FailColEOut** (output_control) ..... line.end.x-array ⇝ *integer*
Spaltenkoordinaten der Endpunkte der Linien, die die Bedingung(en) nicht erfüllen.

Ergebnis

`partition_lines` liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`sobel_amp`, `edges_image`, `threshold`, `hysteresis_threshold`, `split_skeleton_region`, `split_skeleton_lines`

Nachfolger

`set_line_width`, `disp_line`

Alternativen

`line_orientation`, `line_position`, `select_lines`, `select_lines_longest`

Siehe auch

`select_lines`, `select_lines_longest`, `detect_edge_segments`, `select_shape`

Modul

Foundation

read_kalman (: : FileName : Dimension, Model, Measurement, Prediction)

Einlesen der Beschreibungsdatei eines Kalman-Filters.

`read_kalman` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`read_kalman` liest die Beschreibungsdatei `FileName` eines Kalman-Filters. Der Kalman-Filter liefert eine Schätzung des aktuellen Zustands (oder auch eine Prädiktion des künftigen Zustandes) eines diskreten, stochastisch gestörten, linearen Systems. Sie werden im Bereich der Bildverarbeitung insbesondere erfolgreich bei der Bildfolgenanalyse eingesetzt. Eine Kalman-Filterung stützt sich dabei auf ein mathematisches Modell des zu untersuchenden Systems, das durch folgende Größen charakterisiert wird:

Modellparameter: Übergangsmatrix A, Steuermatrix G mit der Stellgröße u und Messmatrix C

Modellstochastik: Systemfehler-Kovarianzmatrix Q, Systemfehler-Messfehler-Kovarianzmatrix L und Messfehler-Kovarianzmatrix R

Schätzung des Anfangszustandes des Systems: Zustand x_0 und zugehörige Kovarianzmatrix P_0

Viele Systeme kommen dabei ohne Eingaben „von außen“ und damit ohne G und u aus. Außerdem sind im Normalfall System- und Messfehler unkorreliert (L entfällt).

Die oben genannten Kenngrößen können in einer ASCII-Datei mit folgendem Aufbau abgelegt und dann mittels `read_kalman` eingelesen werden:

Dimensionszeile
+ Inhaltszeile
+ Matrix A
+ Matrix C

+ Matrix Q
[+ Matrix G + Vektor u]
[+ Matrix L]
+ Matrix R
[+ Matrix P_0]
[+ Vektor x_0]

Die Dimensionszeile hat dabei die Form

n = <integer> m = <integer> p = <integer>

wobei n die Anzahl der Zustandsvariablen, m die Anzahl der Messwerte und p die Anzahl der Stellglieder ist (siehe Parameter `Dimension`). Die maximale Dimension wird dabei durch eine Systemkonstante (derzeit 30) begrenzt.

Die Inhaltszeile ist von der Form

$$A * C * Q * G * u * L * R * P * x *$$

und beschreibt den weiteren Inhalt der Datei. Statt '$*$' ist dabei '+' (Parameter ist vorhanden) bzw. '-' (Parameter fehlt) einzusetzen. Zu beachten ist dabei, dass nur die oben durch [...] als optional gekennzeichneten Parameter in einer Beschreibungsdatei weggelassen werden dürfen. Fehlt die Anfangsschätzung a_0 (d.h. 'x-'), werden die Komponenten des Vektors als 0.0 angenommen. Fehlt die Kovarianzmatrix P_0 der Anfangsschätzung (d.h. 'P-'), wird angenommen, der Fehler sei sehr groß. Die Matrixelemente werden in diesem Fall auf 10000.0 gesetzt. Dieser Wert erscheint zwar hoch, ist aber nur dann ausreichend, wenn die Wertebereiche der Komponenten des Zustandsvektors x um Zehnerpotenzen kleiner sind. $(r \times s)$ Matrizen werden in der Form

$$\begin{array}{cccc} < Kommentar, d.h.\ string > & & & \\ < a_{11} > & < a_{12} > & \cdots & < a_{1s} > \\ \vdots & & \ddots & \vdots \\ < a_{r1} > & < a_{r2} > & \cdots & < a_{rs} > \end{array}$$

zeilenweise (mit beliebigen Zwischenräumen/Zeilenvorschüben) abgespeichert, Vektoren entsprechend in der Form

$$\begin{array}{ccc} < Kommentar, d.h.\ string > & & \\ < a_1 > & \cdots & < a_k > \end{array}$$

Zurückgeliefert werden von `read_kalman` folgende Parameterwerte:

`Dimension`: Dieser Parameter enthält die Dimensionen von Status-, Mess- und Stellvektor. Dimension ist somit ein Vektor *[n,m,p]*, wobei *'n'* die Anzahl der Zustandsvariablen, *'m'* die Anzahl der Messwerte und *'p'* die Anzahl der Stellglieder ist. Für ein System ohne deterministische Steuerung (d.h. ohne Einfluss „von außen“) ist folglich `Dimension` = *[n,m,0]*.

`Model`: Dieser Parameter enthält hintereinandergehängt die zeilenweise linearisierten Matrizen (Vektoren) *'A, C, Q, G, u'* und (gegebenenfalls) *'L'*. `Model` ist also ein Vektor der Länge $n \times n + n \times m + n \times n + n \times p + p[+n \times m]$. Der letzte Summand entfällt, falls System- und Messfehler unkorreliert sind, d.h. kein *'L'* angegeben wurde.

`Measurement`: Dieser Parameter enthält die zeilenweise linearisierte Matrix *'R'* ist also ein Vektor der Dimension $m \times m$.

`Prediction`: Dieser Parameter enthält hintereinandergehängt die zeilenweise linearisierte Matrix *'P_0'* (Fehler-Kovarianzmatrix der Anfangsschätzung) und die Anfangsschätzung *'x_0'*, ist also ein Vektor der Länge $n \times n + n$.

Parameter

▷ **`FileName`** (input_control) . . . . . filename.read ⇝ *string*
Beschreibungsdatei für einen Kalman-Filter.
Defaultwert: 'kalman.init'

▷ **`Dimension`** (output_control) . . . . . integer-array ⇝ *integer*
Die Dimensionen von Status-, Mess- und Stellvektor.

▷ **Model** (output_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierten Matrizen A, C, Q, eventuell G und u und gegebenenfalls L.

▷ **Measurement** (output_control) . . . . . real-array ⇝ *real*
Die zeilenweise linearisierte Matrix R.

▷ **Prediction** (output_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix P_0 (die Fehler-Kovarianzmatrix der Anfangsschätzung) und die Anfangsschätzung x_0

Beispiel

```
* An example of the description-file:
*
* n=3 m=1 p=0
* A+C+Q+G-u-L-R+P+x+
* transition matrix A:
* 1 1 0.5
* 0 1 1
* 0 0 1
* measurement matrix C:
* 1 0 0
* system-error covariance matrix Q:
* 54.3 37.9 48.0
* 37.9 34.3 42.5
* 48.0 42.5 43.7
* measurement-error covariance matrix R:
* 1.2
* estimation-error covariance matrix (for the initial estimate) P0: \
* 0 0 0
* 0 180.5 0
* 0 0 100
* initial estimate x0:
* 0 100 0
*
* the result of read_kalman with the upper descriptionfile
* as inputparameter:
*
* Dimension   = [3,1,0]
* Model       = [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,
*                      54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
* Measurement = [1.2]
* Prediction  = [0.0,0.0,0.0,0.0,180.5,0.0,0.0,0.0,100.0,0.0,100.0,
*                           0.0].
```

Ergebnis

Ist die Beschreibungsdatei lesbar und korrekt, liefert `read_kalman` den Wert TRUE. Andernfalls wird eventuell eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`filter_kalman`

Siehe auch

`update_kalman`, `filter_kalman`

Modul

Foundation

select_lines (: : RowBeginIn, ColBeginIn, RowEndIn, ColEndIn, Feature, Operation, Min, Max : RowBeginOut, ColBeginOut, RowEndOut, ColEndOut)

Auswahl von Linien gemäß verschiedener Kriterien.

select_lines ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

select_lines wählt Linien anhand verschiedener Kriterien aus. Für jede Eingabelinie werden die angegebenen Merkmale (Feature) berechnet. Wenn jedes (Operation = *'and'*) oder mindestens eines (Operation = *'or'*) der so berechneten Merkmale in den vorgegebenen Grenzen (Min,Max) liegt, wird die Linie in die Ausgabe übernommen. Mit den Defaultwerten *'min'* beziehungsweise *'max'* der Parameter Min und Max werden diese Grenzen nach unten beziehungsweise oben offen gelassen.

Bedingung:

$$Min_i \leq Feature_i(Linie) \leq Max_i$$

Mögliche Werte für Feature:

'length' (euklidische) Länge der Linie

'row' Zeilenindex des Schwerpunkts

'column' Spaltenindex des Schwerpunkts

'phi' Orientierung der Linie ($-\frac{\pi}{2} < \varphi \leq \frac{\pi}{2}$)

Achtung

Wird nur ein Merkmal verwendet, dann ist der Wert von Operation bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie übergeben werden.

Parameter

- ▷ **RowBeginIn** (input_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **ColBeginIn** (input_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **RowEndIn** (input_control) ..... line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **ColEndIn** (input_control) ..... line.end.x-array ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **Feature** (input_control) ..... string(-array) ⇝ *string*
  Gewünschte Merkmale.
  Defaultwert: 'length'
  Werteliste: Feature ∈ {'length', 'row', 'column', 'phi'}
- ▷ **Operation** (input_control) ..... string ⇝ *string*
  Gewünschte Verknüpfung.
  Defaultwert: 'and'
  Werteliste: Operation ∈ {'and', 'or'}
- ▷ **Min** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Untere Grenzen der Merkmale oder 'min'.
  Defaultwert: 'min'
- ▷ **Max** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Obere Grenzen der Merkmale oder 'max'.
  Defaultwert: 'max'
- ▷ **RowBeginOut** (output_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Ausgabelinien.
- ▷ **ColBeginOut** (output_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Ausgabelinien.
- ▷ **RowEndOut** (output_control) ..... line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Ausgabelinien.

▷ **ColEndOut** (output_control) ..... line.end.x-array ⇝ *integer*
Spaltenindizes der Endpunkte der Ausgabelinien.

Ergebnis

`select_lines` liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`sobel_amp`, `edges_image`, `threshold`, `hysteresis_threshold`, `split_skeleton_region`, `split_skeleton_lines`

Nachfolger

`set_line_width`, `disp_line`

Alternativen

`line_orientation`, `line_position`, `partition_lines`

Siehe auch

`partition_lines`, `select_lines_longest`, `detect_edge_segments`, `select_shape`

Modul

Foundation

select_lines_longest (: : RowBeginIn, ColBeginIn, RowEndIn, ColEndIn, Num : RowBeginOut, ColBeginOut, RowEndOut, ColEndOut)

Auswahl der längsten Eingabelinien.

`select_lines_longest` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`select_lines_longest` wählt aus den durch die Tupel `RowBeginIn`, `ColBeginIn`, `RowEndIn` und `ColEndIn` beschriebenen Eingabelinien die `Num` längsten Eingabelinien aus.

Parameter

▷ **RowBeginIn** (input_control) ..... line.begin.y-array ⇝ *integer*
Zeilenkoordinaten der Anfangspunkte der Eingabelinien.

▷ **ColBeginIn** (input_control) ..... line.begin.x-array ⇝ *integer*
Spaltenkoordinaten der Anfangspunkte der Eingabelinien.

▷ **RowEndIn** (input_control) ..... line.end.y-array ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Eingabelinien.

▷ **ColEndIn** (input_control) ..... line.end.x-array ⇝ *integer*
Spaltenkoordinaten der Endpunkte der Eingabelinien.

▷ **Num** (input_control) ..... integer ⇝ *integer*
(Maximal) gewünschte Anzahl von Ausgabelinien.
Defaultwert: 10

▷ **RowBeginOut** (output_control) ..... line.begin.y-array ⇝ *integer*
Zeilenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **ColBeginOut** (output_control) ..... line.begin.x-array ⇝ *integer*
Spaltenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **RowEndOut** (output_control) ..... line.end.y-array ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Ausgabelinien.

▷ **ColEndOut** (output_control) ..... line.end.x-array ⇝ *integer*
Spaltenindizes der Endpunkte der Ausgabelinien.

Ergebnis

`select_lines_longest` liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`sobel_amp`, `edges_image`, `threshold`, `hysteresis_threshold`, `split_skeleton_region`, `split_skeleton_lines`

Nachfolger

`set_line_width`, `disp_line`

Alternativen

`line_orientation`, `line_position`, `select_lines`, `partition_lines`

Siehe auch

`select_lines`, `partition_lines`, `detect_edge_segments`, `select_shape`

Modul

Foundation

update_kalman (: : FileName, DimensionIn, ModelIn, MeasurementIn : DimensionOut, ModelOut, MeasurementOut)

Einlesen einer Updatedatei eines Kalman-Filters.

`update_kalman` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.

`update_kalman` liest die Updatedatei `FileName` eines Kalman-Filters. Kalman-Filter liefern eine Schätzung des aktuellen Zustands (oder auch eine Prädiktion des künftigen Zustandes) eines diskreten, stochastisch gestörten, linearen Systems.

Eine Kalman-Filterung stützt sich dabei auf ein mathematisches Modell des zu untersuchenden Systems, das zu jedem Zeitpunkt durch folgende Größen charakterisiert wird:

Modellparameter: Übergangsmatrix A, Steuermatrix G mit der Stellgröße u und Messmatrix C

Modellstochastik: Systemfehler-Kovarianzmatrix Q, Systemfehler-Messfehler-Kovarianzmatrix L und Messfehler-Kovarianzmatrix R

Messvektor: y

Systemvergangenheit: Extrapolationsvektor $\hat{x}$ und Extrapolationsfehler-Kovarianzmatrix $\hat{P}$

Viele Systeme kommen dabei ohne Eingaben „von außen“ und damit ohne G und u aus. Außerdem sind im Normalfall System- und Messfehler unkorreliert (L entfällt). Einige der oben genannten Größen können sich dynamisch (von Iteration zu Iteration) ändern. `update_kalman` dient zur Modifikation von Systemteilen gemäß einer Updatedatei (ASCII) mit folgendem Aufbau (vgl. auch `read_kalman`):

Dimensionszeile
+ Inhaltszeile
+ Matrix A
+ Matrix C
+ Matrix Q
+ Matrix G + Vektor u
+ Matrix L
+ Matrix R

Die Dimensionszeile hat dabei die Form

n = <integer> m = <integer> p = <integer>

wobei n die Anzahl der Zustandsvariablen, m die Anzahl der Messwerte und p die Anzahl der Stellglieder ist (siehe Parameter `DimensionIn` / `DimensionOut`). Die maximale Dimension wird dabei durch eine Systemkonstante (derzeit 30) begrenzt. Da hier an einem gültigen Modell Veränderungen vorgenommen werden sollen, sind die Dimensionen n und m unveränderbar (und werden nur zu Kontrollzwecken angegeben). Die Inhaltszeile ist von der Form

$$A * C * Q * G * u * L * R*$$

und beschreibt den weiteren Inhalt der Datei. Statt '$*$' ist dabei '+' (Parameter ist vorhanden) bzw. '-' (Parameter fehlt) einzusetzen. Im Gegensatz zu Beschreibungsdateien für `read_kalman` muss die Systembeschreibung hier nicht vollständig sein. Anzugeben sind nur die sich ändernden Systemteile. Die Angabe von Schätzwerten entfällt, da diese gemäß Konstruktion des Filters von der jeweils letzten Filterung stammen müssen. $(r \times s)$ Matrizen werden in der Form

$$\begin{array}{cccc} < Kommentar, d.h.\ string > & & & \\ < a_{11} > & < a_{12} > & \cdots & < a_{1s} > \\ \vdots & & \ddots & \vdots \\ < a_{r1} > & < a_{r2} > & \cdots & < a_{rs} > \end{array}$$

zeilenweise (mit beliebigen Zwischenräumen/Zeilenvorschüben) abgespeichert, Vektoren entsprechend in der Form

$$\begin{array}{ccc} < Kommentar, d.h.\ string > & & \\ < a_1 > & \cdots & < a_k > \end{array}$$

Verändert werden von `read_kalman` folgende Parameterwerte:

`DimensionIn` / `DimensionOut`: Diese Parameter enthalten die Dimensionen von Status-, Mess- und Stellvektor und sind daher Vektoren *[n,m,p]*, wobei *'n'* die Anzahl der Zustandsvariablen, *'m'* die Anzahl der Messwerte und *'p'* die Anzahl der Stellglieder ist. *'n'* und *'m'* sind für ein gegebenes System fest, dürfen also in der Updatedatei nicht von den entsprechenden Eingabewerten abweichen. Für ein System ohne deterministische Steuerung (d.h. ohne Einfluss „von außen“) ist *'p' = 0*.

`ModelIn` / `ModelOut`: Diese Parameter enthalten hintereinandergehängt die zeilenweise linearisierten Matrizen (Vektoren) *'A, C, Q, G, u'* und (gegebenenfalls) *'L'*. `ModelIn` / `ModelOut` sind also Vektoren der Länge $n \times n + n \times m + n \times n + n \times p + p[+n \times m]$ Der letzte Summand entfällt, falls System- und Messfehler unkorreliert sind, d.h. kein *'L'* angegeben wurde.

`MeasurementIn` / `MeasurementOut`: Diese Parameter enthalten die zeilenweise linearisierte Matrix *'R'*, sind also Vektoren der Dimension $m \times m$.

Parameter

▷ **`FileName`** (input_control) . . . . . filename.read ⇝ *string*
Updatedatei für einen Kalman-Filter.
Defaultwert: 'kalman.updt'

▷ **`DimensionIn`** (input_control) . . . . . integer-array ⇝ *integer*
Die Dimensionen von Status-, Mess- und Stellvektor.
Defaultwert: [3,1,0]
Typischer Wertebereich: $0 \leq$ `DimensionIn` ≤ 30

▷ **`ModelIn`** (input_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierten Matrizen A,C,Q, eventuell G und u und gegebenenfalls L.
Defaultwert: [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
Typischer Wertebereich: $0.0 \leq$ `ModelIn` ≤ 10000.0

▷ **`MeasurementIn`** (input_control) . . . . . real-array ⇝ *real*
Die zeilenweise linearisierte Matrix R.
Defaultwert: [1,2]
Typischer Wertebereich: $0.0 \leq$ `MeasurementIn` ≤ 10000.0

▷ **`DimensionOut`** (output_control) . . . . . integer-array ⇝ *integer*
Die Dimensionen von Status-, Mess- und Stellvektor.

▷ **ModelOut** (output_control) . . . . . real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierten Matrizen A,C,Q, eventuell G und u und gegebenenfalls L.

▷ **MeasurementOut** (output_control) . . . . . real-array ⇝ *real*
Die zeilenweise linearisierte Matrix R.

Beispiel

```
* The following values are describing the system
*
* DimensionIn    = [3,1,0]
* ModelIn        = [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,
*                       54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
* MeasurementIn = [1,2]
*
* An example of the Updatefile:
*
* n=3 m=1 p=0
* A+C-Q-G-u-L-R-
* transitions at time t=15:
* 2 1 1
* 0 2 2
* 0 0 2
*
* the results of update_kalman:
*
* DimensionOut    = [3,1,0]
* ModelOut        = [2.0,1.0,1.0,0.0,2.0,2.0,0.0,0.0,2.0,1.0,0.0,0.0,
*                         54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
* MeasurementOut = [1.2]
```

Ergebnis

Ist die Updatedatei lesbar und korrekt, liefert `update_kalman` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`filter_kalman`

Siehe auch

`read_kalman`, `filter_kalman`

Modul

Foundation

16.14 XLD

```
union_straight_contours_histo_xld ( Contours : UnionContours,
    SelectedContours : RefLineStartRow,  RefLineStartColumn,
    RefLineEndRow,  RefLineEndColumn,  Width,  MaxWidth,
    FilterSize : HistoValues )
```

Vereinigen von benachbarten geraden Konturen anhand des Abstandes von einer gegebenen Linie.

union_straight_contours_histo_xld ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt. Neue Applikationen sollten stattdessen die Operatoren aus dem Kapitel 1D-Vermessung verwenden.

Mit union_straight_contours_histo_xld werden benachbarte Konturen Contours verglichen und unter gewissen Voraussetzungen vereinigt.

Es wird die minimale und maximale Distanz der Kontur zu einer gegebenen Referenzlinie berechnet. Mit den Distanzen wird ein Histogramm erzeugt. Soll das Histogramm geglättet werden, so muss FilterSize > 1 sein. Das resultierende Histogramm wird anschließend in Bereiche unterteilt (von Minima zu Minima). Zum Abschluss werden die Konturen, die in einem Bereich liegen, zu einer neuen Kontur zusammengefasst. Ist die Breite des Bereichs größer als MaxWidth, so werden alle Konturen des Bereichs ignoriert (verworfen). Liegt eine Kontur in mehreren Bereichen, so wird diese ebenfalls ignoriert. Bei parallel verlaufenden Konturen besteht die Gefahr, dass nebeneinander liegende Konturen vereinigt werden.

Für jede durch Vereinigung neu entstandene Kontur werden die Regressionsgeradenparameter neu berechnet.

Die resultierenden Konturen können nicht dargestellt werden.

Achtung

Bevor Konturen mit union_straight_contours_histo_xld vereinigt werden können, müssen die Regressionsgeradenparameter mit regress_contours_xld berechnet werden. Es ist außerdem zu beachten, dass bereits geschlossene Konturen nicht mehr in eine Vereinigung einbezogen werden.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *object*
  Eingabe-Konturen.
- ▷ **UnionContours** (output_object) ..... xld_cont-array ⇝ *object*
  Ausgabe-Konturen.
- ▷ **SelectedContours** (output_object) ..... xld_cont-array ⇝ *object*
  Ausgabe-Konturen.
- ▷ **RefLineStartRow** (input_control) ..... line.begin.y ⇝ *integer*
  y-Koordinate des Startpunktes der Referenzlinie.
  Defaultwert: 0
- ▷ **RefLineStartColumn** (input_control) ..... line.begin.x ⇝ *integer*
  x-Koordinate des Startpunktes der Referenzlinie.
  Defaultwert: 0
- ▷ **RefLineEndRow** (input_control) ..... line.end.y ⇝ *integer*
  y-Koordinate des Endpunktes der Referenzlinie.
  Defaultwert: 0
- ▷ **RefLineEndColumn** (input_control) ..... line.end.x ⇝ *integer*
  x-Koordinate des Endpunktes der Referenzlinie.
  Defaultwert: 0
- ▷ **Width** (input_control) ..... integer ⇝ *integer*
  Maximale Distanz.
  Defaultwert: 1
- ▷ **MaxWidth** (input_control) ..... integer ⇝ *integer*
  Maximale Breite zwischen zwei Minima.
  Defaultwert: 1
- ▷ **FilterSize** (input_control) ..... integer ⇝ *integer*
  Größe des Glättungsfilters.
  Defaultwert: 1
  Typischer Wertebereich: $1 \leq$ FilterSize ≤ 63
- ▷ **HistoValues** (output_control) ..... integer-array ⇝ *integer*
  Ausgabe der Werte des Histogramms.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

Kapitel 17

Matching

17.1 Deformierbar

clear_deformable_model (: : ModelID :)

Freigabe des Speichers eines deformierbaren Formmodells.

clear_deformable_model gibt den Speicher eines deformierbaren Formmodells, das mit create_planar_uncalib_deformable_model oder create_planar_calib_deformable_model angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle ModelID ist nach dem Aufruf ungültig.

Parameter

▷ **ModelID** (input_control) . . . . . deformable_model(-array) ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert clear_deformable_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

create_planar_uncalib_deformable_model, create_planar_calib_deformable_model, create_local_deformable_model

Modul

Matching

create_local_deformable_model (Template : : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleRMin, ScaleRMax, ScaleRStep, ScaleCMin, ScaleCMax, ScaleCStep, Optimization, Metric, Contrast, MinContrast, GenParamName, GenParamValue : ModelID)

Erzeugt ein deformierbares Modell für lokal deformierbares Matching.

Der Operator `create_local_deformable_model` bereitet ein Muster, das als Bild `Template` übergeben wird, als deformierbares Modell für lokal deformierbares Matching vor. Die ROI des Modells wird als der Definitionsbereich von `Template` übergeben.

Das lokal deformierbare Matching kann dazu verwendet werden, ein Objekt zu erkennen, das durch eine lokale Verformung verzeichnet ist. Das ist ähnlich wie bei `create_planar_uncalib_deformable_model` oder `create_planar_calib_deformable_model`, die aber nur perspektivische Verformungen kompensieren, ansonsten aber ein rigides Objekt erwarten. Weiterhin ist es eine Alternative zu dem Modus 'max_deformation' in `find_shape_model`. Hierbei bleibt das Modell zwar rigide, ist aber in der Lage kleinere Deformationen zu kompensieren. Im Gegensatz zu `find_shape_model` kann `create_local_deformable_model` dazu verwendet werden, die Verformung zu schätzen und den Teil des Bildes zu rektifizieren in dem das Bild gefunden wurde. Weiter ist zu beachten, dass `create_local_deformable_model` größere Deformationen tolerieren kann.

Wie in `create_planar_uncalib_deformable_model` beschrieben, liegt der Ursprung des Modells (Referenzpunkt) im Schwerpunkt der Region des Modellbildes `Template`. Zusätzlich wird das achsenparallele umschließende Rechteck der Region des `Template` verwendet, um den Bereich des Suchbildes, das in Aufrufen von `find_local_deformable_model` rektifiziert wird, zu bestimmen.

Für eine weitere Erklärung des deformierbaren Modells wie auch seiner Parameter verweisen wir auf die Beschreibung in `create_planar_uncalib_deformable_model`. Es ist zu beachten, dass die Parameter der tatsächlichen Verformung in Aufrufen von `find_local_deformable_model` gesetzt werden.

Parameter

▷ **`Template`** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.

▷ **`NumLevels`** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: `NumLevels` ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **`AngleStart`** (input_control) . . . . . angle.rad-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **`AngleExtent`** (input_control) . . . . . angle.rad-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **`AngleStep`** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: `AngleStep` ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep > 0 && AngleStep <= pi / 16

▷ **`ScaleRMin`** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleRMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **`ScaleRMax`** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **`ScaleRStep`** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.
Defaultwert: 'auto'
Wertevorschläge: `ScaleRStep` ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleRStep >= 0

▷ **`ScaleCMin`** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleCMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleCStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.
Defaultwert: 'auto'
Wertevorschläge: `ScaleCStep` ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleCStep >= 0

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung der Modellgenerierung.
Defaultwert: 'none'
Werteliste: `Optimization` ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_part_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **Contrast** (input_control) . . . . . number-array ⇝ *integer* / string
Schwellenwerte bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Musterbild.
Defaultwert: 'auto'
Wertevorschläge: `Contrast` ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer* / string
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: `MinContrast` ∈ {'auto', 1, 2, 3, 5, 7, 10, 20, 30, 40}
Restriktion: MinContrast < Contrast

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {[], 'min_size', 'part_size'}

▷ **GenParamValue** (input_control) . . . . . integer-array ⇝ *integer* / real / string
Werte der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamValue` ∈ {[], 'small', 'medium', 'big'}

▷ **ModelID** (output_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_local_deformable_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`determine_deformable_model_params`

Nachfolger

`set_deformable_model_origin`, `set_deformable_model_param`, `get_deformable_model_contours`, `find_local_deformable_model`, `get_deformable_model_params`, `write_deformable_model`, `clear_deformable_model`

Alternativen

read_deformable_model

Siehe auch

create_planar_uncalib_deformable_model, create_planar_calib_deformable_model

Modul

Matching

create_local_deformable_model_xld (Contours : : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleRMin, ScaleRMax, ScaleRStep, ScaleCMin, ScaleCMax, ScaleCStep, Optimization, Metric, MinContrast, GenParamName, GenParamValue : ModelID)

Vorbereiten eines deformierbaren Modells für lokal deformierbares Matching aus XLD-Konturen.

Der Operator create_local_deformable_model_xld erzeugt ein deformierbares Modell für lokal deformierbares Matching aus XLD Konturen Contours. Die XLD Konturen repräsentieren dabei die Grauwertkanten des zu suchenden Objekts im Bild. Im Gegensatz zu dem Operator create_local_deformable_model, der ein Modell aus einem Bild erzeugt, erzeugt der Operator create_local_deformable_model_xld das Formmodell aus XLD Konturen.

Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt des kleinsten umschließenden Rechtecks der Konturen Contours, das parallel zu den Koordinatenachsen liegt, verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit set_deformable_model_origin gesetzt werden.

Für eine weitere Erklärung des deformierbaren Modells wie auch seiner Parameter wird auf die Beschreibung in create_planar_uncalib_deformable_model und create_local_deformable_model verwiesen. Es ist zu beachten, das mit Hilfe von set_local_deformable_model_metric die Metrik des Modells verändert werden kann.

Achtung

Im Gegensatz zum Operator create_local_deformable_model ist es nicht möglich die Mindestgröße der Modellkomponenten anzugeben. Um kleine Modellkomponenten zu vermeiden, können kurze Konturen vor dem Aufruf von create_local_deformable_model_xld mit dem Operator select_contours_xld eliminiert werden.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabekonturen, die zum Aufbau des Modells verwendet werden.
- ▷ **NumLevels** (input_control) ..... integer ⇝ *integer* / string
  Maximale Anzahl von Pyramidenebenen.
  Defaultwert: 'auto'
  Werteliste: NumLevels ∈ {'auto', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **AngleStart** (input_control) ..... angle.rad-array ⇝ *real*
  Dieser Parameter wird nicht benutzt.
  Defaultwert: []
- ▷ **AngleExtent** (input_control) ..... angle.rad-array ⇝ *real*
  Dieser Parameter wird nicht benutzt.
  Defaultwert: []
- ▷ **AngleStep** (input_control) ..... angle.rad ⇝ *real* / string
  Schrittweite der Winkel (Auflösung).
  Defaultwert: 'auto'
  Wertevorschläge: AngleStep ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
  Restriktion: AngleStep > 0 && AngleStep <= pi / 16
- ▷ **ScaleRMin** (input_control) ..... number ⇝ *real*
  Kleinste auftretende Skalierung des Musters in Zeilenrichtung.
  Defaultwert: 1.0
  Wertevorschläge: ScaleRMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: ScaleRMin > 0

▷ **ScaleRMax** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleRStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.
Defaultwert: 'auto'
Wertevorschläge: `ScaleRStep` ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleRStep > 0

▷ **ScaleCMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleCMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleCStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.
Defaultwert: 'auto'
Wertevorschläge: `ScaleCStep` ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleCStep > 0

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: `Optimization` ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'ignore_local_polarity'
Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_part_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer*
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 5
Wertevorschläge: `MinContrast` ∈ {1, 2, 3, 5, 7, 10, 20, 30, 40}

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {[], 'part_size'}

▷ **GenParamValue** (input_control) . . . . . integer-array ⇝ *integer* / real / string
Werte der Parameter.
Defaultwert: []
Werteliste: `GenParamValue` ∈ {[], 'small', 'medium', 'big'}

▷ **ModelID** (output_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_local_deformable_model_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter `NumLevels` so gewählt wurde, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`read_contour_xld_dxf`, `edges_sub_pix`, `select_contours_xld`

Nachfolger

`find_local_deformable_model`

Siehe auch

`create_local_deformable_model`

Modul

Matching

create_planar_calib_deformable_model (Template : : CamParam, ReferencePose, NumLevels, AngleStart, AngleExtent, AngleStep, ScaleRMin, ScaleRMax, ScaleRStep, ScaleCMin, ScaleCMax, ScaleCStep, Optimization, Metric, Contrast, MinContrast, GenParamName, GenParamValue : ModelID)

Erzeugen eines deformierbaren Modells für das kalibrierte, perspektivische Matching.

Der Operator `create_planar_calib_deformable_model` bereitet ein Muster, das als Bild `Template` übergeben wird, als deformierbares Modell für planares, kalibriertes Matching vor. Die ROI des Modells wird als der Definitionsbereich von `Template` übergeben.

Das planare, kalibrierte Matching kann dazu verwendet werden, die 3D Lage eines planaren Objektes oder eines planaren Teiles eines nichtplanaren Objektes, das durch eine perspektivische Ansicht verzerrt wurde, zu finden.

Dies ist eine Alternative zu `create_planar_uncalib_deformable_model`, das die Lage als 2D Homographie statt der 3D Lage zurückliefert.

Damit man die 3D Lage bestimmen kann, benötigt man die internen Parameter `CamParam` der Kamera. Typischerweise können die internen Parameter mit Hilfe von `calibrate_cameras` ermittelt werden.

Zusätzlich zu den internen Kameraparametern muss die metrische Größe des Bildausschnitts, welcher in `Template` übergeben wird, bekannt sein.

Hierfür muss eine 3D Referenzlage, d.h. die 3D Ebene in der das Modell liegt, mit `ReferencePose` angegeben werden. Diese 3D Modellebene wird durch die x und y Achse des Koordinatensystems, das durch `ReferencePose` angegeben wird, aufgespannt.

Wie in `create_planar_uncalib_deformable_model` beschrieben, liegt der Ursprung des Modells (Referenzpunkt) im Schwerpunkt der Region des Modellbildes `Template`. Dieser Bildpunkt definiert einen Sichtstrahl der mit der Modellebene, die in `ReferencePose` übergeben wurde, geschnitten wird, um die absolute 3D Lage des Modells zu erhalten. Das ist auch die absolute Lage, die später in `find_planar_calib_deformable_model` zurückgegeben wird. Deshalb kann sich die 3D Lage des Modells von der Referenzpose um eine 2D Translation entlang der Modellebene unterscheiden. Die finale Pose des Modells kann durch Aufruf von `get_deformable_model_params` bestimmt werden. Die 3D Modellebene kann durch `calibrate_cameras` oder `vector_to_pose` ermittelt werden. Um die Koordinatensysteme konsistent zu halten, sollte `ReferencePose` so ausgerichtet sein, als ob eine Kalibrierplatte auf das Objekt gelegt wurde.

Für eine weitere Erklärung des planaren, deformierbaren Modells wie auch seiner Parameter verweisen wir auf die Beschreibung in `create_planar_uncalib_deformable_model`.

Parameter

- ▷ **Template** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.
- ▷ **CamParam** (input_control) . . . . . campar ⇝ *real* / integer / string
  Die internen Kameraparameter der Kamera.
- ▷ **ReferencePose** (input_control) . . . . . pose ⇝ *real* / integer
  Die Pose des Objekts im Referenzbild.

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: NumLevels ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStart** (input_control) . . . . . angle.rad-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **AngleExtent** (input_control) . . . . . angle.rad-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: AngleStep ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep > 0 && AngleStep <= pi / 16

▷ **ScaleRMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: ScaleRMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **ScaleRMax** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleRStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleRStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleRStep >= 0

▷ **ScaleCMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: ScaleCMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleCStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleCStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleCStep >= 0

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung der Modellgenerierung.
Defaultwert: 'none'
Werteliste: Optimization ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_part_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **Contrast** (input_control) . . . . . number-array ⇝ *integer* / string
Schwellenwerte bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Musterbild.
Defaultwert: 'auto'
Wertevorschläge: Contrast ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

▷ **MinContrast** (input_control) ..... number ⇝ *integer* / string
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: MinContrast ∈ {'auto', 1, 2, 3, 5, 7, 10, 20, 30, 40}
Restriktion: MinContrast < Contrast

▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
Namen der Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {[], 'part_size'}

▷ **GenParamValue** (input_control) ..... integer-array ⇝ *integer* / real / string
Werte der Parameter.
Defaultwert: []
Werteliste: GenParamValue ∈ {[], 'small', 'medium', 'big'}

▷ **ModelID** (output_control) ..... deformable_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_planar_calib_deformable_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

determine_deformable_model_params

Nachfolger

set_deformable_model_origin, set_deformable_model_param, get_deformable_model_contours, find_planar_calib_deformable_model, get_deformable_model_params, write_deformable_model, clear_deformable_model

Alternativen

read_deformable_model

Siehe auch

create_planar_uncalib_deformable_model

Modul

Matching

create_planar_calib_deformable_model_xld (Contours : : CamParam, ReferencePose, NumLevels, AngleStart, AngleExtent, AngleStep, ScaleRMin, ScaleRMax, ScaleRStep, ScaleCMin, ScaleCMax, ScaleCStep, Optimization, Metric, MinContrast, GenParamName, GenParamValue : ModelID)

Vorbereiten eines deformierbaren Modells für planares kalibriertes Matching aus XLD-Konturen.

create_planar_calib_deformable_model_xld erzeugt ein deformierbares Modell für planares, kalibriertes Matching aus den XLD Konturen Contours. Die XLD Konturen repräsentieren dabei die Grauwertkanten des zu suchenden Objekts im Bild. Im Gegensatz zu dem Operator create_planar_calib_deformable_model, der ein Modell aus einem Bild erzeugt, erzeugt der Operator create_planar_calib_deformable_model_xld das Formmodell aus XLD Konturen. Es ist zu

beachten, dass der Operator die Konturen `Contours` in metrischen Weltkoordinaten erwartet. Das ist im Gegensatz zu `create_planar_uncalib_deformable_model_xld`, bei der die Konturen `Contours` in Bildkoordinaten übergeben werden.

Weiterhin ist zu beachten, dass der Ursprung (Referenzpunkt) des Modells direkt von den übergebenen Konturen `Contours` verwendet wird. Das ist im Gegensatz zu `create_planar_uncalib_deformable_model_xld`, bei der der Schwerpunkt des kleinsten umschließenden Rechtecks der Konturen `Contours`, das parallel zu den Koordinatenachsen liegt, verwendet wird. Falls ein anderer Ursprung gewünscht wird, kann dieser mit `set_deformable_model_origin` im Weltkoordinatensystem gesetzt werden.

Für eine weitere Erklärung des planaren, deformierbaren Modells wie auch seiner Parameter wird auf die Beschreibung in `create_planar_uncalib_deformable_model` und `create_planar_calib_deformable_model` verwiesen.

Achtung

Im Gegensatz zum Operator `create_planar_calib_deformable_model` ist es nicht möglich die Mindestgröße der Modellkomponenten anzugeben. Um kleine Modellkomponenten zu vermeiden, können kurze Konturen vor dem Aufruf von `create_planar_calib_deformable_model_xld` mit dem Operator `select_contours_xld` eliminiert werden.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabekonturen, die zum Aufbau des Modells verwendet werden.
- ▷ **`CamParam`** (input_control) ..... campar ⇝ *real* / integer / string
  Die internen Kameraparameter der Kamera.
- ▷ **`ReferencePose`** (input_control) ..... pose ⇝ *real* / integer
  Die Referenzpose des Objekts.
- ▷ **`NumLevels`** (input_control) ..... integer ⇝ *integer* / string
  Maximale Anzahl von Pyramidenebenen.
  Defaultwert: 'auto'
  Werteliste: `NumLevels` ∈ {'auto', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **`AngleStart`** (input_control) ..... angle.rad-array ⇝ *real*
  Dieser Parameter wird nicht benutzt.
  Defaultwert: []
- ▷ **`AngleExtent`** (input_control) ..... angle.rad-array ⇝ *real*
  Dieser Parameter wird nicht benutzt.
  Defaultwert: []
- ▷ **`AngleStep`** (input_control) ..... angle.rad ⇝ *real* / string
  Schrittweite der Winkel (Auflösung).
  Defaultwert: 'auto'
  Wertevorschläge: `AngleStep` ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
  Restriktion: AngleStep > 0 && AngleStep <= pi / 16
- ▷ **`ScaleRMin`** (input_control) ..... number ⇝ *real*
  Kleinste auftretende Skalierung des Musters in Zeilenrichtung.
  Defaultwert: 1.0
  Wertevorschläge: `ScaleRMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: ScaleRMin > 0
- ▷ **`ScaleRMax`** (input_control) ..... number-array ⇝ *real*
  Dieser Parameter wird nicht benutzt.
  Defaultwert: []
- ▷ **`ScaleRStep`** (input_control) ..... number ⇝ *real* / string
  Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.
  Defaultwert: 'auto'
  Wertevorschläge: `ScaleRStep` ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
  Restriktion: ScaleRStep > 0
- ▷ **`ScaleCMin`** (input_control) ..... number ⇝ *real*
  Kleinste auftretende Skalierung des Musters in Spaltenrichtung.
  Defaultwert: 1.0
  Wertevorschläge: `ScaleCMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleCStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.
Defaultwert: 'auto'
Wertevorschläge: `ScaleCStep` ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleCStep > 0

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: `Optimization` ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'ignore_local_polarity'
Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_part_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer*
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 5
Wertevorschläge: `MinContrast` ∈ {1, 2, 3, 5, 7, 10, 20, 30, 40}

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {[], 'part_size'}

▷ **GenParamValue** (input_control) . . . . . integer-array ⇝ *integer* / real / string
Werte der Parameter.
Defaultwert: []
Werteliste: `GenParamValue` ∈ {[], 'small', 'medium', 'big'}

▷ **ModelID** (output_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_planar_calib_deformable_model_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter `NumLevels` so gewählt wurde, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`read_contour_xld_dxf`, `edges_sub_pix`, `select_contours_xld`

Nachfolger

`find_planar_calib_deformable_model`

Siehe auch

`create_planar_calib_deformable_model`

Modul

Matching

create_planar_uncalib_deformable_model (Template : : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleRMin, ScaleRMax, ScaleRStep, ScaleCMin, ScaleCMax, ScaleCStep, Optimization, Metric, Contrast, MinContrast, GenParamName, GenParamValue : ModelID)

Erzeugt ein deformierbares Modell für unkalibriertes, planares Matching.

Der Operator create_planar_uncalib_deformable_model bereitet ein Muster, das als Bild Template übergeben wird, als deformierbares Modell für das Matching vor. Die ROI des Modells wird als der Definitionsbereich von Template übergeben.

Das planare, unkalibrierte Matching kann zum Erkennen von planaren Objekten oder planaren Teilen von nicht-planaren Objekten, die durch eine projektive Ansicht verzerrt wurden, verwendet werden.

Dies ist eine Alternative zu create_shape_model, create_scaled_shape_model und create_aniso_shape_model, bei denen typischerweise das Suchbild mit Hilfe von calibrate_cameras und gen_image_to_world_plane_map vorher entzerrt werden muss.

Im Gegensatz zu create_shape_model_3d werden vorab keine Ansichten des Objektes generiert, was zu einem kleineren Speicherverbrauch führt. Deshalb sollte in dem Fall eines planaren, perspektivisch verzerrten Objektes create_planar_uncalib_deformable_model vorgezogen werden.

Das Modell wird auf mehreren Pyramidenebenen generiert und im Speicher abgelegt. Der Ausgabeparameter ModelID ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von find_planar_uncalib_deformable_model verwendet wird.

Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt der Region des Modellbildes Template verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit set_deformable_model_origin gesetzt werden.

Eingabeparameter im Detail

NumLevels: Die Anzahl der Pyramidenebenen wird mit NumLevels festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von NumLevels ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt. Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit 0 Pyramidenstufen entstehen würde, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert create_planar_uncalib_deformable_model eine Fehlermeldung zurück. Falls für NumLevels *'auto'* übergeben wird, wählt create_planar_uncalib_deformable_model die Anzahl der Pyramidenstufen automatisch. Die automatisch gewählte Anzahl von Pyramidenstufen kann mit get_deformable_model_params abgefragt werden. In seltenen Fällen kann es vorkommen, dass create_planar_uncalib_deformable_model die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für MinScore oder Greediness in find_planar_uncalib_deformable_model selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in find_planar_uncalib_deformable_model kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen mit Hilfe der Ausgabe von inspect_shape_model gewählt werden.

Winkel- und Skalierungs-Parameter: Beachten Sie, dass die AngleStart, AngleExtent, ScaleRMax und ScaleCMax nicht von diesem Operator genutzt werden. Stattdessen müssen sie in dem Operator find_planar_uncalib_deformable_model spezifiziert werden. ScaleRMin und ScaleCMin können relevant sein, wenn Matches erwartet werden, die kleiner sind als im Bild Template. Dann verringert sich eventuell die Anzahl der benutzten Pyramidenebenen, falls für NumLevels *'auto'* übergeben wird.

Im Normalfall können die Parameter AngleStep, ScaleRStep und ScaleCStep automatisch bestimmt werden - die berechneten Winkel- und Skalierungsschrittweiten können mit get_deformable_model_params ausgegeben werden. Mehr Informationen dazu, wie sich diese Parameter auswirken, können in der Dokumentation von find_planar_uncalib_deformable_model gefunden werden.

Optimization: Bei besonders großen Modellen kann es sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters Optimization auf einen Wert ungleich *'none'* zu setzen. Falls Optimization = *'none'*, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter Optimization reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei find_planar_uncalib_deformable_model notwendig werden, den Parameter Greediness auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen. Falls für Optimization *'auto'* übergeben wird, wählt create_planar_uncalib_deformable_model die Reduktion der Punkte automatisch.

Contrast: Der Parameter Contrast legt fest, welchen Grauwertkontrast die Punkte des Modells besitzen müssen. Der Kontrast ist ein Maß für die lokalen Grauwertdifferenzen zwischen dem Objekt und dem Hintergrund und zwischen verschiedenen Teilen des Objektes. Contrast sollte so gewählt werden, dass nur die signifikanten Merkmale des Musters für das Modell verwendet werden.

Folgende Werte können an Contrast übergeben werden:

- Ein einzelner Kontrastwert. Setzbar sind:
  - Integer-Wert: der manuell bestimmte Kontrastwert wird verwendet.
  - *'auto_contrast'*: der verwendete Kontrastwert wird automatisch ermittelt.
- Hysterese-Schwellenwerte. In diesem Fall wird bei der Segmentierung des Modells ein Verfahren eingesetzt, das ähnlich zum Hysterese-Schwellenwert-Verfahren in edges_image ist. Für weitere Informationen über das Hysterese-Schwellenwert-Verfahren siehe hysteresis_threshold. Setzbar sind:
  - Tupel bestehend aus zwei Werten: Der erste Wert bestimmt den unteren Schwellenwert und der zweite Wert den oberen Schwellenwert.
  - *'auto_contrast_hyst'* oder *'auto'* (Standardwert): Die beiden Hysterese-Schwellenwerte werden automatisch bestimmt.

In bestimmten Fällen kann es vorkommen, dass die automatische Bestimmung der Kontrast-Schwellenwerte nicht zum gewünschten Ergebnis führt. So ist eine manuelle Parameterwahl zu bevorzugen, wenn anwendungsbedingt bestimmte Modellkomponenten ausgeschlossen oder ins Modell integriert werden sollen oder wenn das Objekt mehrere unterschiedliche Kontraststufen umfasst. Daher ist es sinnvoll, die Kontrastwerte vor der Modellgenerierung mit determine_deformable_model_params automatisch zu ermitteln und das Ergebnis mit inspect_shape_model zu überprüfen. Es ist zu beachten, dass MinContrast die Bestimmung der Kontrast-Schwellenwerte beeinflusst, und daher auch die Schätzung der Mindestgröße.

MinContrast: Mit MinContrast wird festgelegt, welchen Grauwertkontrast im Bild das Modell später bei der Erkennung mit find_planar_uncalib_deformable_model mindestens besitzen muss. Mit anderen Worten stellt dieser Parameter eine Abgrenzung des Musters vom Rauschen im Bild dar. Ein guter Wert entspricht deshalb dem Bereich der Grauwertänderungen, die durch das Rauschen im Bild verursacht werden. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte MinContrast auf 10 gesetzt werden. Offensichtlich muss MinContrast kleiner als Contrast sein. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss MinContrast entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte MinContrast etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells mit find_planar_uncalib_deformable_model zu gewährleisten.

Wird MinContrast auf *'auto'* gesetzt, wählt create_planar_uncalib_deformable_model den Mindestkontrast basierend auf dem Rauschen im Modellbild automatisch. Eine automatische Bestimmung ergibt demzufolge nur Sinn, wenn das bei der Erkennung zu erwartende Bildrauschen dem Rauschen im Modellbild entspricht. Außerdem kann es in manchen Fällen sinnvoll sein, den automatisch ermittelten Wert zu erhöhen, um eine höhere Robustheit gegenüber Verdeckungen zu erzielen (siehe oben). Der automatisch bestimmte Mindestkontrast kann mit get_deformable_model_params abgefragt werden. Der MinContrast eines deformierbaren Modells kann später mit set_deformable_model_param noch geändert werden.

Metric: Der Parameter Metric legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird. Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist. Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_planar_uncalib_deformable_model erhöht sich in diesem Fall geringfügig.

Falls Metric = *'ignore_part_polarity'* ausgewählt wird, dann darf sich die Kontrastpolarität nur zwischen verschiedenen Modellteilen ändern, wohingegen die Polarität von Modellpunkten innerhalb desselben Modellteils nicht wechseln darf. Zu beachten ist, dass der Begriff *'ignore_part_polarity'* missverstanden werden kann. Er bedeutet, dass Polaritätswechsel zwischen benachbarten Modellteilen den Score nicht beeinflussen und daher ignoriert werden.

Falls Metric = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. In diesem Fall erhöht sich die Laufzeit von find_planar_uncalib_deformable_model aber wesentlich. Die obigen vier Metriken können nur auf einkanalige Bilder angewendet werden. Falls im Modellbild oder in den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes verwendet. Im Falle von Metric = *'ignore_color_polarity'* wird das Modell auch dann gefunden, wenn sich die Farbkontraste lokal ändern. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün. Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von find_planar_uncalib_deformable_model erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für Metric = *'ignore_color_polarity'* die Anzahl der Kanäle bei der Modellgenerierung mit create_planar_uncalib_deformable_model und bei der Suche mit find_planar_uncalib_deformable_model verschieden sein können. Dies erlaubt z.B. die Modellgenerierung aus einem einkanaligen synthetisch generierten Bild. Es ist auch zu beachten, dass die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein.

GenParamName, GenParamValue: Mit Hilfe der generischen Parameter GenParamName und GenParamValue kann der Benutzer Parameter eines deformierbaren Modells setzen, die nur in seltenen Fällen benötigt werden. Zur Zeit wird es dazu verwendet, die durchschnittliche Größe der Teile aus denen ein deformierbares Modell besteht zu setzen. Hierfür sollte GenParamName auf 'part_size' und GenParamValue auf 'small', 'medium' oder 'big' gesetzt werden. Für Objekte, die aus vielen kleinen Teilen bestehen, sollte 'small' ausgewählt werden. Für Objekte, die aus großen Konturen bestehen, sollte 'big' gesetzt werden. Als Default ist der Wert des Parameters 'big'.

Optional kann GenParamName = *'min_size'* gesetzt werden. Dieser Wert dient zur Selektion von signifikanten Komponenten des Modells basierend auf der Größe der Komponenten, d.h. Komponenten, deren Anzahl von Punkten kleiner ist als diese Mindestgröße, welche im entsprechenden GenParamValue gesetzt wird, werden unterdrückt. Dieser Schwellenwert wird pro Pyramidenstufe halbiert. *'min_size'* kann auf einen beliebigen Integerwert größer oder gleich 0 gesetzt werden. Außerdem kann er auf 'auto' gesetzt werden, womit die Mindestgröße automatisch bestimmt wird. Der Standardwert dieses Parameters ist 'auto'.

Der Schwerpunkt der Domäne (Region) des Modellbildes Template wird als Ursprung (Referenzpunkt) des Modells verwendet. Ein abweichender Ursprung für das Modell kann mit set_deformable_model_origin gesetzt werden.

Parameter

▷ **Template** (input_object) . . . . . (multichannel-)image ⇝ *object* : byte / uint2
Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: NumLevels ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStart** (input_control) . . . . . angle.rad-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **AngleExtent** (input_control) . . . . . angle.rad-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: AngleStep ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep > 0 && AngleStep <= pi / 16

▷ **ScaleRMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: ScaleRMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **ScaleRMax** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleRStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleRStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleRStep >= 0

▷ **ScaleCMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: ScaleCMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleCStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleCStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleCStep >= 0

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung der Modellgenerierung.
Defaultwert: 'none'
Werteliste: Optimization ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_part_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **Contrast** (input_control) . . . . . number-array ⇝ *integer* / string
Schwellenwerte bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Musterbild.
Defaultwert: 'auto'
Wertevorschläge: Contrast ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer* / string
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: MinContrast ∈ {'auto', 1, 2, 3, 5, 7, 10, 20, 30, 40}
Restriktion: MinContrast < Contrast

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {[], 'min_size', 'part_size'}

▷ **GenParamValue** (input_control) . . . . . integer-array ⇝ *integer* / real / string
Werte der Parameter.
Defaultwert: []
Werteliste: GenParamValue ∈ {[], 'small', 'medium', 'big'}

▷ **ModelID** (output_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_planar_uncalib_deformable_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`determine_deformable_model_params`

Nachfolger

`set_deformable_model_origin`, `set_deformable_model_param`, `get_deformable_model_contours`, `find_planar_uncalib_deformable_model`, `get_deformable_model_params`, `write_deformable_model`, `clear_deformable_model`

Alternativen

`read_deformable_model`

Modul

Matching

create_planar_uncalib_deformable_model_xld (
Contours : : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleRMin, ScaleRMax, ScaleRStep, ScaleCMin, ScaleCMax, ScaleCStep, Optimization, Metric, MinContrast, GenParamName, GenParamValue : ModelID)

Vorbereiten eines deformierbaren Modells für planares unkalibriertes Matching aus XLD-Konturen.

`create_planar_uncalib_deformable_model_xld` erzeugt ein deformierbares Modell für planares, unkalibriertes Matching aus den XLD Konturen `Contours`. Die XLD Konturen repräsentieren dabei die Grauwertkanten des zu suchenden Objekts. Im Gegensatz zu dem Operator `create_planar_uncalib_deformable_model`, der ein Modell aus einem Bild erzeugt, erzeugt der Operator `create_planar_uncalib_deformable_model_xld` das deformierbare Modell aus XLD Konturen.

Das Modell wird für mehrere Pyramidenebenen generiert und im Speicher abgelegt. Der Ausgabeparameter `ModelID` ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von `find_planar_uncalib_deformable_model` verwendet wird.

Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt des kleinsten umschließenden Rechtecks der Konturen `Contours`, das parallel zu den Koordinatenachsen liegt, verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit `set_deformable_model_origin` gesetzt werden.

Eingabeparameter im Detail

NumLevels: Die Anzahl der Pyramidenebenen wird mit `NumLevels` festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von `NumLevels` ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens vier). Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entsteht, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert `create_planar_uncalib_deformable_model_xld` eine Fehlermeldung zurück. Falls für

NumLevels *'auto'* übergeben wird, wählt create_planar_uncalib_deformable_model_xld die Anzahl der Pyramidenstufen automatisch. Die gewählte Anzahl von Pyramidenstufen kann mit get_deformable_model_params abgefragt werden. In seltenen Fällen kann es vorkommen, dass create_planar_uncalib_deformable_model_xld die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Suchbild nicht erkannt wird oder dass sehr niedrige Parameter für MinScore oder Greediness in find_planar_uncalib_deformable_model selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in find_planar_uncalib_deformable_model kommen. In beiden Fällen sollte die Anzahl der Pyramidenstufen manuell gewählt werden.

Winkel- und Skalierungs-Parameter: Beachten Sie, dass die AngleStart, AngleExtent, ScaleRMax und ScaleCMax nicht von diesem Operator genutzt werden. Stattdessen müssen sie in dem Operator find_planar_uncalib_deformable_model spezifiziert werden. ScaleRMin und ScaleCMin können relevant sein, wenn Matches erwartet werden, die kleiner sind als die übergebenen XLD-Konturen Contours. Dann verringert sich eventuell die Anzahl der benutzten Pyramidenebenen, falls für NumLevels *'auto'* übergeben wird.

Im Normalfall können die Parameter AngleStep, ScaleRStep und ScaleCStep automatisch bestimmt werden - die berechneten Winkel- und Skalierungsschrittweiten können mit get_deformable_model_params ausgegeben werden. Mehr Informationen dazu, wie sich diese Parameter auswirken, können in der Dokumentation von find_planar_uncalib_deformable_model gefunden werden.

Optimization: Bei besonders großen Modellen kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters Optimization auf einen Wert ungleich *'none'* zu setzen. Falls Optimization = *'none'* gesetzt ist, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter Optimization reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei find_planar_uncalib_deformable_model notwendig werden, den Parameter Greediness auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen. Falls für Optimization *'auto'* übergeben wird, wählt create_planar_uncalib_deformable_model_xld die Reduktion der Punkte automatisch.

MinContrast: Mit MinContrast wird festgelegt, welchen Grauwertkontrast die Objektkonturen später bei der Erkennung mit find_planar_uncalib_deformable_model im Bild mindestens besitzen müssen. Mit anderen Worten stellt dieser Parameter somit eine Abgrenzung des Objektes von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte MinContrast auf 10 gesetzt werden. Falls mehrkanalige Bilder für das Modell und für die Suchbilder verwendet werden, und falls der Parameter Metric auf *'ignore_color_polarity'* gesetzt wird (siehe unten), muss das Rauschen in einem Kanal noch mit der Wurzel der Anzahl der Kanäle multipliziert werden, um MinContrast zu bestimmen. Falls die Grauwerte in einem Kanal z.B. in einem Bereich von 10 Graustufen schwanken, sollte MinContrast bei einem dreikanaligen Bild auf 17 gesetzt werden. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss MinContrast entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte MinContrast etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells zu gewährleisten.

Metric: Der Parameter Metric legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird. Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist. Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_planar_uncalib_deformable_model erhöht sich in diesem Fall geringfügig. Falls Metric = *'ignore_part_polarity'* oder Metric = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Die obigen vier Metriken können nur auf einkanalige Bilder angewendet werden. Falls in den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes verwendet. Falls Metric = *'ignore_color_polarity'*, wird das Modell auch dann gefunden, falls sich die Farbkontraste lokal ändern können. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün.

Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von `find_planar_uncalib_deformable_model` erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für `Metric` = *'ignore_color_polarity'* die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein. Es ist auch zu beachten, dass die beiden ersten Werte für `Metric` (*'use_polarity'* und *'ignore_global_polarity'*) nur dann verwendet werden können, wenn bei allen Konturen, die in `Contours` übergeben wurden, das Attribut *'edge_direction'* gesetzt ist, da es die Polarität der Kanten festlegt. Für weitere Informationen zu Konturattributen wie *'edge_direction'* siehe `get_contour_attrib_xld`. Ansonsten besteht die Möglichkeit, die beiden Werte für `Metric` mit dem Operator `set_planar_uncalib_deformable_model_metric` zu setzen, der die Polarität der Konturen aus einem Bild bestimmt.

Achtung

Im Gegensatz zum Operator `create_planar_uncalib_deformable_model` ist es nicht möglich die Mindestgröße der Modellkomponenten anzugeben. Um kleine Modellkomponenten im Modell zu vermeiden, können kurze Konturen vor dem Aufruf von `create_planar_uncalib_deformable_model_xld` mit dem Operator `select_contours_xld` eliminiert werden.

Parameter

▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *object*
Eingabekonturen, die zum Aufbau des Modells verwendet werden.

▷ **`NumLevels`** (input_control) ..... integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Wertevorschläge: `NumLevels` ∈ {'auto', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **`AngleStart`** (input_control) ..... angle.rad-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **`AngleExtent`** (input_control) ..... angle.rad-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **`AngleStep`** (input_control) ..... angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: `AngleStep` ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep > 0 && AngleStep <= pi / 16

▷ **`ScaleRMin`** (input_control) ..... number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleRMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **`ScaleRMax`** (input_control) ..... number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **`ScaleRStep`** (input_control) ..... number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.
Defaultwert: 'auto'
Wertevorschläge: `ScaleRStep` ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleRStep > 0

▷ **`ScaleCMin`** (input_control) ..... number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleCMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **`ScaleCMax`** (input_control) ..... number-array ⇝ *real*
Dieser Parameter wird nicht benutzt.
Defaultwert: []

▷ **ScaleCStep** (input_control) ..... number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleCStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleCStep > 0

▷ **Optimization** (input_control) ..... string(-array) ⇝ *string*
Art der Optimierung der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: Optimization ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **Metric** (input_control) ..... string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'ignore_local_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_part_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **MinContrast** (input_control) ..... number ⇝ *integer*
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 5
Wertevorschläge: MinContrast ∈ {1, 2, 3, 5, 7, 10, 20, 30, 40}

▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
Namen der Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {[], 'part_size'}

▷ **GenParamValue** (input_control) ..... integer-array ⇝ *integer* / real / string
Werte der Parameter.
Defaultwert: []
Werteliste: GenParamValue ∈ {[], 'small', 'medium', 'big'}

▷ **ModelID** (output_control) ..... deformable_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_planar_uncalib_deformable_model_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter NumLevels so gewählt wurde, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_contour_xld_dxf, edges_sub_pix, select_contours_xld

Nachfolger

find_planar_uncalib_deformable_model

Siehe auch

create_planar_uncalib_deformable_model

Modul

Matching

deserialize_deformable_model (: : SerializedItemHandle : ModelID)

Deserialisieren eines deformierbaren Formmodells.

deserialize_deformable_model deserialisiert ein deformierbares Formmodell, welches mit serialize_deformable_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte deformierbare Formmodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten deformierbaren Formmodell mit dem Handle ModelID gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **ModelID** (output_control) ..... deformable_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_deformable_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_deformable_model

Nachfolger

find_planar_uncalib_deformable_model, find_planar_calib_deformable_model, find_local_deformable_model

Siehe auch

create_planar_uncalib_deformable_model, create_planar_calib_deformable_model, create_local_deformable_model

Modul

Matching

```
determine_deformable_model_params ( Template : : NumLevels,
    AngleStart, AngleExtent, ScaleMin, ScaleMax, Optimization,
    Metric, Contrast, MinContrast, GenParamName, GenParamValue,
    Parameters : ParameterName, ParameterValue )
```

Berechnung der Parameter eines Formmodells.

determine_deformable_model_params berechnet bestimmte Parameter eines deformierbaren Formmodells automatisch aus dem Modellbild Template. Die zu berechnenden Parameter werden in Parameters angegeben. Mit determine_deformable_model_params lassen sich dieselben Parameter automatisch bestimmen, die auch mit create_planar_uncalib_deformable_model durch Setzen des jeweiligen Parameters auf *'auto'* automatisch bestimmt werden können: die Anzahl der Pyramidenebenen des Modells (Parameters = *'num_levels'*), die Winkelschrittweite (Parameters = *'angle_step'*), die Skalierungsschrittweite (Parameters = *'scale_step'* für isotrope Skalierungen bzw. *'scale_r_step'* und/oder *'scale_c_step'* für anisotrope Skalierungen), die Art der Optimierung (Parameters = *'optimization'*), der Schwellenwert (Parameters = *'contrast'*) bzw. die Hystereseschwellenwerte (Parameters = *'contrast_hyst'*) für den Kontrast, die Mindestgröße der Objektteile (Parameters = *'min_size'*) und der minimale Kontrast (Parameters = *'min_contrast'*). Es kann eine beliebige Kombination dieser Parameter bestimmt werden, indem ein Tupel von Werten in Parameters übergeben wird. Falls alle Parameter automatisch bestimmt werden sollen, kann zur Vereinfachung auch der Wert *'all'* übergeben werden. In diesem Fall werden beide Hystereseschwellenwerte bestimmt, d.h. der Operator verhält sich so, als ob *'contrast_hyst'* anstatt *'contrast'* übergeben worden wäre.

determine_deformable_model_params kann hauptsächlich dazu verwendet werden, die obigen Parameter automatisch zu bestimmen, bevor das Modell erzeugt wird. Dies ist vor allem in interaktiven Systemen nützlich, in denen dem Benutzer ein Vorschlag für diese Werte präsentiert wird, der Benutzer diese aber modifizieren kann.

Die berechneten Parameter werden in ParameterName und ParameterValue als Name der Parameter und den jeweiligen Werten zurückgeliefert. Dabei haben die Parameter in ParameterName dieselben Namen wie in Parameters, wobei natürlich der Wert *'all'* durch die entsprechenden Parameternamen ersetzt wird. Eine Ausnahme besteht jedoch für den Parameter *'contrast_hyst'*, für den zwei Werte *'contrast_low'* und *'contrast_high'* zurückgegeben werden.

Die restlichen Parameter (NumLevels, AngleStart, AngleExtent, ScaleMin, ScaleMax, Optimization, Metric, Contrast und MinContrast) haben dieselbe Bedeutung wie bei create_planar_uncalib_deformable_model und find_planar_uncalib_deformable_model. Die Beschreibung der Parameter kann dort nachgelesen werden. Diese Parameter werden von determine_deformable_model_params dazu verwendet, die zu bestimmenden Parameter auf dieselbe Art zu berechnen wie in create_planar_uncalib_deformable_model. Es ist zu beachten, dass in ScaleMin und ScaleMax jeweils ein Wert übergeben werden muss, falls die Parameter eines Formmodells mit isotroper Skalierung bestimmt werden sollen, falls also Parameters *'scale_step'* explizit oder implizit über *'all'* enthält. Falls die Parameter eines Formmodells mit anisotroper Skalierung bestimmt werden sollen, d.h. falls Parameters *'scale_r_step'* oder *'scale_c_step'* explizit oder implizit enthält, müssen in ScaleMin und ScaleMax jeweils zwei Werte angegeben werden. Dabei bezieht sich der jeweils erste Parameter auf die Skalierung in Zeilenrichtung, der jeweils zweite Parameter auf die Skalierung in Spaltenrichtung.

Zu beachten ist, dass determine_deformable_model_params einige Parameter enthält, die auch automatisch bestimmt werden können (NumLevels, Optimization, Contrast, MinContrast). Falls diese Parameter nicht automatisch bestimmt werden sollen, d.h. ihr Name wird nicht in ParameterName übergeben, müssen für die entsprechenden Parameter zulässige Werte übergeben werden. Insbesondere dürfen die entsprechenden Parameter nicht auf *'auto'* gesetzt sein. Sollen diese Parameter dagegen automatisch bestimmt werden, so werden die in den entsprechenden Parametern übergebenen Werte wie folgt behandelt: Sollen die Art der Optimierung oder die (Hysterese)-Schwellenwerte für den Kontrast automatisch bestimmt werden, d.h. ParameterName enthält die Werte *'optimization'* oder *'contrast'* (*'contrast_hyst'*), dann werden die in Optimization bzw. Contrast übergebenen Werte ignoriert. Sollen dagegen die Anzahl an Pyramidenebenen oder der minimale Kontrast automatisch bestimmt werden, d.h. ParameterName enthält die Werte *'num_levels'* oder *'min_contrast'*, dann kann HALCON zwar geeignete Werte ermitteln, zusätzlich besteht jedoch die Möglichkeit, eine obere bzw. untere Grenze anzugeben:

Falls die maximale Anzahl der Pyramidenstufen vorgegeben werden soll, kann NumLevels auf den betreffenden Wert gesetzt werden. Falls in diesem Fall Parameters den Wert *'num_levels'* enthält, kann der berechnete Wert der Anzahl der Pyramidenebenen höchstens NumLevels betragen. Falls NumLevels = *'auto'* (oder *0* für Rückwärtskompatibilität) gesetzt wird, wird die Anzahl der Pyramidenebenen ohne Beschränkung so hoch wie möglich bestimmt.

Falls der minimale Kontrast vorgegeben werden soll, kann MinContrast auf den betreffenden Wert gesetzt werden. Falls in diesem Fall Parameters den Wert *'min_contrast'* enthält, beträgt der berechnete Wert des minimalen Kontrastes mindestens MinContrast. Falls MinContrast = *'auto'* gesetzt wird, wird der minimale Kontrast ohne Beschränkung bestimmt.

Wenn der Benutzer ein kalibriertes, deformierbares Modell mit create_planar_calib_deformable_model erzeugen möchte, benötigt er die internen Kameraparameter für die korrekte Schätzung von ParameterValue. Die internen Parameter der Kamera können übergeben werden, indem GenParamName auf *'cam_param'* gesetzt wird und in GenParamValue das Tupel, das die Kamera Parameter enthält, übergeben wird.

Parameter

▷ **Template** (input_object) . . . . . (multichannel-)image ⇝ *object* : byte / uint2
Eingabebild, dessen Definitionsbereich für das Modell verwendet wird.

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: NumLevels ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Modells.
Defaultwert: -0.39
Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: AngleExtent ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **ScaleMin** (input_control) . . . . . number(-array) ⇝ *real*
Kleinste auftretende Skalierung des Modells.
Defaultwert: 0.9
Wertevorschläge: ScaleMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleMin > 0

▷ **ScaleMax** (input_control) . . . . . number(-array) ⇝ *real*
Größte auftretende Skalierung des Modells.
Defaultwert: 1.1
Wertevorschläge: ScaleMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleMax >= ScaleMin

▷ **Optimization** (input_control) . . . . . string ⇝ *string*
Art der Optimierung.
Defaultwert: 'auto'
Werteliste: Optimization ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **Contrast** (input_control) . . . . . number(-array) ⇝ *integer* / string
Schwellenwert bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Musterbild und optional Mindestgröße der Objektteile.
Defaultwert: 'auto'
Wertevorschläge: Contrast ∈ {'auto', 'auto_contrast', 'auto_contrast_hyst', 'auto_min_size', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer* / string
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: MinContrast ∈ {'auto', 1, 2, 3, 5, 7, 10, 20, 30, 40}
Restriktion: MinContrast < Contrast

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der allgemeinen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'cam_param'}

▷ **GenParamValue** (input_control) . . . . . number-array ⇝ *real* / integer / string
Werte der allgemeinen Parameter.
Defaultwert: []
Werteliste: GenParamValue ∈ {[]}

▷ **Parameters** (input_control) . . . . . string(-array) ⇝ *string*
Zu berechnende Parameter.
Defaultwert: 'all'
Werteliste: Parameters ∈ {'all', 'num_levels', 'angle_step', 'scale_step', 'scale_r_step', 'scale_c_step', 'optimization', 'contrast', 'contrast_hyst', 'min_size', 'min_contrast'}

▷ **ParameterName** (output_control) . . . . . string-array ⇝ *string*
Name des berechneten Parameters.

▷ **ParameterValue** (output_control) . . . . . number-array ⇝ *real* / integer
Wert des berechneten Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert determine_deformable_model_params den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn die Parameter NumLevels und Contrast so gewählt worden sind, dass das Modell zu wenige Punkte besitzt, oder nicht genügend signifikante Merkmale im Eingabebild vorhanden sind, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`create_planar_uncalib_deformable_model`, `create_planar_calib_deformable_model`, `create_local_deformable_model`

Siehe auch

`find_planar_uncalib_deformable_model`, `find_planar_calib_deformable_model`

Modul

Matching

```
find_local_deformable_model ( Image : ImageRectified, VectorField,
    DeformedContours : ModelID, AngleStart, AngleExtent, ScaleRMin,
    ScaleRMax, ScaleCMin, ScaleCMax, MinScore, NumMatches,
    MaxOverlap, NumLevels, Greediness, ResultType, GenParamName,
    GenParamValue : Score, Row, Column )
```

Suche nach den besten Matches eines lokal deformierbaren Modells in einem Bild.

Der Operator `find_local_deformable_model` sucht die besten `NumMatches` Instanzen eines lokal deformierbaren Modells `ModelID` in dem Suchbild `Image`. Die Modelle müssen vorher durch `create_local_deformable_model` oder `read_deformable_model` erzeugt worden sein.

Im Normalfall wird das Modell innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als `MinScore` erreichen würde. Außerdem kann es sein, dass das Modell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit `set_system('border_shape_models','true')` umgestellt werden, so dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als `MinScore` erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit `dilation_circle` zu vergrößern.

Der Operator `find_local_deformable_model` gibt die Zeilen `Row` und Spalten `Column` der Koordinaten einer gefundenen Instanz zurück. Zusätzlich kann ein rektifizierter Teil des Bildes, das entsprechende Vektorfeld und die Konturen der gefundenen deformierten Modellinstanz in `ImageRectified`, `VectorField` und `DeformedContours` zurückgegeben werden. Die Voreinstellung ist, dass diese ikonischen Objekte nicht zurückgegeben werden. In dem Fall das sie gebraucht werden muss der Parameter `ResultType` auf 'image_rectified', 'vector_field' und 'deformed_contours' gesetzt werden. Die Größe von `ImageRectified` und `VectorField` ist das kleinste achsenparallele Rechteck der ROI des Bildes, das verwendet wurde um das lokal deformierbare Model mit `create_local_deformable_model` zu erzeugen. Die Größe des rektifizierten Bildes und des Vektorfeldes kann in alle Richtungen durch setzen von `GenParamName` auf 'expand_border' und des entsprechenden `GenParamValue` (die Anzahl Pixel) vergrößert werden. Optional kann auch eine spezifische Richtung durch 'expand_border_top', 'expand_border_bottom', 'expand_border_left' und 'expand_border_right' gesetzt werden. Es ist zu beachten, dass das `VectorField` vom semantischen Typ 'vector_field_absolute' ist und damit in absoluten Koordinaten angegeben ist, und `convert_map_type` übergeben werden kann.

Die Glattheit der geschätzten Deformation kann durch setzen von `GenParamName` auf 'deformation_smoothness' angegeben werden. Der Wert für 'deformation_smoothness' kann als Größe eines Glättungsfilters der auf das Deformationsfeld angewendet wird, verstanden werden. Ein sehr kleiner Wert erwartet eine starke Deformation, ein sehr großer Wert ein rigides Objekt. Der minimale Wert für 'deformation_smoothness' ist 3, ein typischer Wert

ist 11. Für Fälle in denen nur sehr kleine globale Deformationen erwartet werden, sollte der Wert weiter erhöht werden.

Zusätzlich wird in `Score` die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist.

Für eine weitere Erklärung des deformierbaren Modells und seiner Parameter sei auf die Beschreibung in `find_planar_uncalib_deformable_model` verwiesen.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, in dem das Modell gefunden werden soll.

▷ **`ImageRectified`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
Rektifizierter Bildausschnitt.

▷ **`VectorField`** (output_object) ..... singlechannelimage(-array) ⇝ *object :* vector_field
Vektorfeld des gefundenen Modells.

▷ **`DeformedContours`** (output_object) ..... xld_cont-array ⇝ *object*
Die deformierten Konturen des gefundenen Modells.

▷ **`ModelID`** (input_control) ..... deformable_model ⇝ *handle*
Handle des Modells.

▷ **`AngleStart`** (input_control) ..... angle.rad ⇝ *real*
Kleinste auftretende Rotation des Modells.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **`AngleExtent`** (input_control) ..... angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
Restriktion: AngleExtent >= 0

▷ **`ScaleRMin`** (input_control) ..... number ⇝ *real*
Kleinste auftretende Skalierung des Modells in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleRMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **`ScaleRMax`** (input_control) ..... number ⇝ *real*
Größte auftretende Skalierung des Modells in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleRMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleRMax >= ScaleRMin

▷ **`ScaleCMin`** (input_control) ..... number ⇝ *real*
Kleinste auftretende Skalierung des Modells in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleCMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **`ScaleCMax`** (input_control) ..... number ⇝ *real*
Größte auftretende Skalierung des Modells in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleCMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleCMax >= ScaleCMin

▷ **`MinScore`** (input_control) ..... real ⇝ *real*
Minimale Bewertung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **`NumMatches`** (input_control) ..... integer ⇝ *integer*
Anzahl der zu findenden Instanzen des Modells (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Überlappung der zu findenden Instanzen des Modells.
Defaultwert: 1.0
Wertevorschläge: `MaxOverlap` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ `MaxOverlap` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen.
Defaultwert: 0
Werteliste: `NumLevels` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real ⇝ *real*
„Gierigkeit“ der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen“ werden).
Defaultwert: 0.9
Wertevorschläge: `Greediness` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ `Greediness` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **ResultType** (input_control) . . . . . string-array ⇝ *string*
Die zu berechnenden ikonischen Ergebnisse.
Defaultwert: []
Werteliste: `ResultType` ∈ {[], 'image_rectified', 'vector_field', 'deformed_contours'}

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {[], 'subpixel', 'angle_step', 'scale_r_step', 'scale_c_step', 'deformation_smoothness', 'expand_border', 'expand_border_top', 'expand_border_bottom', 'expand_border_left', 'expand_border_right'}

▷ **GenParamValue** (input_control) . . . . . integer-array ⇝ *integer* / real / string
Werte der generischen Parameter.
Defaultwert: []
Werteliste: `GenParamValue` ∈ {[], 'none', 'least_squares', 'least_squares_high', 'least_squares_very_high'}

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Instanzen des Modells.

▷ **Row** (output_control) . . . . . real-array ⇝ *real*
Zeilenkoordinate des gefundenen Modells.

▷ **Column** (output_control) . . . . . real-array ⇝ *real*
Spaltenkoordinate des gefundenen Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_local_deformable_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_local_deformable_model`, `create_local_deformable_model_xld`, `read_deformable_model`

Modul

Matching

find_planar_calib_deformable_model (Image : : ModelID, AngleStart, AngleExtent, ScaleRMin, ScaleRMax, ScaleCMin, ScaleCMax, MinScore, NumMatches, MaxOverlap, NumLevels, Greediness, GenParamName, GenParamValue : Pose, CovPose, Score)

Suche nach den besten Matches eines kalibrierten, deformierbaren Modells in einem Bild und bestimme ihre 3D Lage.

Der Operator find_planar_calib_deformable_model findet die besten NumMatches Instanzen eines projektiv verzerrten, deformierbaren Modells ModelID im Suchbild Image. Das Modell muss vorher durch Aufruf von create_planar_calib_deformable_model oder read_deformable_model erzeugt worden sein. Der Operator find_planar_calib_deformable_model liefert die 3D Lage Pose der gefundenen Instanzen im Kamerakoordinatensystem zurück.

Zusätzlich zu der Pose wird die Genauigkeit der sechs Positionsparameter in CovPose zurückgeliefert. Normalerweise enthält CovPose die 6 Standardabweichungen der Pose Parameter für jeden Match. Falls der Parameter *'cov_pose_mode'* auf *'covariances'* gesetzt wurde, enthält CovPose die 36 Werte der gesamten 6×6 Kovarianzmatrix der 6 Positionsparameter. Es ist zu beachten das dies nur die innere Genauigkeit darstellt und sie deshalb von der echten Genauigkeit abweichen kann. Schließlich wird eine Bewertung für jede Instanz in Score zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1, was eine angenäherte Messung ist, und beschreibt wie viel des Modells im Bild sichtbar ist.

Für eine weitere Erklärung des planaren, deformierbaren Modells und seiner Parameter sei auf die Beschreibung in find_planar_uncalib_deformable_model verwiesen.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, in dem das Modell gefunden werden soll.

▷ **ModelID** (input_control) ..... deformable_model ⇝ *handle*
Handle des Modells.

▷ **AngleStart** (input_control) ..... angle.rad ⇝ *real*
Kleinste auftretende Rotation des Modells.
Defaultwert: -0.39
Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.78, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) ..... angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.78
Wertevorschläge: AngleExtent ∈ {6.29, 3.14, 1.57, 0.78, 0.39, 0.0}
Restriktion: AngleExtent >= 0

▷ **ScaleRMin** (input_control) ..... number ⇝ *real*
Kleinste auftretende Skalierung des Modells in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: ScaleRMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **ScaleRMax** (input_control) ..... number ⇝ *real*
Größte auftretende Skalierung des Modells in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: ScaleRMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleRMax >= ScaleRMin

▷ **ScaleCMin** (input_control) ..... number ⇝ *real*
Kleinste auftretende Skalierung des Modells in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: ScaleCMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) ..... number ⇝ *real*
Größte auftretende Skalierung des Modells in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: ScaleCMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleCMax >= ScaleCMin

▷ **MinScore** (input_control) . . . . . real ⇝ *real*
Minimale Bewertung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: MinScore ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MinScore ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumMatches** (input_control) . . . . . integer ⇝ *integer*
Anzahl der zu findenden Instanzen des Modells (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: NumMatches ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Überlappung der zu findenden Instanzen des Modells.
Defaultwert: 1.0
Wertevorschläge: MaxOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MaxOverlap ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumLevels** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls |NumLevels| = 2).
Defaultwert: 0
Werteliste: NumLevels ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real ⇝ *real*
„Gierigkeit“ der Suchheuristik (0: sicher aber langsam; 1: schnell, aber Matches können „übersehen“ werden).
Defaultwert: 0.9
Wertevorschläge: Greediness ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ Greediness ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der allgemeinen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {[], ’subpixel’, ’angle_change_restriction’, ’aniso_scale_change_restriction’, ’angle_step’, ’scale_r_step’, ’scale_c_step’, ’cov_pose_mode’}

▷ **GenParamValue** (input_control) . . . . . integer-array ⇝ *integer* / real / string
Werte der allgemeinen Parameter.
Defaultwert: []
Werteliste: GenParamValue ∈ {[], ’least_squares’, ’least_squares_high’, ’least_squares_very_high’, ’standard_deviations’, ’covariances’}

▷ **Pose** (output_control) . . . . . pose(-array) ⇝ *real* / integer
Pose des Objekts.

▷ **CovPose** (output_control) . . . . . real-array ⇝ *real*
6 Standardabweichungen bzw. 36 Kovarianzen der Poseparameter.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Instanzen des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert find_planar_calib_deformable_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

create_planar_calib_deformable_model, read_deformable_model

Modul

Matching

find_planar_uncalib_deformable_model (Image : : ModelID, AngleStart, AngleExtent, ScaleRMin, ScaleRMax, ScaleCMin, ScaleCMax, MinScore, NumMatches, MaxOverlap, NumLevels, Greediness, GenParamName, GenParamValue : HomMat2D, Score)

Unkalibrierte Suche nach den besten Matches eines planaren deformierbaren Modells in einem Bild.

Der Operator find_planar_uncalib_deformable_model sucht die besten NumMatches Instanzen eines perspektivisch verzerrten deformierbaren Modells ModelID in dem Suchbild Image. Die Modelle müssen vorher durch create_planar_uncalib_deformable_model oder read_deformable_model erzeugt worden sein.

Die projektive Transformation (Homographie), die die Lage des gefundenen Matches beschreibt, wird in HomMat2D zurückgegeben. In dem Fall dass mehrere Objekte gefunden werden, werden die verschiedenen Homographien aneinandergefügt. Eine einzelne Homographie kann einfach durch tuple_select_range(HomMat2D,Index*9,(Index+1)*9-1, SelectedHomMat2D) extrahiert werden. Die verschiedenen Ergebnisse werden in abfallender Reihenfolge nach Score sortiert. Die Zeilen und Spaltenkoordinaten des Ursprungs des deformierbaren Modells im Suchbild können durch Aufruf von projective_trans_pixel(HomMat2D,0,0,Row,Column) bestimmt werden. Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_planar_uncalib_deformable_model verwendet wurde.

Ein anderer Ursprung kann mit set_deformable_model_origin gesetzt werden. Für Visualisierungszwecke können die Modellkonturen, die durch get_deformable_model_contours extrahiert wurden, mit Hilfe des Operators projective_trans_contour_xld und die Stelle, die durch HomMat2D gegeben ist, dargestellt werden.

Zusätzlich wird in Score die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Eingabeparameter im Detail

Image: Der Definitionsbereich des Bildes Image gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des deformierbaren Modells mit create_planar_uncalib_deformable_model verwendet wurde. Ein eventuell mit set_deformable_model_origin anders gesetzter Ursprung wird nicht berücksichtigt. Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn eine Bewertung größer als MinScore erreichen würde (siehe unten). Außerdem kann es sein, dass das Modell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit set_system('border_shape_models','true') umgestellt werden, so dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als MinScore erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit dilation_circle zu vergrößern.

Winkel- und Skalierungs-Parameter: Die Parameter AngleStart, AngleExtent, ScaleRMin, ScaleRMax, ScaleCMin und ScaleCMax legen einen grundsätzlichen Bereich bis zu einer anisotropen Transformation fest, der auf der obersten Pyramidenstufe des Bildes erschöpfend durchsucht wird. Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Modells im Bild fest, die erschöpfend durchsucht werden. ScaleRMin, ScaleRMax, ScaleCMin und ScaleCMax bestimmen den möglichen Bereich von anisotropen Skalierungen die im Bild erschöpfend

durchsucht werden. Eine Skalierung von 1 für beide Skalierungsfaktoren entspricht der Originalgröße des Modells.

Es ist zu beachten, dass der Operator `find_planar_uncalib_deformable_model` Objekte außerhalb dieses Bereiches, z.B. auch dann wenn das Objekt perspektivisch verzerrt ist, findet. Demzufolge sind die Bereichsparameter eine Art Hinweis für den Suchalgorithmus `find_planar_uncalib_deformable_model`, und ausgehend von diesem Bereich kann in einem größeren Bereich, der von der Anzahl der Pyramidenstufen wie auch vom Bildinhalt abhängt, detektiert werden. Es ist zum Beispiel wichtig zu beachten, dass kleine Skalierungsunterschiede ohne die explizite Angabe eines Skalierungsbereichs detektiert werden, was zu einer schnelleren Ausführungszeit führt.

Falls das Objekt signifikante Skalierungsunterschiede aufweist, aber nur kleine perspektivische Verzeichnungen, benötigt man nur eine isotrope Skalierung und `ScaleCMin` wie auch `ScaleCMax` sollten auf den Wert 1.0 gesetzt werden, um zu signalisieren, dass keine anisotrope Skalierung benötigt wird. Falls das Objekt auch unter deutlichen Verkippungen erkannt werden soll, benötigt man eine anisotrope Skalierung. In diesem Fall wird die anisotrope Skalierung durch `ScaleRMin` und `ScaleRMax` in Zeilenrichtung, und durch `ScaleCMin` und `ScaleCMax` in Spaltenrichtung eingestellt.

Es ist zu beachten, dass die Transformationen intern so behandelt werden, dass erst die Skalierungen und dann die Rotation angewendet werden. Daher sollte das Modell im Normalfall so ausgerichtet werden, dass es im Modellbild horizontal oder vertikal erscheint.

Zusätzlich verarbeitet `find_planar_uncalib_deformable_model` die Parameter 'angle_step', 'scale_r_step' und 'scale_c_step', die mit dem Operator `create_planar_uncalib_deformable_model` gesetzt werden können, oder aber, wie unten beschrieben, mit den generischen Parametern `GenParamName` und `GenParamValue`. In den meisten Fällen führen die Werte, die automatisch mit `create_planar_uncalib_deformable_model` berechnet werden können, zu guten Resultaten.

Der Parameter 'angle_step' bestimmt die Schrittweite innerhalb des ausgewählten Winkelbereichs. Der Parameter 'angle_step' sollte aufgrund der Größe des Objekts gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte 'angle_step' für kleinere Modelle größer gewählt werden. Falls `AngleExtent` kein ganzzahliges Vielfaches von 'angle_step' ist, wird 'angle_step' entsprechend angepasst. Die Parameter 'scale_r_step' und 'scale_c_step' bestimmen die Schrittweite innerhalb des Skalierungsbereichs. Wie auch 'angle_step' sollten 'scale_r_step' und 'scale_c_step' aufgrund der Größe des Modells gewählt werden. Falls der jeweilige Skalierungsbereich kein ganzzahliges Vielfaches von 'scale_r_step' und 'scale_c_step' ist, werden 'scale_r_step' und 'scale_c_step' entsprechend angepasst.

MinScore: Der Parameter `MinScore` bestimmt, welchen Score ein potenzieller Match mindestens haben muss, damit er als Instanz des Modells im Bild angesehen wird. Umso höher `MinScore` gewählt wird, umso schneller wird die Suche im Bild. Falls das Modell niemals verdeckt wird, kann `MinScore` sogar auf 0.8 oder sogar 0.9 gesetzt werden.

NumMatches: Die maximale Anzahl der Instanzen, die gefunden werden können, kann mit `NumMatches` angegeben werden. Falls mehr als `NumMatches` Instanzen einen Wert größer als `MinScore` haben, werden nur die besten `NumMatches` zurückgegeben. Falls weniger als `NumMatches` gefunden werden, wird nur diese Anzahl zurückgegeben, das heißt, dass der Parameter `MinScore` Vorrang vor `NumMatches` hat. Sollen alle Modellinstanzen, deren Bewertung `MinScore` übersteigt, im Bild gefunden werden, muss `NumMatches` auf *0* gesetzt werden. In seltenen Fällen muss `NumMatches` auf einen höheren Wert als die beabsichtigte Anzahl von gefundenen Instanzen gesetzt werden. Das ist z.B. der Fall, wenn ein kleiner Wert für `MinScore` gesetzt wurde.

Beim Tracken der Matches durch die Bildpyramide werden auf jeder Ebene Matches verworfen, die wenig vielversprechend sind. Dadurch ist es möglich, dass Matches verworfen werden, die auf der niedrigsten Pyramidenebene einen höheren Score gehabt hätten. Beispielsweise kann das dazu führen, dass der gefundene Match für `NumMatches` gleich *1* nicht mit dem Match mit dem höchsten Score übereinstimmt, der von `NumMatches` gleich *0* oder > *1* zurückgegeben wird.

Falls erwartet wird, dass mehrere Objekte mit einem ähnlichen Score gefunden werden können, aber nur das Modell mit dem höchsten Score zurückgegeben werden soll, kann es vorteilhaft sein, `NumMatches` zu erhöhen, und dann den Match mit dem höchsten Score auszuwählen.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter `MaxOverlap` kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden.

Falls sich zwei Instanzen um mehr als MaxOverlap überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe smallest_rectangle2). Bei MaxOverlap = 0 dürfen sich die gefundenen Instanzen nicht überlappen, bei MaxOverlap = 1 werden alle gefundenen Instanzen zurückgeliefert.

GenParamName, GenParamValue: Mit Hilfe der generischen Parameter GenParamName und GenParamValue können mögliche Parameter angepasst werden, die typischerweise nicht von dem Benutzer verändert werden müssen. Grundsätzlich wird die Position mit hoher Subpixelgenauigkeit (*'least_squares_very_high'*) durch eine Kleinste-Quadrate Ausgleichung berechnet, das heißt, das die Abstände von Modellpunkten zu ihren korrespondierenden Bildpunkten minimiert werden. Falls diese hohe Genauigkeit von einer Anwendung nicht benötigt wird, kann die Subpixelgenaue Verfeinerung verringert oder sogar ganz ausgeschaltet werden, um die Laufzeit zu verkürzen. Hierzu wird 'subpixel' in GenParamName und *'none'*, *'least_squares'* oder *'least_squares_high'* für GenParamValue übergeben. Eine weitere Anwendung für GenParamName und GenParamValue ist das Überschreiben der Diskretisierungsschritte des Suchraums 'angle_step', 'scale_r_step' und 'scale_c_step', die definiert wurden als das Model mit create_planar_uncalib_deformable_model erzeugt wurde.

Wie in create_planar_uncalib_deformable_model beschrieben, führt der deformierbare Matching-Algorithmus eine vollkommene Suche des Basissuchbereichs, der mit AngleStart, AngleExtent, ScaleRMin, ScaleRMax, ScaleCMin und ScaleCMax angegeben wird, aus. Um die Erkennung von sogar perspektivisch verzeichneten Objekten zu erlauben, wird eine zusätzliche Transformation geschätzt. Diese zusätzliche Transformation bring das Modell vom ursprünglichen Suchbereich in einen größeren, perspektivisch verzeichneten Suchbereich. Dadurch, dass perspektivische Verzeichnungen erlaubt werden, steigt das Risiko von falschen Matches. Eine zusätzliche Verwendung des Parameters GenParamName ist es, falsch als positiv Klassifizierte Matches zu verwerfen. Das ist zum Beispiel nötig, falls ein kleiner Wert in MinScore angegeben wurde und das Bild signifikanten Hintergrund mit ähnlichem Aussehen wie das Modell enthält.

Um beliebige, zufällige, perspektivische Verzeichnungen zu unterdrücken, können die Parameter *'angle_change_restriction'* und *'aniso_scale_change_restriction'* in GenParamName verwendet werden. Mit *'angle_change_restriction'* wird die maximal tolerierte Winkelverzeichnung eingeschränkt. Als Standardwert wird $\frac{\pi}{2}$ verwendet, welcher Ergebnisse mit beliebigen Deformationen zulässt. Wird für *'angle_change_restriction'* der Wert 0 gesetzt, wird keinerlei Deformation erlaubt. Es sollten Werte im Intervall $[0, \frac{\pi}{2}]$ verwendet werden. Der Parameter überprüft, ob ein Winkel von 90 Grad an den Ecken des an den Achsen ausgerichteten Rechtecks um die Modellpunkte um mehr als den korrespondierenden GenParamValue für die Modellinstanz verändert wird. Es ist zu beachten, dass dieser Parameter sowohl den affinen (die Scherung) als auch den perspektivischen Anteil der Transformation einschränkt. Zum Beispiel kann mit *'angle_change_restriction'* ein rechteckiges Modell davon abgehalten werden sich zu einem Parallelogramm oder Trapez zu verformen.

Mit dem Parameter *'aniso_scale_change_restriction'* kann der anisotrope Skalierungsanteil eingeschränkt werden (der kleinere Anteil geteilt durch den größeren Skalierungsfaktor). Der Wertebereich dieses Parameters reicht vom Standardwert 0.0, der beliebige Deformationen erlaubt, bis zu 1.0, wo keine Deformationen erlaubt sind. Ein Beispiel dafür ist, dass mit diesem Parameter ein quadratisches Modell nicht zu einem rechteckigen Modell verformt werden kann.

NumLevels: Mit NumLevels wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit create_planar_uncalib_deformable_model angegebenen Bereich beschnitten. Falls NumLevels als *0* angegeben wird, wird die mit create_planar_uncalib_deformable_model angegebene Anzahl verwendet.

Greediness: Der Parameter Greediness bestimmt, wie „gierig“ die Suche durchgeführt werden soll. Für Greediness = 0 wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden, immer findet, wenn die anderen Parameter passend gesetzt sind. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für Greediness = 1 wird eine unsichere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, dass das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für Greediness = 1 wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten Fällen wird das Formmodell für Greediness = 0.9 immer sicher gefunden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, in dem das Modell gefunden werden soll.

▷ **ModelID** (input_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Modells.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.78
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
Restriktion: AngleExtent >= 0

▷ **ScaleRMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Modells in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleRMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **ScaleRMax** (input_control) . . . . . number ⇝ *real*
Größte auftretende Skalierung des Modells in Zeilenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleRMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleRMax >= ScaleRMin

▷ **ScaleCMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Modells in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleCMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number ⇝ *real*
Größte auftretende Skalierung des Modells in Spaltenrichtung.
Defaultwert: 1.0
Wertevorschläge: `ScaleCMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleCMax >= ScaleCMin

▷ **MinScore** (input_control) . . . . . real ⇝ *real*
Minimale Bewertung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumMatches** (input_control) . . . . . integer ⇝ *integer*
Anzahl der zu findenden Instanzen des Modells (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Überlappung der zu findenden Instanzen des Modells.
Defaultwert: 1.0
Wertevorschläge: `MaxOverlap` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ `MaxOverlap` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumLevels** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls $|$`NumLevels`$| = 2$).
Defaultwert: 0
Werteliste: `NumLevels` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real ⇝ *real*
„Gierigkeit“ der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen“ werden).
Defaultwert: 0.9
Wertevorschläge: `Greediness` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ `Greediness` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen der allgemeinen Parameter.
Defaultwert: []
Werteliste: `GenParamName` ∈ {[], ’subpixel’, ’angle_change_restriction’, ’aniso_scale_change_restriction’, ’angle_step’, ’scale_r_step’, ’scale_c_step’}

▷ **GenParamValue** (input_control) . . . . . integer-array ⇝ *integer* / real / string
Werte der allgemeinen Parameter.
Defaultwert: []
Werteliste: `GenParamValue` ∈ {[], ’none’, ’least_squares’, ’least_squares_high’, ’least_squares_very_high’}

▷ **HomMat2D** (output_control) . . . . . hom_mat2d(-array) ⇝ *real*
Homographie zwischen Modell und gefundenem Objekt.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Instanzen des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_planar_uncalib_deformable_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_planar_uncalib_deformable_model`, `read_deformable_model`

Alternativen

`find_planar_calib_deformable_model`

Modul

Matching

get_deformable_model_contours (: ModelContours : ModelID, Level :)

Auslesen der Konturrepräsentation eines Formmodells.

Der Operator `get_deformable_model_contours` gibt in `ModelContours` eine Repräsentation des Formmodells `ModelID` als XLD-Konturen zurück. Der Parameter `Level` gibt an, für welche Pyramidenstufe des Modells die Konturrepräsentation zurückgegeben werden soll. Die Konturen können z.B. zur Visualisierung der gefundenen Instanzen des Modells verwendet werden.

Im Falle, dass das Modell mit `create_planar_calib_deformable_model_xld` erzeugt wurde, wird die Kontur des Modells standardmäßig im Weltkoordinatensystem in metrischem Maß zurückgegeben. Für die Visualisierung müssen die Konturen mit der zurückgelieferten Pose transformiert werden.

In allen anderen Fällen wird die Kontur des Modells standardmäßig im Bildkoordinatensystem im Pixelmaß zurückgegeben. Für den kalibrierten Fall handelt es sich dabei um das rektifizierte Bildkoordinatensystem. Das rektifizierte Bildkoordinatensystem entspricht dem Koordinatensystem eines Bildes, das man mit `change_radial_distortion_image` unter Verwendung der rektifizierten Kameraparameter erhält. Die rektifizierten Kameraparameter lassen sich mit `get_deformable_model_params` abfragen. Es ist darauf zu achten, dass die Lage von `ModelContours` so normalisiert worden ist, dass der Referenzpunkt des Modells

(siehe set_deformable_model_origin) an der Pixelposition $(0, 0)$ liegt. D.h., die Konturen müssen mit der Homographie lediglich an die gefundene Stelle transformiert werden.

Das Standardverhalten kann mit dem generischen Parameter *'get_deformable_model_contours_coord_system'* des Operators set_deformable_model_param geändert werden.

Parameter

- ▷ **ModelContours** (output_object) ..... xld_cont-array ⇝ *object*
  Konturrepräsentation des Formmodells.
- ▷ **ModelID** (input_control) ..... deformable_model ⇝ *handle*
  Handle des Modells.
- ▷ **Level** (input_control) ..... integer ⇝ *integer*
  Pyramidenebene, für die die Konturrepräsentation zurückgegeben werden soll.
  Defaultwert: 1
  Wertevorschläge: Level ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Restriktion: Level >= 1

Ergebnis

Ist der Handle des Modells gültig, dann liefert get_deformable_model_contours den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_planar_uncalib_deformable_model, create_planar_uncalib_deformable_model_xld, read_deformable_model, create_planar_calib_deformable_model, create_planar_calib_deformable_model_xld, create_local_deformable_model, create_local_deformable_model_xld

Siehe auch

set_deformable_model_param, find_planar_uncalib_deformable_model, find_planar_calib_deformable_model

Modul

Matching

get_deformable_model_origin (: : ModelID : Row, Column)

Auslesen des Ursprungs (Referenzpunktes) eines deformierbaren Modells.

Der Operator get_deformable_model_origin gibt den Ursprung (Referenzpunkt) des Formmodells ModelID zurück. Der Ursprung wird relativ zum Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_planar_uncalib_deformable_model verwendet wurde, angegeben. Ein Ursprung von (0,0) heißt also, dass der Schwerpunkt der Region des Modellbildes verwendet wird. Ein Ursprung von (-20,-40) heißt, dass der Ursprung links oberhalb des Schwerpunktes liegt.

Parameter

- ▷ **ModelID** (input_control) ..... deformable_model ⇝ *handle*
  Handle des Modells.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenkoordinate des Ursprungs des deformierbaren Modells.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenkoordinate des Ursprungs des deformierbaren Modells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `get_deformable_model_origin` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_planar_uncalib_deformable_model`, `read_deformable_model`, `set_deformable_model_origin`

Nachfolger

`find_planar_uncalib_deformable_model`

Siehe auch

`area_center`

Modul

Matching

get_deformable_model_params (: : ModelID, GenParamName : GenParamValue)

Auslesen der Parameter eines 3D-Formmodells.

Der Operator `get_deformable_model_params` erlaubt es die Parameter eines deformierbaren Modells zu ermitteln. Die Namen der gewünschten Parameter werden als generische Parameter in `GenParamName` übergeben und die korrespondierenden Werte in `GenParamValue` zurückgegeben.

Die folgenden Parameter können ermittelt werden:

'cam_param': Interne Parameter der Kamera, die für den kalibrierten Fall verwendet wird.

'cam_param_rect': Rektifizierte interne Parameter der Kamera, die für den kalibrierten Fall verwendet wird.

'reference_pose': Die Referenzposition und Orientierung des kalibrierten, deformierbaren Modells. Die zurückgelieferte Position beschreibt die Ebene in der das Model liegt, das der Benutzer übergeben hat.

'model_pose': Referenzposition und Orientierung des deformierbaren Modells. Die zurückgelieferte Position beschreibt die Position des internen verwendeten metrischen 3D Modells, das für die Modellgenerierung verwendet wird.

'angle_start': Der kleinste Rotationswinkel des Modells.

'angle_extent': Die Ausdehnung des Winkelbereichs des Modells.

'angle_step': Schrittweite der Winkel (Auflösung) des Modells.

'scale_r_min': Kleinste auftretende Skalierung des Musters in Zeilenrichtung.

'scale_r_max': Größte auftretende Skalierung des Musters in Zeilenrichtung.

'scale_r_step': Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.

'scale_c_min': Kleinste auftretende Skalierung des Musters in Spaltenrichtung.

'scale_c_max': Größte auftretende Skalierung des Musters in Spaltenrichtung.

'scale_c_step': Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.

'num_levels': Maximale Anzahl der Pyramidenebenen.

'optimization': Art der Optimierung der Modellgenerierung.

'min_size': Minimale Größe der Kanten eines Modells.

'min_contrast': Minimaler Kontrast der Objekte im Suchbild.

'metric': Das verwendete Ähnlichkeitsmaß.

'model_row': Zeilenkoordinate des Ursprungs im radial entzerrten Modellbild.

'model_col': Spaltenkoordinate des Ursprungs im radial entzerrten Modellbild.

'model_type': Die Art des Modells: 'planar_uncalib', 'planar_calib' oder 'local'.

'created_from_xld': Wurde das Model aus XLD erzeugt oder aus einem Bild: 'created_from_xld' oder 'created_from_image'.

'get_deformable_model_contours_coord_system': Koordinatensystem (*'image'* oder *'world'*), in dem die Konturen beim Aufruf des Operators `get_deformable_model_contours` zurückgegeben werden.

Eine detaillierte Beschreibung der Parameter kann bei dem Operator `create_planar_uncalib_deformable_model` und `create_planar_calib_deformable_model` oder bei `get_deformable_model_contours` gefunden werden.

Obwohl der Parameter `Contrast` bei der Erstellung des Modells automatisch bestimmt werden kann, ist es nicht möglich, seinen Wert mit dem Operator `get_deformable_model_params` abzufragen. Stattdessen soll der Operator `determine_deformable_model_params` verwendet werden, um seinen Wert abzufragen.

Es ist möglich, mehrere Parameterwerte durch einen Operatoraufruf abzufragen, indem ein Tupel von Namen aller gewünschten Parameter in `GenParamName` übergeben wird. Als Ergebnis wird ein Tupel der gleichen Länge in `GenParamValue` zurückgeliefert. Es ist zu beachten das dies nur für Parameter möglich ist, die nur einen Rückgabeparameter haben.

Parameter

▷ **`ModelID`** (input_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

▷ **`GenParamName`** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für das deformierbare Modell abgefragt werden sollen.
Defaultwert: 'angle_start'
Werteliste: `GenParamName` ∈ {'cam_param', 'cam_param_rect', 'angle_start', 'angle_extent', 'angle_step', 'scale_r_min', 'scale_r_max', 'scale_r_step', 'scale_c_min', 'scale_c_max', 'scale_c_step', 'optimization', 'metric', 'min_contrast', 'num_levels', 'min_size', 'reference_pose', 'model_pose', 'model_row', 'model_col', 'model_type', 'created_from_xld', 'get_deformable_model_contours_coord_system'}

▷ **`GenParamValue`** (output_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Werte der generischen Parameter.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_deformable_model_params` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_planar_uncalib_deformable_model`, `create_planar_calib_deformable_model`, `create_local_deformable_model`

Nachfolger

`find_planar_uncalib_deformable_model`, `find_planar_calib_deformable_model`

Siehe auch

`set_deformable_model_param`

Modul

Matching

`read_deformable_model` (: : FileName : ModelID)

Einlesen eines deformierbares Formmodells aus einer Datei.

read_deformable_model liest ein deformierbares Formmodell, das mit write_deformable_model geschrieben wurde, aus der Datei FileName ein. Die Dateiendung für das deformierbare Formmodell ist in HALCON 'dfm'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: .dfm

▷ **ModelID** (output_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

find_planar_uncalib_deformable_model, find_planar_calib_deformable_model, find_local_deformable_model

Siehe auch

create_planar_uncalib_deformable_model, create_planar_calib_deformable_model, create_local_deformable_model

Modul

Matching

serialize_deformable_model (: : ModelID : SerializedItemHandle)

Serialisieren eines deformierbaren Modells.

serialize_deformable_model serialisiert die Daten eines deformierbaren Formmodells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_deformable_model in eine Datei geschrieben werden. Das deformierbare Formmodell wird in dem Handle ModelID übergeben. Das serialisierte deformierbare Formmodell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_deformable_model wieder deserialisiert werden.

Parameter

▷ **ModelID** (input_control) . . . . . deformable_model ⇝ *handle*
Handle des zu speichernden Modells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_deformable_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_planar_uncalib_deformable_model, create_planar_calib_deformable_model, create_local_deformable_model

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_deformable_model

Modul

Matching

set_deformable_model_origin (: : ModelID, Row, Column :)

Setzen des Ursprungs (Referenzpunktes) eines deformierbaren Modells.

Der Operator set_deformable_model_origin setzt den Ursprung (Referenzpunkt) des Formmodells ModelID auf einen neuen Wert. Der Ursprung wird relativ zum Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_planar_uncalib_deformable_model verwendet wurde, angegeben. Ein Ursprung von (0,0) heißt also, dass der Schwerpunkt der Region des Modellbildes verwendet wird. Ein Ursprung von (-20,-40) heißt, dass der Ursprung links oberhalb des Schwerpunktes liegt. Falls ein deformierbares Modell mit create_planar_calib_deformable_model erzeugt wurde, wird die 3D Lage vom Ursprung durch eine entsprechende Translation verschoben. Das bedeutet, dass zukünftige Aufrufe von find_planar_calib_deformable_model einen entsprechenden Versatz beinhalten. Falls ein deformierbares Modell mit create_planar_calib_deformable_model_xld erzeugt wurde, wird die 3D Lage vom Ursprung direkt um den Versatz im Weltkoordinaten verschoben. In diesem Fall entspricht die Zeilenkoordinate der y Koordinate (von oben nach unten) und die Spaltenkoordinate der x Koordinate (von links nach rechts).

Parameter

- ▷ **ModelID** (input_control) ..... deformable_model ⇝ *handle*
  Handle des Modells.
- ▷ **Row** (input_control) ..... point.y ⇝ *real*
  Zeilenkoordinate des Ursprungs des deformierbaren Modells.
- ▷ **Column** (input_control) ..... point.x ⇝ *real*
  Spaltenkoordinate des Ursprungs des deformierbaren Modells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert set_deformable_model_origin den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_planar_uncalib_deformable_model, create_planar_calib_deformable_model, create_local_deformable_model, read_deformable_model

Nachfolger

find_planar_uncalib_deformable_model, find_planar_calib_deformable_model, get_deformable_model_origin

Siehe auch

area_center

Modul

Matching

set_deformable_model_param (: : ModelID, GenParamName, GenParamValue :)

Setzen ausgewählter Parameter eines deformierbaren Modells.

Der Operator set_deformable_model_param setzt ausgewählte Parameter im Modell ModelID. Die folgenden Parameter können modifiziert werden:

'min_contrast'

Setzt im Modell ModelID den minimalen Kontrast des Objekts in den Suchbildern. Damit wird der Wert *'min_contrast'*, der ursprünglich z.B. mit create_planar_uncalib_deformable_model gesetzt wurde, überschrieben. Falls das Modell ModelID aus einer Datei eingelesen wurde und diese Datei geändert werden soll, muss das Formmodell ModelID nochmals in die Datei geschrieben werden, nachdem der Operator set_deformable_model_param ausgeführt wurde.

'get_deformable_model_contours_coord_system'

Setzt im Modell ModelID das Koordinatensystem, in dem die Konturen beim Aufruf des Operators get_deformable_model_contours zurückgegeben werden. Wird als Wert *'world'* übergeben, so werden die Konturen im Weltkoordinatensystem in metrischem Maß zurückgegeben. Dies ist nur für den kalibrierten Fall möglich. Wird als Wert *'image'* übergeben, so werden die Konturen im Bildkoordinatensystem im Pixelmaß zurückgeben. Für den kalibrierten Fall handelt es sich dabei um das rektifizierte Bildkoordinatensystem. Das rektifizierte Bildkoordinatensystem entspricht dem Koordinatensystem eines Bildes, das man mit change_radial_distortion_image unter Verwendung der rektifizierten Kameraparameter erhält. Die rektifizierten Kameraparameter lassen sich mit get_deformable_model_params abfragen. Falls das Modell mit create_planar_calib_deformable_model_xld erzeugt wurde, ist dieser Wert standardmäßig auf *'world'* gesetzt, in allen anderen Fällen auf *'image'*.

Parameter

▷ **ModelID** (input_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Parameternamen.
Werteliste: GenParamName ∈ {'min_contrast', 'get_deformable_model_contours_coord_system'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Parameterwerte.
Wertevorschläge: GenParamValue ∈ {5, 6, 7, 8, 9, 10, 'image', 'world'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_deformable_model_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_planar_uncalib_deformable_model, create_planar_calib_deformable_model, create_local_deformable_model, read_deformable_model

Nachfolger

find_planar_uncalib_deformable_model, find_planar_calib_deformable_model

Siehe auch

get_deformable_model_contours, get_deformable_model_params

Modul

Matching

set_local_deformable_model_metric (Image, VectorField : : ModelID, Metric :)

Setzen der Metrik eines lokal deformierbaren Modells, das aus XLD-Konturen erzeugt wurde.

set_local_deformable_model_metric ermöglicht eine Änderung des Wertes des Parameters Metric, der bei der Erstellung des Formmodells ModelID gesetzt wurde. Weil keine Information über die Polarität von Modellkonturen aus XLD Konturen vorhanden ist, wird die Polarität unter Verwendung des Bildes Image bestimmt. Dafür werden die Modellkonturen an der entsprechenden Position an der das Objekt im Bild Image erscheint, abgebildet. Hierfür wird das VectorField benötigt, welches vom semantischen Typ 'vector_field_absolute' sein muss und mit einem vorherigen Aufruf von find_local_deformable_model ermittelt wird. Es ist zu beachten, das die 'expand_border' Option in find_local_deformable_model nicht verwendet werden darf, da sonst das zurückgelieferte VectorField die falsche Größe hat.

Der Parameter Metric legt fest, unter welchen Bedingungen das Muster später im Suchbild noch erkannt wird. Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist. Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_local_deformable_model erhöht sich in diesem Fall geringfügig.

Es muss sichergestellt werden, dass die Objektkonturen im Bild Image die gleiche (oder inverse) Polarität aufweisen wie die entsprechenden Konturen im späteren Suchbild. Insbesondere darf das Objekt nicht verdeckt sein und der Hintergrund muss entweder heller oder dunkler sein als das Objekt. Ansonsten stimmt die Polarität der Modellkanten nicht mit der Polarität der Kanten im Suchbild überein und das Formmodell kann dort nicht gefunden werden. Zu beachten ist, dass nur die Polarität der Konturen bestimmt wird, nicht ihr Kontrast. Des Weiteren ist zu beachten, dass die Polarität nur aus einem einkanaligen Bild bestimmt wird. Wird ein Mehrkanalbild in Image übergeben, so wird nur der erste Kanal verwendet (und keine Fehlermeldung zurückgegeben).

Eine typische Vorgehensweise besteht darin, zuerst die XLD-Konturen aus einer Datei einzulesen. Da diese XLD-Konturen keine Polaritätsinformation haben, muss der Parameter Metric bei der Erzeugung des Formmodells aus den XLD-Konturen auf den Wert *'ignore_local_polarity'* gesetzt werden. Dann kann das Modell in einem ersten Suchbild gefunden werden. Aus den resultierenden Matchingergebnissen lassen sich die Transformationsparameter (VectorField) bestimmen, die die Modellkonturen an der Stelle des Suchbildes abbilden, an der das Objekt erscheint. Ist das Matchingergebnis korrekt, kann der Wert des Parameters Metric z.B. auf *'use_polarity'* gesetzt werden. Dies führt in den nachfolgenden Bildern zu einer schnelleren und robusteren Erkennung.

Achtung

set_local_deformable_model_metric kann nur mit Modellen genutzt werden, die aus XLD-Konturen erzeugt wurden.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild, das zur Bestimmung der Polarität verwendet wird.
- ▷ **VectorField** (input_object) ..... singlechannelimage ⇝ *object :* vector_field
  Vektorfeld der lokalen Deformation.
- ▷ **ModelID** (input_control) ..... deformable_model ⇝ *handle*
  Handle des Modells.
- ▷ **Metric** (input_control) ..... string ⇝ *string*
  Art der zum Matchen verwendeten Metrik.
  Defaultwert: 'use_polarity'
  Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_local_deformable_model_metric den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_local_deformable_model_xld`, `find_local_deformable_model`

Nachfolger

`find_local_deformable_model`

Siehe auch

`create_local_deformable_model_xld`

Modul

Matching

set_planar_calib_deformable_model_metric (Image : : ModelID, Pose, Metric :)

Setzen der Metrik eines planaren kalibrierten deformierbaren Modells, das aus XLD-Konturen erzeugt wurde.

`set_planar_calib_deformable_model_metric` ändert den Wert des Parameters `Metric` der bei der Erstellung des Formmodells `ModelID` gesetzt wurde. Weil keine Information über die Polarität von Modellkonturen aus XLD Konturen vorhanden ist, wird die Polarität unter Verwendung des Bildes `Image` bestimmt. Dafür werden die Modellkonturen and die entsprechende Position an der das Objekt im Bild `Image` erscheint mit Hilfe von `Pose` abgebildet. `Pose` kann mit einem vorherigen Aufruf von `find_planar_uncalib_deformable_model` ermittelt werden.

Der Parameter `Metric` legt fest, unter welchen Bedingungen das Muster später im Suchbild noch erkannt wird. Falls `Metric` = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist. Falls `Metric` = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von `find_planar_calib_deformable_model` erhöht sich in diesem Fall geringfügig.

Es muss sichergestellt werden, dass die Objektkonturen im Bild `Image` die gleiche (oder inverse) Polarität aufweisen wie die entsprechenden Konturen im späteren Suchbild. Insbesondere darf das Objekt nicht verdeckt sein und der Hintergrund muss entweder heller oder dunkler sein als das Objekt. Ansonsten stimmt die Polarität der Modellkonturen nicht mit der Polarität der Konturen im Suchbild überein und das Formmodell kann dort nicht gefunden werden. Zu beachten ist, dass nur die Polarität der Konturen bestimmt wird, nicht ihr Kontrast. Des Weiteren ist zu beachten, dass die Polarität nur aus einem einkanaligen Bild bestimmt wird. Wird ein Mehrkanalbild in `Image` übergeben, so wird nur der erste Kanal verwendet (und keine Fehlermeldung zurückgegeben).

Eine typische Vorgehensweise besteht darin, zuerst die XLD-Konturen aus einer Datei einzulesen. Da diese XLD-Konturen keine Polaritätsinformation haben, muss der Parameter `Metric` bei der Erzeugung des Formmodells aus den XLD-Konturen auf den Wert *'ignore_local_polarity'* gesetzt werden. Dann kann das Modell in einem ersten Suchbild gefunden werden. Aus den resultierenden Matchingergebnissen lassen sich die Transformationsparameter bestimmen, die die Modellkonturen an die Stelle des Suchbildes abbilden, an der das Objekt erscheint. Um die Matchingergebnisse interaktiv zu kontrollieren, können die Modellkonturen entsprechend transformiert werden. Ist das Matchingergebnis korrekt, kann der Wert des Parameters `Metric` z.B. zu *'use_polarity'* verändert werden. Dies führt in den nachfolgenden Bildern zu einer schnelleren und robusteren Erkennung.

Achtung

`set_planar_calib_deformable_model_metric` kann nur mit Modellen genutzt werden, die aus XLD-Konturen erzeugt wurden.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Eingabebild, das zur Bestimmung der Polarität verwendet wird.

▷ **ModelID** (input_control) . . . . . deformable_model ⇝ *handle*
Handle des Modells.

▷ **Pose** (input_control) . . . . . pose ⇝ *real* / integer
Lage des Modells im Bild.
Parameteranzahl: 7

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_planar_calib_deformable_model_metric` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_planar_calib_deformable_model_xld`, `find_planar_calib_deformable_model`

Nachfolger

`find_planar_calib_deformable_model`

Siehe auch

`create_planar_calib_deformable_model_xld`

Modul

Matching

set_planar_uncalib_deformable_model_metric (Image : : ModelID, HomMat2D, Metric :)

Setzen der Metrik eines planaren unkalibrierten deformierbaren Modells, das aus XLD-Konturen erzeugt wurde.

`set_planar_uncalib_deformable_model_metric` ändert den Wert des Parameters `Metric` der bei der Erstellung des Formmodells `ModelID` aus XLD Konturen gesetzt wurde. Weil keine Information über die Polarität von Modellkonturen aus XLD Konturen vorhanden ist, wird die Polarität unter Verwendung des Bildes `Image` bestimmt. Dafür werden die Modellkonturen und die entsprechende Position an der das Objekt im Bild `Image` erscheint mit Hilfe von `HomMat2D` abgebildet. `HomMat2D` kann mit einem vorherigen Aufruf von `find_planar_uncalib_deformable_model` ermittelt werden.

Der Parameter `Metric` legt fest, unter welchen Bedingungen das Muster später im Suchbild noch erkannt wird. Falls `Metric` = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist. Falls `Metric` = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von `find_planar_uncalib_deformable_model` erhöht sich in diesem Fall geringfügig.

Es muss sichergestellt werden, dass die Objektkonturen im Bild `Image` die gleiche (oder inverse) Polarität aufweisen wie die entsprechenden Konturen im späteren Suchbild. Insbesondere darf das Objekt nicht verdeckt sein und der Hintergrund muss entweder heller oder dunkler sein als das Objekt. Ansonsten stimmt die Polarität der Modellkanten nicht mit der Polarität der Kanten im Suchbild überein und das Formmodell kann dort nicht gefunden werden. Zu beachten ist, dass nur die Polarität der Konturen bestimmt wird, nicht ihr Kontrast. Des Weiteren ist zu beachten, dass die Polarität nur aus einem einkanaligen Bild bestimmt wird. Wird ein Mehrkanalbild in `Image` übergeben, so wird nur der erste Kanal verwendet (und keine Fehlermeldung zurückgegeben).

Eine typische Vorgehensweise besteht darin, zuerst die XLD-Konturen aus einer Datei einzulesen. Da diese XLD-Konturen keine Polaritätsinformation haben, muss der Parameter `Metric` bei der Erzeugung des Formmodells aus den XLD-Konturen auf den Wert *'ignore_local_polarity'* gesetzt werden. Dann kann das Modell in einem ersten Suchbild gefunden werden. Aus den resultierenden Matchingergebnissen lassen sich die Transformationsparameter (`HomMat2D`) bestimmen, die die Modellkonturen an die Stelle des Suchbildes abbilden, an der das Objekt erscheint. Um die Matchingergebnisse interaktiv zu kontrollieren, können die Modellkonturen entsprechend transformiert werden. Ist das Matchingergebnis korrekt, kann der Wert des Parameters `Metric` z.B. zu *'use_polarity'* verändert werden. Dies führt in den nachfolgenden Bildern zu einer schnelleren und robusteren Erkennung.

Achtung

`set_planar_uncalib_deformable_model_metric` kann nur mit Modellen genutzt werden, die aus XLD-Konturen erzeugt wurden.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild, das zur Bestimmung der Polarität verwendet wird.
- ▷ **`ModelID`** (input_control) ..... deformable_model ⇝ *handle*
  Handle des Modells.
- ▷ **`HomMat2D`** (input_control) ..... hom_mat2d ⇝ *real*
  Transformationsmatrix.
- ▷ **`Metric`** (input_control) ..... string ⇝ *string*
  Art der zum Matchen verwendeten Metrik.
  Defaultwert: 'use_polarity'
  Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_planar_uncalib_deformable_model_metric` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_planar_uncalib_deformable_model_xld`,
`find_planar_uncalib_deformable_model`

Nachfolger

`find_planar_uncalib_deformable_model`

Siehe auch

`create_planar_uncalib_deformable_model_xld`

Modul

Matching

write_deformable_model (: : ModelID, FileName :)

Schreiben eines deformierbaren Modells in eine Datei.

write_deformable_model schreibt ein deformierbares Formmodell in die Datei FileName. Das Modell kann mit read_deformable_model wieder eingelesen werden. Die Dateiendung für das deformierbare Formmodell ist in HALCON 'dfm'.

Parameter

▷ **ModelID** (input_control) . . . . . deformable_model ⇝ *handle*
Handle des zu speichernden Modells.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Der Pfad sowie Name des zu speichernden Modells.
Dateiendung: .dfm

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_planar_uncalib_deformable_model, create_planar_calib_deformable_model, create_local_deformable_model

Modul

Matching

17.2 Deskriptorbasiert

clear_descriptor_model (: : ModelID :)

Freigeben des Speichers eines Deskriptormodells.

clear_descriptor_model gibt den Speicher eines Deskriptormodells, das mit create_calib_descriptor_model oder create_uncalib_descriptor_model angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle ModelID ist nach dem Aufruf ungültig.

Parameter

▷ **ModelID** (input_control) . . . . . descriptor_model(-array) ⇝ *handle*
Handle des Deskriptormodells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert clear_descriptor_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

`create_uncalib_descriptor_model`, `create_calib_descriptor_model`

Modul

Matching

`create_calib_descriptor_model` (`Template` : : `CamParam`, `ReferencePose`, `DetectorType`, `DetectorParamName`, `DetectorParamValue`, `DescriptorParamName`, `DescriptorParamValue`, `Seed` : `ModelID`)

Erzeugen eines Deskriptormodells für Punkt-Matching und Objektposenbestimmung.

Der Operator `create_calib_descriptor_model` erstellt aus einer Region, die als Bild in `Template` übergeben wird, ein Deskriptormodell, mit dessen Hilfe kalibriertes deskriptorbasiertes Matching durchgeführt werden kann. Es ist zu beachten, dass der Teil des Objektes, der in dem `Template`-Bild zu sehen ist, planar sein muss.

`ReferencePose` beschreibt die 3D Position und Orientierung der Objektebene, welche im Referenzbild `Template` abgebildet ist. Die internen Kameraparameter `CamParam` und die Referenzpose werden intern dazu genutzt, ein Modellkoordinatensystem zu erstellen. Der Ursprung dieses Modellkoordinatensystems wird dabei durch die Projektion des Schwerpunkts einer intern entzerrten Version des Referenzbildes auf die Objektebene gebildet. Die Achsen des Modellkoordinatensystems liegen parallel zu den Achsen von Reference Pose. Um ein Deskriptormodell aus dem Referenzbild `Template` zu generieren, werden die durch einen Detektor im entzerrten Referenzbild extrahierten Punkte ebenfalls auf die Objektebene projiziert. Ihre Koordinaten im Modellkoordinatensystem werden als Weltkoordinaten im Modell gespeichert, um später für die Schätzung der Pose eines Suchobjekts verwendet zu werden (siehe `find_calib_descriptor_model`).

Ein mit Hilfe des Operators `create_calib_descriptor_model` erstelltes Deskriptormodell kann auch mit dem Operator `find_uncalib_descriptor_model` zur Ermittlung einer Homographie verwendet werden. Ein mit dem Operator `create_uncalib_descriptor_model` erstelltes Modell kann jedoch nicht in `find_calib_descriptor_model` verwendet werden.

Das Deskriptormodell beschreibt einen Satz von Merkmalspunkten. Es speichert die Punktpositionen, sowie unterschiedliche Beschreibungen ihrer lokalen Bildumgebung. Die Extraktion dieser Merkmalspunkte wird durch `DetectorType`, `DetectorParamName` und `DetectorParamValue` parametrisiert. Der zugehörige Deskriptor für die Umgebung der Merkmalspunkte wird durch `DescriptorParamName` und `DescriptorParamValue` parametrisiert. Derzeit wird der Deskriptor durch *randomized ferns* implementiert. Der Parameter `Seed` initialisiert den Zufallszahlengenerator, der für den Aufbau der *randomized ferns* Strukturen verwendet wird. Ist das Deskriptormodell generiert, wird dessen Referenz in `ModelID` zurückgegeben. Weitere Informationen über diese Parameter können unter der Beschreibung des Operators `create_uncalib_descriptor_model` gefunden werden.

Alle Parameter sowie die Koordinaten der letztendlichen Merkmalspunkte können über `get_descriptor_model_params` und `get_descriptor_model_points` abgerufen werden.

`create_calib_descriptor_model` legt den Detektor und den Deskriptor fest. Somit werden beide auch bei jedem nachfolgenden Aufruf von `find_calib_descriptor_model` verwendet. Der Referenzpunkt (Ursprung) des Modells ist der Schwerpunkt der ROI des Templates. Er kann über `set_descriptor_model_origin` geändert werden.

Parameter

- ▷ **`Template`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.
- ▷ **`CamParam`** (input_control) ..... campar ⇝ *real* / integer / string
  Die internen Kameraparameter der Kamera.
- ▷ **`ReferencePose`** (input_control) ..... pose ⇝ *real* / integer
  Die Referenzpose des Objekts im Referenzbild.

▷ **DetectorType** (input_control) . . . . . string ⇝ *string*
Art des Detektors.
Defaultwert: 'lepetit'
Werteliste: DetectorType ∈ {'lepetit', 'harris', 'harris_binomial'}

▷ **DetectorParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der Detektor-Parameter.
Defaultwert: []
Werteliste: DetectorParamName ∈ {'alpha', 'check_neighbor', 'mask_size_grd', 'mask_size_smooth', 'min_check_neighbor_diff', 'min_score', 'radius', 'sigma_grad', 'sigma_smooth', 'subpix', 'threshold'}

▷ **DetectorParamValue** (input_control) . . . . . attribute.value-array ⇝ *integer* / real / string
Werte der Detektor-Parameter.
Defaultwert: []
Wertevorschläge: DetectorParamValue ∈ {0.08, 1, 1.2, 3, 15, 30, 1000, 'on', 'off'}

▷ **DescriptorParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der Deskriptor-Parameter.
Defaultwert: []
Werteliste: DescriptorParamName ∈ {'depth', 'max_rot', 'max_scale', 'min_rot', 'min_scale', 'number_ferns', 'patch_size', 'tilt'}

▷ **DescriptorParamValue** (input_control) . . . . . attribute.value-array ⇝ *integer* / real / string
Werte der Deskriptor-Parameter.
Defaultwert: []
Wertevorschläge: DescriptorParamValue ∈ {0.5, 1.4, 11, 21, 30, -180, 180, 'on', 'off'}

▷ **Seed** (input_control) . . . . . integer ⇝ *integer*
Initialisierung des Zufallsgenerators.
Defaultwert: 42

▷ **ModelID** (output_control) . . . . . descriptor_model ⇝ *handle*
Der Handle des Deskriptormodells

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

points_lepetit, points_harris, reduce_domain

Nachfolger

get_descriptor_model_params, find_calib_descriptor_model

Siehe auch

get_descriptor_model_params, find_calib_descriptor_model

Literatur

V. Lepetit and P. Fua: „Keypoint Recognition using Randomized Trees.„ IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 28, Nr. 9, pp. 1465-1479, 2006.

M. Ozuysal, P. Fua, and V. Lepetit: „Fast Keypoint Recognition in Ten Lines of Code.„ In Proceedings of Conference on Computer Vision and Pattern Recognition, 2007.

Modul

Matching

create_uncalib_descriptor_model (Template : : DetectorType, DetectorParamName, DetectorParamValue, DescriptorParamName, DescriptorParamValue, Seed : ModelID)

Vorbereiten eines Deskriptormodells für Punkt-Matching.

Der Operator create_uncalib_descriptor_model erstellt ein Deskriptormodell einer Bildregion, die als das Bild Template übergeben wird. Mit Hilfe dieses Modells kann deskriptorbasiertes Matching durchgeführt werden. Durch anschließende Verwendung des Operators find_uncalib_descriptor_model kann damit eine projektive 2D Abbildung (Homographie) von Template zu einem Suchbild ermittelt werden. Der Schwerpunkt der Bildregion im Template wird als Ursprung des Models genommen. Im Gegensatz zu create_calib_descriptor_model ist hier keine kalibrierte Kamera nötig, allerdings ist das Ergebnis eines nachfolgenden Matchings lediglich eine 2D Projektion. Es ist zu beachten, dass der Teil des Objektes, der in dem Template-Bild zu sehen ist, planar sein muss.

Das Deskriptormodell beschreibt einen Satz von Merkmalspunkten. Es speichert die Punktpositionen, sowie unterschiedlichen Beschreibungen ihrer lokalen Bildumgebung. Die Extraktion dieser Merkmalspunkte wird durch DetectorType, DetectorParamName und DetectorParamValue parametrisiert. Der zugehörige Deskriptor für die Umgebung der Merkmalspunkte wird durch DescriptorParamName und DescriptorParamValue parametrisiert. Derzeit wird der Deskriptor durch *randomized ferns* implementiert. Der Parameter Seed initialisiert den Zufallszahlengenerator, der für den Aufbau der *randomized ferns* Strukturen verwendet wird. Ist das Deskriptormodell generiert, wird dessen Referenz in ModelID zurückgegeben. Durch Verwendung dieses Modells in find_uncalib_descriptor_model kann ein schnelles, projektivinvariantes Matching zwischen Modell- und Suchbild gewährleistet werden. Da das deskriptorbasierte Matching auf der Existenz von stabilen und repräsentativen Merkmalspunkten basiert, muss das detektierte Objekt ausreichend Textur aufweisen, die sich nicht wiederholt.

Detektorparameter

Wie bereits erwähnt, ist der Detektor dafür verantwortlich, stabile Merkmalspunkte aus dem Modellbild zu extrahieren. Durch den Parameter DetectorType kann festgelegt werden, welcher Punktoperator verwendet werden soll. Derzeit werden points_lepetit, points_harris oder dessen binomiale Approximation points_harris_binomial unterstützt (*'lepetit'*, *'harris'*, *'harris_binomial'*). Bei Anwendung auf dunkle Modell- / Suchbilder wird die Verwendung eines der Harris-Punktoperatoren empfohlen. Je nach Wahl des Punktoperators DetectorType müssen dessen Parameter eingestellt werden, was über DetectorParamName und DetectorParamValue möglich ist.

Gültige Parameternamen für DetectorParamName und die entsprechenden Defaultwerte sind:

'lepetit' **:** ['radius','check_neighbor', 'min_check_neighbor_diff','min_score','subpix']
[3, 1, 15, 30, 'interpolation']

'harris' **:** ['sigma_grad','sigma_smooth','alpha', 'threshold']
[0.7, 2.0, 0.08, 1000]

'harris_binomial' **:** ['mask_size_grd','mask_size_smooth', 'alpha','threshold','subpix']
[5, 15, 0.08, 1000, 'on']

Weitere Informationen über diese Parameter können unter der Beschreibung von points_harris, points_harris_binomial beziehungsweise points_lepetit gefunden werden. Für die Parameter, die nicht explizit durch DetectorParamName und DetectorParamValue gesetzt sind, werden die oben genannten Werte für den entsprechenden Punktoperator genommen. Dementsprechend, wenn sowohl DetectorParamName als auch DetectorParamValue auf [] gesetzt sind, werden diese Defaultwerte für alle Parameter übernommen.

Beim Optimieren der Punktoperator-Parameter sollten zwischen 50 und 450 Merkmalspunkte extrahiert werden (je nach Textur und Bildgröße von Template). In den meisten Fällen ist es jedoch ausreichend, die Defaultwerte zu übernehmen ([] in DetectorParamName und DetectorParamValue eingeben).

Deskriptorparameter

Der Punktdeskriptor ist ein Klassifikator, der charakteristische Beschreibungen der Grauwertnachbarschaft der Merkmalspunkte aufbaut. Derzeit ist der Deskriptor durch die so genannten *randomized ferns* implementiert. Diese entnehmen die Polarität der Grauwertdifferenzen von *zufällig* ausgewählten Pixelpaaren aus der Nachbarschaft des Punktes. Dieser Deskriptor wird danach in find_uncalib_descriptor_model verwendet, um Merkmalspunkte im Suchbild zu klassifizieren, oder mit anderen Worten: um potentielle Modellpunkte auch im Suchbild zu erkennen (matchen).

Der Deskriptor speichert nur die projektiv *stabilen* Merkmalspunkte (die in vielen projektiven Ansichten des Modellbildes auftauchen). Um die *Stabilität* einzelner Punkte zu bewerten, wird die folgende Simulation durchgeführt: Template wird vielen affinen Transformationen ausgesetzt, und die Punkte, die in den meisten Ansichten

erscheinen, werden als *stabil* eingestuft. Die affinen Transformationen gelten als eine gute Näherung zu projektiven Transformationen in der lokalen Umgebung der Punkten.

Die folgenden Parameter des Deskriptors und seines Trainings können mit `DescriptorParamName` und `DescriptorParamValue` gesetzt werden:

Deskriptorstrukturen-Parameter:

'depth': Tiefe des Farns, der für die Punktklassifikation verwendet wird. Ein tieferer Farn kann Merkmalspunkte besser unterscheiden. Andererseits steigt der Speicherbedarf bei Erhöhung der Tiefe um 2 an.
Typische Werte sind [*5 ... 11*], Defaultwert ist *11*.

'number_ferns': Anzahl der Farne. Mehr Farne erhöhen die Erkennungsrobustheit, aber auch die Laufzeit der Punktdetektion. Soll der Speicherverbrauch des Deskriptormodells minimiert werden, sind viele Farne mit geringer Tiefe empfohlen (z.B. *'number_ferns'=150*, *'depth'= 5*). Soll hingegen die Detektionszeit verkürzt werden sind wenige Farne mit größerer Tiefe empfohlen (z.B. *'number_ferns'=10*, *'depth'=11*).
Typische Werte sind [*1 ... 150*], Defaultwert ist *30*.

'patch_size': Seitenlänge der quadratischen Umgebung eines Merkmalspunktes. Höhere Werte dieses Parameters können zu längeren Trainingszeiten führen.
Typische Werte sind [*15 ... 33*], Defaultwert ist *17*.

Die Parameter *'depth'*, *'number_ferns'* und *'patch_size'* erlauben eine transparente Kontrolle über die Robustheit der Erkennung, die Geschwindigkeit und den Speicherverbrauch.

Simulationparameter:

'tilt': Schalter für projektive Transformationen während des Simulationsschrittes. Im eingeschalteten Zustand (*'on'*), verbessert sich die Robustheit des Modells, indem Objekte mit größerer Neigung gefunden werden können. Im ausgeschalteten Zustand (*'off'*) verkürzt sich die Trainingzeit deutlich, und das Modell kann immer noch projektiv transformierte Objekte erkennen.
Mögliche Werte sind [*'on'*, *'off'*], Defaultwert ist *'on'*.

'min_rot': Minimaler Rotationswinkel um den Normalvektor des `Template`-Bildes.
Typische Werte sind [*-180 ... 0*], Defaultwert ist *-180*.

'max_rot': Maximaler Rotationswinkel um den Normalvektor des `Template`-Bildes.
Typische Werte sind [*0 ... 180*], Defaultwert ist *180*.

'min_scale': Minimaler Skalierungsfaktor des `Template`-Bildes.
Typische Werte sind [*0.1 ... 1.0*], Defaultwert ist *0.5*.

'max_scale': Maximaler Skalierungsfaktor des `Template`-Bildes.
Typische Werte sind [*1.0 ... 3.5*], Defaultwert ist *1.4*.

Die Parameter *'min_rot'*, *'max_rot'*, *'min_scale'* und *'max_scale'* ermöglichen eine manuelle Selektion der für das Training durch Affintransformationen erzeugten Ansichten von `Template`. Eine Einschränkung ist insbesondere in Verbindung mit dem Parameter *'tilt'* zur Reduzierung der Trainingszeiten hilfreich. Es ist zu beachten, dass die Einstellung dieser Parameter direkten Einfluss auf die Ergebnisse von `find_uncalib_descriptor_model` hat. Deshalb sollten sie mit Bedacht verwendet werden. Falls die Rotationsspanne beispielsweise auf *'min_rot'* = -10 bis *'max_rot'* = 10 eingeschränkt wird, ist es in `find_uncalib_descriptor_model` nicht möglich, Ansichten von `Template` zu finden, die außerhalb dieses Bereichs liegen. Eine Einschränkung des Trainingsbereichs hat auch zur Folge, dass weniger Farne / geringere Farbtiefen nötig sind, um `Template` zu erkennen. Eine Ausweitung bedingt folglich mehr Bäume / Baumtiefen.

Bemerkungen

Es ist zu beachten, dass das Training der randomized ferns je nach Rechenleistung mehrere Sekunden bis zu wenigen Minuten dauern kann. Aus diesem Grund kann ein generierter Fern mittels `write_descriptor_model` abgespeichert und mittels `read_descriptor_model` wieder geladen werden.

Alle Parameter sowie die Koordinaten der letztendlichen Merkmalspunkte können über `get_descriptor_model_params` und `get_descriptor_model_points` abgerufen werden.

`create_uncalib_descriptor_model` legt den Detektor und den Deskriptor fest. Somit werden beide auch bei jedem nachfolgenden Aufruf von `find_uncalib_descriptor_model` verwendet. Der Referenzpunkt (Ursprung) des Modells ist der Schwerpunkt der ROI des Templates. Er kann über `set_descriptor_model_origin` geändert werden.

Parameter

▷ **Template** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.

▷ **DetectorType** (input_control) . . . . . string ⇝ *string*
Art des Detektors.
Defaultwert: 'lepetit'
Werteliste: `DetectorType` ∈ {'lepetit', 'harris', 'harris_binomial'}

▷ **DetectorParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der Detektor-Parameter.
Defaultwert: []
Werteliste: `DetectorParamName` ∈ {'alpha', 'check_neighbor', 'mask_size_grd', 'mask_size_smooth', 'min_check_neighbor_diff', 'min_score', 'radius', 'sigma_grad', 'sigma_smooth', 'subpix', 'threshold'}

▷ **DetectorParamValue** (input_control) . . . . . attribute.value-array ⇝ *integer* / real / string
Werte der Detektor-Parameter.
Defaultwert: []
Wertevorschläge: `DetectorParamValue` ∈ {0.08, 1, 1.2, 3, 15, 30, 1000, 'on', 'off', 'none', 'interpolation'}

▷ **DescriptorParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der Deskriptor-Parameter.
Defaultwert: []
Werteliste: `DescriptorParamName` ∈ {'depth', 'max_rot', 'max_scale', 'min_rot', 'min_scale', 'number_ferns', 'patch_size', 'tilt'}

▷ **DescriptorParamValue** (input_control) . . . . . attribute.value-array ⇝ *integer* / real / string
Werte der Deskriptor-Parameter.
Defaultwert: []
Wertevorschläge: `DescriptorParamValue` ∈ {0.5, 1.4, 11, 21, 30, -180, 180, 'on', 'off'}

▷ **Seed** (input_control) . . . . . integer ⇝ *integer*
Initialisierung des Zufallsgenerators.
Defaultwert: 42

▷ **ModelID** (output_control) . . . . . descriptor_model ⇝ *handle*
Der Handle des Deskriptormodells

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`points_lepetit`, `points_harris`, `reduce_domain`

Nachfolger

`get_descriptor_model_params`, `find_uncalib_descriptor_model`

Siehe auch

`get_descriptor_model_params`, `find_uncalib_descriptor_model`

Literatur

V. Lepetit and P. Fua: „Keypoint Recognition using Randomized Trees.„ IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 28, Nr. 9, pp. 1465-1479, 2006.

M. Ozuysal, P. Fua, and V. Lepetit: „Fast Keypoint Recognition in Ten Lines of Code.„ In Proceedings of Conference on Computer Vision and Pattern Recognition, 2007.

Modul

Matching

deserialize_descriptor_model (: : SerializedItemHandle : ModelID)

Deserialisieren eines Deskriptormodells.

deserialize_descriptor_model deserialisiert ein Deskriptormodell, welches mit serialize_descriptor_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Deskriptormodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Deskriptormodell mit dem Handle ModelID gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **ModelID** (output_control) ..... descriptor_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_descriptor_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_descriptor_model

Nachfolger

find_uncalib_descriptor_model, find_calib_descriptor_model

Siehe auch

create_calib_descriptor_model, create_uncalib_descriptor_model

Modul

Matching

find_calib_descriptor_model (Image : : ModelID, DetectorParamName, DetectorParamValue, DescriptorParamName, DescriptorParamValue, MinScore, NumMatches, CamParam, ScoreType : Pose, Score)

Suchen der besten Matches eines kalibrierten, Deskriptormodells in einem Suchbild und Bestimmung seiner 3D-Lage (Pose).

Der Operator find_calib_descriptor_model sucht den besten Match eines Deskriptormodells ModelID in einem Suchbild Image. Ein Deskriptormodell beinhaltet Beschreibungen von Merkmalspunkten und muss zuvor durch create_calib_descriptor_model erzeugt oder mit read_descriptor_model eingelesen worden sein.

Die zurückgelieferte Pose ist die Pose (im Kamera-Koordinatensystem) des gefundenen Musterobjekts im Suchbild Image. Da für Modellgenerierung und Suche unterschiedliche Kameras verwendet werden können, müssen die Kameraparameter der 'Suchkamera' in CamParam übergeben werden.

Der Suchprozess ist derselbe wie in find_uncalib_descriptor_model. Die Einstellungen für DetectorParamName, DetectorParamValue, DescriptorParamName, DescriptorParamValue, MinScore, NumMatches und ScoreType sind beim Operator find_uncalib_descriptor_model beschrieben. Wenn ein Objekt gefunden wird, werden die korrespondierenden Modell- und Suchpunkte intern an vector_to_pose übergeben, um die Pose des Objekts zu bestimmen. Wie beim Operator create_calib_descriptor_model beschrieben, werden die Modellpunkte davor in Weltkoordinaten der Modellebene umgerechnet.

Die gefundenen Punktkorrespondenzen einzelner Instanzen können mit Hilfe des Operators `get_descriptor_model_points` abgefragt werden.

Achtung

Der Definitionsbereich des Suchbildes sollte das gesamte gesuchte Objekt enthalten, da die Merkmalspunkte nur innerhalb des Definitionsbereiches des Suchbildes extrahiert werden. Das bedeutet, dass der resultierende `Score` entsprechend kleiner wird, wenn der Definitionsbereich nicht das gesamte gesuchte Objekt enthält. Des Weiteren ist zu beachten, dass Matches auch dann zurück gegeben werden, wenn der Referenzpunkt des Modells außerhalb des Definitionsbereiches des Suchbildes liegt. Die beiden hier geschilderten Auswirkungen des Definitionsbereiches des Suchbildes unterscheiden sich grundsätzlich von der Auswirkung des Definitionsbereiches beim formbasierten Matching, wo der Definitionsbereich den Suchraum für den Referenzpunkt des Modells darstellt.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Das Suchbild.
- ▷ **`ModelID`** (input_control) ..... descriptor_model ⇝ *handle*
  Der Handle des Deskriptormodells.
- ▷ **`DetectorParamName`** (input_control) ..... attribute.name-array ⇝ *string*
  Namen der Parameter des Detektors.
  Defaultwert: []
  Werteliste: `DetectorParamName` ∈ {'alpha', 'check_neighbor', 'mask_size_grd', 'mask_size_smooth', 'min_check_neighbor_diff', 'min_score', 'radius', 'sigma_grad', 'sigma_smooth', 'subpix', 'threshold'}
- ▷ **`DetectorParamValue`** (input_control) ..... attribute.value-array ⇝ *integer* / real / string
  Werte der Detektor-Parameter.
  Defaultwert: []
  Wertevorschläge: `DetectorParamValue` ∈ {0.08, 1, 1.2, 3, 15, 30, 1000, 'on', 'off'}
- ▷ **`DescriptorParamName`** (input_control) ..... attribute.name-array ⇝ *string*
  Namen der Deskriptor-Parameter.
  Defaultwert: []
  Werteliste: `DescriptorParamName` ∈ {'min_score_descr', 'guided_matching'}
- ▷ **`DescriptorParamValue`** (input_control) ..... attribute.value-array ⇝ *real* / integer / string
  Werte der Deskriptor-Parameter.
  Defaultwert: []
  Wertevorschläge: `DescriptorParamValue` ∈ {0.0, 0.001, 0.005, 0.01, 'on', 'off'}
- ▷ **`MinScore`** (input_control) ..... real(-array) ⇝ *real*
  Minimale Bewertung der zu findenden Instanzen der Modelle.
  Defaultwert: 0.2
  Wertevorschläge: `MinScore` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
- ▷ **`NumMatches`** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von gefundenen Instanzen.
  Defaultwert: 1
  Wertevorschläge: `NumMatches` ∈ {1, 2, 3, 4}
  Restriktion: NumMatches >= 1
- ▷ **`CamParam`** (input_control) ..... campar ⇝ *real* / integer / string
  Kamera Parameter (innere Orientierung) aus Kamerakalibrierung.
- ▷ **`ScoreType`** (input_control) ..... string(-array) ⇝ *string*
  Bewertungstyp berechnet in `Score`.
  Defaultwert: 'num_points'
  Werteliste: `ScoreType` ∈ {'num_points', 'inlier_ratio'}
- ▷ **`Pose`** (output_control) ..... pose(-array) ⇝ *real* / integer
  3D Pose des Objekts.
- ▷ **`Score`** (output_control) ..... number(-array) ⇝ *real* / integer
  Bewertung der gefundenen Instanzen gemäß ausgewählten ScoreType.

Beispiel

```
* Perform camera calibration (with standard calibration plate)
calibrate_cameras (CalibDataID, Error)
```

```
get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)
*  World coordinate system is defined by calibration plate in first image
get_calib_data (CalibDataID, 'calib_obj_pose',  [0,0], 'pose', \
                ObjInCameraPose)
*  Compensate thickness of plate
set_origin_pose(ObjInCameraPose, 0, 0, 0.0006, WorldPose)

* Create descriptor model:
create_calib_descriptor_model (Image,CamParam,WorldPose, \
                               'harris',[],[],[],[],42,ModelID)

get_descriptor_model_params (ModelID,DetectorType, \
                             DetectorParamName,DetectorParamValue, \
                             DescriptorParamName,DescriptorParamValue)
write_descriptor_model (ModelID,'simple_example.dsm')

read_descriptor_model ('simple_example.dsm',ModelID)
find_calib_descriptor_model (SearchImage,ModelID,[],[],[],[],0.2,1, \
                             CamParam,['num_points','inlier_ratio'], \
                             Pose,Score)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_calib_descriptor_model`, `read_descriptor_model`

Siehe auch

`vector_to_pose`, `points_lepetit`, `create_calib_descriptor_model`

Modul

Matching

find_uncalib_descriptor_model (Image : : ModelID, DetectorParamName, DetectorParamValue, DescriptorParamName, DescriptorParamValue, MinScore, NumMatches, ScoreType : HomMat2D, Score)

Suchen die besten Matches eines Deskriptormodells in einem Suchbild.

Der Operator `find_uncalib_descriptor_model` sucht den besten Match eines Deskriptormodells `ModelID` in einem Suchbild `Image`. Ein Deskriptormodell beinhaltet Beschreibungen von Merkmalspunkten und muss zuvor durch `create_uncalib_descriptor_model` bzw. `create_calib_descriptor_model` erzeugt oder mit `read_descriptor_model` eingelesen worden sein.

Ein Match wird akzeptiert, wenn er eine bessere Bewertung als `MinScore` hat. Dieses Kriterium basiert auf der 'inlier_ratio' Bewertung, die weiter unten erklärt wird. Hauptergebnis des Operators `find_uncalib_descriptor_model` ist die 3x3 Matrix `HomMat2D`, die eine projektive 2D Transformation von Modellpunkten zu Suchbildpunkten darstellt. Die Matrix ist durch ihre zeilenweise gegebenen 9 Elementen dargestellt. Tritt das gesuchte Objekt (Template) mehrfach in `Image` auf und erfüllt es das Kriterium `MinScore`, werden die resultierenden Homographien aneinandergehängt. Die Anzahl der letztendlich gefundenen Objekte lässt sich durch NumObjects = |`HomMat2D`| / 9 berechnen.

Der Suchprozess ist in drei Phasen unterteilt. **Als erstes** werden Merkmalspunkte aus dem Suchbild (aber nur innerhalb des Definitionsbereiches des Suchbildes) extrahiert. Dies erfolgt mit dem Punktoperator, der bei der Modellerstellung ausgewählten wurde. Während der Modellerstellung werden auch die Parameter des Punktoperators gesetzt. Sie können allerdings mittels `DetectorParamName` und `DetectorParamValue` auch zusätzlich

angepasst werden. Hierdurch können dieser Parameter insbesondere an Beleuchtungsschwankungen zwischen Modellerzeugung und Suchbild angepasst werden. Es wird jedoch empfohlen, die Werte des Modells zu verwenden (Übergabe eines leeren Tupels).

In der **zweiten Phase** der Suche werden Korrespondenzen zwischen Modellpunkten und extrahierten Punkten des Suchbildes erstellt. Die Parameter des Deskriptors werden in `DescriptorParamName` und `DescriptorParamValue` gesetzt:

'min_score_descr': Die minimale Klassifikatorbewertung eines Punktes, damit er als potentieller Match betrachtet wird. Diese Bewertungsfunktion hat einen Wertebereich von *0.0* bis *1.0*. Sinnvolle Werte liegen jedoch zwischen *0.0* und *0.1*. Es ist zu beachten, dass *'min_score_descr'* u.U. eine negative Auswirkung auf die Robustheit des Erkennungsprozesses hat. Zumindest dann, wenn wenige Merkmalspunkte auf dem Objekt gefunden werden können.
Typische Werte sind [*0.0* ... *0.1*], Defaultwert ist *0.0*.

'guided_matching': Erhöht die Genauigkeit der Objekterkennung. Das *'guided_matching'* kann allerdings in manchen Fällen auch die Rechenzeit um bis zu 10% erhöhen.
Mögliche Werte sind [*'on'*, *'off'*], Defaultwert ist *'on'*.

Der **letzte Schritt** ist die Schätzung der Homographie, die die Punktkorrespondenzen beschreibt. Die resultierende Homographie ist eine 2D Projektion, die eine Transformation von Modellpunkten zu Punkten in `Image` beschreibt. Hierfür werden Natural 3D Markers (N3Ms) verwendet, um die Punktkorrespondenz zu bestimmen (siehe Literatur).

Zusätzlich zu der geschätzten Homographie `HomMat2D` liefert der Operator eine oder mehrere `Score` Bewertungen für jede gefundene Objektinstanz, je nachdem, was der Benutzer in `ScoreType` eingegeben hat. Derzeit sind die folgenden Werte für `ScoreType` unterstützt:

'num_points': Anzahl der Punktkorrespondenzen je Instanz. Wurde eine Homographie mit weniger als 10 Korrespondenzen gefunden, so kann sie nicht als gut angenommen werden, da im Extremfall beliebige 4 Punkte eine mathematisch gültige Homographie beschreiben.

'inlier_ratio': Anteil von Punktkorrespondenzen an der Anzahl der Modellpunkte. Obwohl diese Bewertung einen Wertebereich von *0.0* bis *1.0* hat, ist es eher untypisch, dass sie einen Wert von *1.0* erreicht. Instanzen mit einem Bewertungwert kleiner als *0.1* sollten nicht berücksichtigt werden.

Falls im Suchbild mehrere Objektinstanzen gefunden werden, werden die resultierenden `Score` Bewertungen für die einzelnen Instanzen aneinandergehängt, sodass |`Score`| = NumObjects * |`ScoreType`|.

Die gefundenen Punktkorrespondenzen einzelner Instanzen können mit Hilfe des Operators `get_descriptor_model_points` abgefragt werden.

Achtung

Der Definitionsbereich des Suchbildes sollte das gesamte gesuchte Objekt enthalten, da die Merkmalspunkte nur innerhalb des Definitionsbereiches des Suchbildes extrahiert werden. Das bedeutet, dass der resultierende `Score` entsprechend kleiner wird, wenn der Definitionsbereich nicht das gesamte gesuchte Objekt enthält. Des Weiteren ist zu beachten, dass Matches auch dann zurück gegeben werden, wenn der Referenzpunkt des Modells außerhalb des Definitionsbereiches des Suchbildes liegt. Die beiden hier geschilderten Auswirkungen des Definitionsbereiches des Suchbildes unterscheiden sich grundsätzlich von der Auswirkung des Definitionsbereiches beim formbasierten Matching, wo der Definitionsbereich den Suchraum für den Referenzpunkt des Modells darstellt.

Parameter

▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object* : byte / uint2
Das Suchbild.

▷ **`ModelID`** (input_control) ..... descriptor_model ⇝ *handle*
Der Handle des Deskriptormodells.

▷ **`DetectorParamName`** (input_control) ..... attribute.name-array ⇝ *string*
Namen der Parameter des Detektors.
Defaultwert: []
Werteliste: `DetectorParamName` ∈ {'alpha', 'check_neighbor', 'mask_size_grd', 'mask_size_smooth', 'min_check_neighbor_diff', 'min_score', 'radius', 'sigma_grad', 'sigma_smooth', 'subpix', 'threshold'}

▷ **`DetectorParamValue`** (input_control) ..... attribute.value-array ⇝ *integer* / real / string
Werte der Detektor-Parameter.
Defaultwert: []
Wertevorschläge: `DetectorParamValue` ∈ {0.08, 1, 1.2, 3, 15, 30, 1000, 'on', 'off'}

▷ **DescriptorParamName** (input_control) ..... attribute.name-array ⇝ *string*
Namen der Deskriptor-Parameter.
Defaultwert: []
Werteliste: DescriptorParamName ∈ {'min_score_descr', 'guided_matching'}

▷ **DescriptorParamValue** (input_control) ..... attribute.value-array ⇝ *real* / integer / string
Werte der Deskriptor-Parameter.
Defaultwert: []
Wertevorschläge: DescriptorParamValue ∈ {0.0, 0.001, 0.005, 0.01, 'on', 'off'}

▷ **MinScore** (input_control) ..... real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen der Modelle.
Defaultwert: 0.2
Wertevorschläge: MinScore ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ MinScore ≤ 1

▷ **NumMatches** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl von gefundenen Instanzen.
Defaultwert: 1
Wertevorschläge: NumMatches ∈ {1, 2, 3, 4}
Restriktion: NumMatches >= 1

▷ **ScoreType** (input_control) ..... string(-array) ⇝ *string*
Bewertungstyp berechnet in Score.
Defaultwert: 'num_points'
Werteliste: ScoreType ∈ {'num_points', 'inlier_ratio'}

▷ **HomMat2D** (output_control) ..... hom_mat2d(-array) ⇝ *real*
Homographie zwischen Modell und gefundenem Objekt.

▷ **Score** (output_control) ..... number(-array) ⇝ *real* / integer
Bewertung der gefundenen Instanzen gemäß ausgewählten ScoreType.

Beispiel

```
create_uncalib_descriptor_model (ImageReduced,'harris',[],[], \
                                 [],[],42,ModelID)

get_descriptor_model_params (ModelID,DetectorType, \
                             DetectorParamName,DetectorParamValue, \
                             DescriptorParamName,DescriptorParamValue)
write_descriptor_model (ModelID,'simple_example.dsm')

read_descriptor_model ('simple_example.dsm',ModelID)
find_uncalib_descriptor_model (SearchImage,ModelID,[],[],[],[],0.2,1, \
                               ['num_points','inlier_ratio'],HomMat2D,Score)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_uncalib_descriptor_model, create_calib_descriptor_model, read_descriptor_model

Siehe auch

create_uncalib_descriptor_model, create_calib_descriptor_model, find_calib_descriptor_model, get_descriptor_model_points

Literatur

S. Hinterstoisser, S. Benhimane, and N. Navab: „N3M: Natural 3D Markers for Real-Time Object Detection and Pose Estimation.„ IEEE 11th International Conference on Computer Vision, 2007. pp. 1-7, ICCV 2007.

Modul

Matching

get_descriptor_model_origin (: : ModelID : Row, Column)

Zurückgeben des Ursprungs eines Deskriptormodells.

Der Operator get_descriptor_model_origin gibt den Ursprung (Referenzpunkt) des Deskriptormodells ModelID zurück. Der Ursprung wird relativ zum Schwerpunkt der Region des Bildes, das zur Erzeugung des Deskriptormodells mit create_uncalib_descriptor_model oder create_calib_descriptor_model verwendet wurde, angegeben. Ein Ursprung von (0,0) heißt also, dass der Schwerpunkt der Region des Modellbildes verwendet wird. Ein Ursprung von (-20,-40) heißt, dass der Ursprung links oberhalb des Schwerpunktes liegt.

Parameter

- ▷ **ModelID** (input_control) . . . . . descriptor_model ⇝ *handle*
  Handle zu einem Deskriptormodell.
- ▷ **Row** (output_control) . . . . . real ⇝ *real* / integer
  Lage des Ursprungs in Zeilenrichtung.
- ▷ **Column** (output_control) . . . . . real ⇝ *real* / integer
  Lage des Ursprungs in Spaltenrichtung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_uncalib_descriptor_model, read_descriptor_model

Siehe auch

set_descriptor_model_origin, create_uncalib_descriptor_model

Modul

Matching

get_descriptor_model_params (: : ModelID : DetectorType, DetectorParamName, DetectorParamValue, DescriptorParamName, DescriptorParamValue)

Ermitteln der Parameter eines Deskriptormodells.

Der Operator get_descriptor_model_params erlaubt es, die Parameter eines durch create_uncalib_descriptor_model oder create_calib_descriptor_model erzeugten Deskriptormodells zu bestimmen. Hierfür muss ein Handle zu einem Modell in ModelID übergeben werden. Der Operator liefert in DetectorType, DetectorParamName und DetectorParamValue den Typ, sowie die Parameter des Detektors zurück. Entsprechend werden in DescriptorType, DescriptorParamName und DescriptorParamValue die Einstellungen des Deskriptors zurückgeliefert.

Eine Anwendung dieses Operators ist die Ermittlung der Parameter eines Deskriptormodells das mit read_descriptor_model geladen wurde.

Parameter

- ▷ **ModelID** (input_control) . . . . . descriptor_model ⇝ *handle*
  Der Handle des Deskriptormodells.
- ▷ **DetectorType** (output_control) . . . . . string ⇝ *string*
  Art des Detektors.
- ▷ **DetectorParamName** (output_control) . . . . . attribute.name-array ⇝ *string*
  Namen der Detektor-Parameter.
- ▷ **DetectorParamValue** (output_control) . . . . . attribute.value-array ⇝ *integer* / real / string
  Werte der Detektor-Parameter.

▷ **DescriptorParamName** (output_control) . . . . . attribute.name-array ⇝ *string*
Namen der Deskriptor-Parameter.

▷ **DescriptorParamValue** (output_control) . . . . . attribute.value-array ⇝ *integer* / real / string
Werte der Deskriptor-Parameter.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_uncalib_descriptor_model`, `create_calib_descriptor_model`, `read_descriptor_model`

Siehe auch

`create_uncalib_descriptor_model`, `create_calib_descriptor_model`

Modul

Matching

get_descriptor_model_points (: : ModelID, Set, Subset : Row, Column)

Ermitteln der Koordinaten der Punkte eines Deskriptormodells.

Der Operator `get_descriptor_model_points` erlaubt es, die Punktkoordinaten eines Deskriptormodells oder des zuletzt bearbeiteten Suchbildes abzufragen. Hierfür muss ein Handle zu einem Modell in `ModelID` übergeben werden, das durch `create_calib_descriptor_model`, `create_uncalib_descriptor_model` oder `read_descriptor_model` erzeugt wurde.

Die im Modell gespeicherten Merkmalspunkte können immer abgefragt werden, indem man `Set` auf *'model'* und `Subset` auf *'all'* setzt. Wenn `find_uncalib_descriptor_model` oder `find_calib_descriptor_model` schon ausgeführt wurde, können die Merkmalspunkte des Suchbildes abgefragt werden, indem man `Set` auf *'search'* und `Subset` auf *'all'* setzt. Darüber hinaus können die gematchten (Korrespondenz-)Punkte für jede Objektinstanz abgefragt werden, indem man `Set` auf *'model'* oder *'search'* (für die Modell- oder Suchbild Merkmalspunkte) und `Subset` auf die Ergebnisnummer der Instanz setzt.

Die Bildkoordinaten der abgefragten Punkte werden in `Row` und `Column` zurückgeliefert.

Parameter

▷ **ModelID** (input_control) . . . . . descriptor_model ⇝ *handle*
Der Handle des Deskriptormodells.

▷ **Set** (input_control) . . . . . string ⇝ *string*
Satz von Merkmalspunkten.
Defaultwert: 'model'
Werteliste: `Set` ∈ {'model', 'search'}

▷ **Subset** (input_control) . . . . . integer ⇝ *integer* / string
Teilsatz von Merkmalspunkten.
Defaultwert: 'all'
Wertevorschläge: `Subset` ∈ {'all', 0, 1, 2}

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real* / integer
Zeilenkoordinaten der Merkmalspunkte.

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real* / integer
Spaltenkoordinaten der Merkmalspunkte.

Beispiel

```
create_uncalib_descriptor_model (Template,'harris',[],[],[],[],42, \
                                 ModelID)
```

```
* Model points can be queried from a model, even if just created
get_descriptor_model_points (ModelID,'model','all',ModelRow,ModelColumn)
find_uncalib_descriptor_model (Image,ModelID,[],[],[],[],0.2,1, \
                               'num_points',HomMat2D,Score)
* Search points can be queried only after a
* find_[un]calib_descriptor_model was executed
get_descriptor_model_points (ModelID,'search','all',SearchRow,SearchColumn)
* Additionally, correspondences for the results can be queried
NumObjects := |HomMat2D|/9
for I := 0 to NumObjects-1 by 1
    * Query corresponding points in the model
    get_descriptor_model_points (ModelID,'model',I, \
                                 CorrModelRow,CorrModelColumn)
    * Query corresponding points in the search image
    get_descriptor_model_points (ModelID,'search',I, \
                                 CorrSearchRow,CorrSearchColumn)
    * Those points are typically for visualizational purposes
    gen_cross_contour_xld (CrossModel,CorrModelRow,CorrModelColumn, \
                           6,0.78)
    gen_cross_contour_xld (CrossSearch,CorrSearchRow,CorrSearchColumn, \
                           6,0.78)

    * ....

endfor
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_uncalib_descriptor_model`, `create_calib_descriptor_model`, `find_uncalib_descriptor_model`, `find_calib_descriptor_model`, `read_descriptor_model`

Siehe auch

`create_uncalib_descriptor_model`, `create_calib_descriptor_model`

Modul

Matching

get_descriptor_model_results (: : ModelID, ObjectID, ResultNames : Results)

Abfrage von alphanumerischen Ergebnissen, die bei deskriptorbasiertem Matching angefallen sind.

Mit dem Operator `get_descriptor_model_results` können alphanumerische Ergebnisse des deskriptorbasierten Matchings abgefragt werden. Diese Ergebnisse sind im deskriptorbasierten Modell (`ModelID`) gespeichert. Der Parameter `ObjectID` gibt das Objekt an, dessen Ergebnisse zurückgegeben werden sollen. Der Index der Objekte beginnt mit 0. Auf die Objekte ist in der gleichen Reihenfolge zuzugreifen in der sie von `find_uncalib_descriptor_model` bzw. `find_calib_descriptor_model` gefunden werden. Ergebnisse können auch für alle Objekte gleichzeitig abgefragt werden, wenn der gewünschte Wert ein Einzelwert und kein Tupelwert ist (z.B. `ResultNames`=*'num_points'* oder `ResultNames`=*'inlier_ratio'*).

Welche Ergebnisse in `Results` zurückgeliefert werden hängt von den gewählten `ResultNames` ab:

'num_points': Anzahl der Objektpunkte, die mit Modellpunkten übereinstimmen.

'inlier_ratio': Verhältnis der Anzahl der übereinstimmenden Objektpunkte zu der Anzahl der Modellpunkte.

'homography': 3x3 projektive Transformationsmatrix, welche die Modellpunkte in Objektpunkte transformiert. Es ist zu beachten, dass die Punktabbildung von dem ausgewählten Modellursprung abhängt (siehe `set_descriptor_model_origin`).

'pose': Geschätzte Objektpose. Dieses Ergebnis kann nur nach einem Aufruf des Operators `find_calib_descriptor_model` abgefragt werden.

'point_classification': liefert hintereinander hängende Dreiergruppen, welche die Klassifikationsergebnisse der durch `find_uncalib_descriptor_model` oder `find_calib_descriptor_model` im Suchbild extrahierten Punkte wiedergeben. Jede Dreiergruppe enthält einen *Suchpunktindex* an der ersten, einen *Modellpunktindex* an der zweiten und die *Klassifikationsbewertung* and der dritten Stelle. Die Reihenfolge der Such- und Modellpunktindizes entspricht der Reihenfolge der Punkte, die vom Operator `get_descriptor_model_points` mit Parameter `Subset`=*'all'* und Parameter `Set`=*'model'* bzw. `Set`=*'search'* zurückgegeben werden. Wenn eine Nummer in `ObjectID` übergeben wird, wird nur die Klassifikation der zu diesem Objekt korrespondierenden Punkte zurückgeliefert. Wenn *'all'* in `ObjectID` übergeben wird, wird die Klassifikation für alle Punkte zurückgeliefert, auch wenn sie keinem Objekt zugeordnet wurden. Die Dreiergruppen werden nach ihrer Klassifikationsbewertung geordnet. Punkte, deren Bewertung niedriger als der Wert des Deskriptorparameters *'min_score_descr'* ist (siehe `find_calib_descriptor_model` oder `find_uncalib_descriptor_model`), werden verworfen und werden deswegen im Ergebnistupel nicht aufgelistet.

Parameter

▷ **`ModelID`** (input_control) ..... descriptor_model ⇝ *handle*
Handle zu einem Deskriptormodell.

▷ **`ObjectID`** (input_control) ..... integer ⇝ *integer* / string
Handle des Objekts, dessen Ergebnisse abgefragt werden sollen.
Defaultwert: 'all'
Wertevorschläge: `ObjectID` ∈ {'all', 0, 1, 2, 3}

▷ **`ResultNames`** (input_control) ..... attribute.name ⇝ *string*
Name des Ergebnisses, das abgefragt werden soll.
Defaultwert: 'num_points'
Werteliste: `ResultNames` ∈ {'num_points', 'inlier_ratio', 'homography', 'pose', 'point_classification'}

▷ **`Results`** (output_control) ..... attribute.value(-array) ⇝ *string* / integer / real
Zurückgelieferte Ergebnisse.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_uncalib_descriptor_model`, `find_calib_descriptor_model`

Siehe auch

`get_descriptor_model_origin`, `get_descriptor_model_params`, `get_descriptor_model_points`

Modul

Matching

`read_descriptor_model` (: : `FileName` : `ModelID`)

Einlesen eines Deskriptormodells von einer Datei.

`read_descriptor_model` liest ein Deskriptormodell, das mit `write_descriptor_model` geschrieben wurde, aus der Datei `FileName` ein. Die Dateiendung für das Deskriptormodell ist in HALCON 'dsm'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: .dsm

▷ **ModelID** (output_control) . . . . . descriptor_model ⇝ *handle*
Handle des Modells.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`find_uncalib_descriptor_model`, `find_calib_descriptor_model`

Siehe auch

`create_calib_descriptor_model`, `create_uncalib_descriptor_model`

Modul

Matching

serialize_descriptor_model (: : ModelID : SerializedItemHandle)

Serialisieren eines Deskriptormodells.

`serialize_descriptor_model` serialisiert die Daten eines Deskriptormodells (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_descriptor_model` in eine Datei geschrieben werden. Das Deskriptormodell wird in dem Handle `ModelID` übergeben. Das serialisierte Deskriptormodell wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_descriptor_model` wieder deserialisiert werden.

Parameter

▷ **ModelID** (input_control) . . . . . descriptor_model ⇝ *handle*
Handle zu dem zu speichernden Modell.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_descriptor_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`points_lepetit`, `create_uncalib_descriptor_model`, `create_calib_descriptor_model`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_descriptor_model`

Siehe auch

`create_uncalib_descriptor_model`, `create_calib_descriptor_model`

Modul

Matching

set_descriptor_model_origin (: : ModelID, Row, Column :)

Setzen des Ursprungs eines Deskriptormodells.

Der Operator set_descriptor_model_origin setzt den Ursprung (Referenzpunkt) des Deskriptormodells ModelID auf einen neuen Wert. Der Ursprung wird relativ zum Schwerpunkt der Region des Bildes, das zur Erzeugung des Deskriptormodells mit create_uncalib_descriptor_model oder create_calib_descriptor_model verwendet wurde, angegeben. Ein Ursprung von (0,0) heißt also, dass der Schwerpunkt der Region des Modellbildes verwendet wird. Ein Ursprung von (-20,-40) heißt, dass der Ursprung links oberhalb des Schwerpunktes liegt.

Parameter

- ▷ **ModelID** (input_control) ..... descriptor_model ⇝ *handle*
  Handle zu einem Deskriptormodell.
- ▷ **Row** (input_control) ..... real ⇝ *real* / integer
  Verschiebung des Ursprungs in Zeilenrichtung.
  Defaultwert: 0
- ▷ **Column** (input_control) ..... real ⇝ *real* / integer
  Verschiebung des Ursprungs in Spaltenrichtung.
  Defaultwert: 0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_uncalib_descriptor_model, create_calib_descriptor_model, read_descriptor_model

Siehe auch

get_descriptor_model_origin, create_uncalib_descriptor_model

Modul

Matching

write_descriptor_model (: : ModelID, FileName :)

Schreiben eines Deskriptormodells in eine Datei.

write_descriptor_model schreibt ein Deskriptormodell in die Datei FileName. Das Modell kann mit read_descriptor_model wieder eingelesen werden. Die Die Dateiendung für das Deskriptormodell ist in HALCON 'dsm'.

Parameter

▷ **ModelID** (input_control) . . . . . descriptor_model ⇝ *handle*
Handle zu dem zu speichernden Modell.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Der Pfad sowie Name des zu speichernden Modells.
Dateiendung: .dsm

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

points_lepetit, create_uncalib_descriptor_model, create_calib_descriptor_model

Siehe auch

create_uncalib_descriptor_model, create_calib_descriptor_model

Modul

Matching

17.3 Formbasiert

```
adapt_shape_model_high_noise ( ImageReduced : : ModelID,
    GenParam : ResultDict )
```

Anpassen der Parameter eines Formmodells an Bilder mit starkem Rauschen.

adapt_shape_model_high_noise passt die Parameter eines Formmodells ModelID an Bilder mit starkem Rauschen an und liefert sie in ResultDict zurück. Basierend auf einem beispielhaften Suchbild ImageReduced werden die optimalen Werte für die Parameter eines Formmodells geschätzt und im Modell angepasst. Die Domäne von ImageReduced sollte das Modell und umgebenden Hintergrund beinhalten, so dass für die Suchbilder typisches Rauschen zur Schätzung der Parameter berücksichtigt werden kann. Mit GenParam werden die Parameter, die geschätzt werden sollen, gesteuert. Zur Zeit unterstützt der Operator das Schätzen der untersten Pyramidenstufe, die bei der Suche verwendet wird. Entsprechend soll der Wert *'all'* oder *'lowest_level'* in GenParam übergeben werden. Es ist zu beachten, dass der Operator das Formmodell ModelID verändert und zwar so, dass nachfolgende Aufrufe von z.B. find_shape_model die geschätzte untere Pyramidenstufe verwenden. Über diese geschätzte untere Pyramidenstufe kann man sich in nachfolgenden Suchaufrufen hinwegsetzten, wie in den verschieden Suchoperator wie find_shape_model dokumentiert ist.

Parameter

▷ **ImageReduced** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
Beispielhaftes Suchbild mit reduzierter Domäne.

▷ **ModelID** (input_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

▷ **GenParam** (input_control) . . . . . string-array ⇝ *string*
Die zu schätzenden Parameter.
Defaultwert: 'all'
Werteliste: GenParam ∈ {'lowest_level', 'all'}

▷ **ResultDict** (output_control) . . . . . dict ⇝ *handle*
Dictionary mit den geschätzten Parameterwerten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert adapt_shape_model_high_noise den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_shape_model`, `create_scaled_shape_model`, `create_aniso_shape_model`

Nachfolger

`find_shape_model`, `find_scaled_shape_model`, `find_aniso_shape_model`, `find_shape_models`, `find_scaled_shape_models`, `find_aniso_shape_models`

Modul

Matching

clear_shape_model (: : ModelID :)

Freigabe des Speichers eines Formmodells.

`clear_shape_model` gibt den Speicher eines Formmodells, das mit `create_shape_model`, `create_scaled_shape_model` oder `create_aniso_shape_model` angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `ModelID` ist nach dem Aufruf ungültig.

Parameter

▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
Handle des Modells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `clear_shape_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_shape_model`, `create_scaled_shape_model`, `create_aniso_shape_model`, `read_shape_model`, `write_shape_model`

Modul

Matching

create_aniso_shape_model (Template : : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleRMin, ScaleRMax, ScaleRStep, ScaleCMin, ScaleCMax, ScaleCStep, Optimization, Metric, Contrast, MinContrast : ModelID)

Vorbereiten eines anisotrop skalierten Formmodells für das Matching.

create_aniso_shape_model bereitet ein Muster, das als Bild Template übergeben wird, als anisotrop skaliertes Formmodell für das Matching vor. Die ROI des Modells wird als der Definitionsbereich von Template übergeben.

Der Ausgabeparameter ModelID ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von find_aniso_shape_model verwendet wird. Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt der Region des Modellbildes Template verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit set_shape_model_origin gesetzt werden. Das Modell wird auf mehreren Pyramidenebenen generiert und im Speicher abgelegt. Falls eine vollständige Vorabgenerierung des Modells gewählt wird (siehe unten), wird das Modell in mehreren Rotationen und anisotropen Skalierungen (unabhängigen Skalierungen in Zeilen- und Spaltenrichtung) auf jeder Ebene generiert. Das Modell kann mit set_shape_model_clutter um Störparameter erweitert werden.

Eingabeparameter im Detail

NumLevels: Die Anzahl der Pyramidenebenen wird mit NumLevels festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von NumLevels ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens vier). Dies kann anhand der Ausgabe von inspect_shape_model überprüft werden. Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entstehen würde, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert create_aniso_shape_model eine Fehlermeldung zurück.

Falls für NumLevels *'auto'* (oder *0* für Rückwärtskompatibilität) übergeben wird, wählt create_aniso_shape_model die Anzahl der Pyramidenstufen automatisch. Die automatisch gewählte Anzahl von Pyramidenstufen kann mit get_shape_model_params abgefragt werden. In seltenen Fällen kann es vorkommen, dass create_aniso_shape_model die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für MinScore oder Greediness in find_aniso_shape_model selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in find_aniso_shape_model kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen mit Hilfe der Ausgabe von inspect_shape_model gewählt werden.

AngleStart, AngleExtent und AngleStep: Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Modells im Bild fest. Das Modell kann also mit find_aniso_shape_model nur in diesem Winkelbereich gefunden werden. Der Parameter AngleStep gibt die Schrittweite der Winkel in dem gewählten Winkelbereich an. Falls bei find_aniso_shape_model keine Subpixelgenauigkeit spezifiziert wird, gibt AngleStep also die erreichbare Winkelgenauigkeit an. AngleStep sollte aufgrund der Größe des Objektes gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte AngleStep für kleinere Modelle größer gewählt werden. Falls AngleExtent kein ganzzahliges Vielfaches von AngleStep ist, wird AngleStep entsprechend angepasst. Um sicherzustellen, dass find_aniso_shape_model für Modellinstanzen ohne Rotation Winkel von exakt 0.0 zurückgibt, wird der Winkelbereich der möglichen Rotationen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den AngleStart plus n mal AngleStep genau Null ergibt, wird AngleStart um höchstens AngleStep verringert, und AngleExtent wird um AngleStep vergrößert.

ScaleRMin, ScaleRMax, ScaleCMin, ScaleCMax, ScaleRStep und ScaleCStep: Die Parameter ScaleRMin, ScaleRMax, ScaleCMin und ScaleCMax legen den Bereich der möglichen anisotropen Skalierung des Modells in Zeilen- und Spaltenrichtung im Bild fest. Eine Skalierung von 1 in beiden Skalierungsfaktoren entspricht der Originalgröße des Modells. Die Parameter ScaleRStep und ScaleCStep geben die Schrittweite der Skalierungen an. Falls bei find_aniso_shape_model keine Subpixelgenauigkeit spezifiziert wird, geben ScaleRStep und ScaleCStep also die erreichbare Genauigkeit der Skalierungen an. Wie AngleStep sollten ScaleRStep und ScaleRStep aufgrund der Größe des Objektes

gewählt werden. Falls der jeweilige Skalierungsbereich kein ganzzahliges Vielfaches von ScaleRStep und ScaleCStep ist, werden ScaleRStep und ScaleCStep entsprechend angepasst. Um sicherzustellen, dass find_aniso_shape_model für Modellinstanzen ohne Skalierung Skalierungen von exakt 1.0 zurückgibt, wird der Bereich der möglichen Skalierungen folgendermaßen angepasst: Wenn keine positiven ganzzahligen Werte n und m existieren, für die ScaleRMin plus n mal ScaleRStep sowie ScaleCMin plus m mal ScaleCStep genau 1.0 ergeben, werden ScaleRMin und ScaleCMin um höchstens ScaleRStep bzw. ScaleCStep verringert, und ScaleRMax und ScaleCMax werden so erhöht, dass der jeweilige Skalierungsbereich um ScaleRStep bzw. ScaleCStep vergrößert wird.

Es ist zu beachten, dass die Transformationen intern so behandelt werden, dass erst die Skalierungen und dann die Rotation angewendet werden. Daher sollte das Modell im Normalfall so ausgerichtet werden, dass es im Modellbild horizontal oder vertikal erscheint.

Optimization: Bei besonders großen Modellen kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters Optimization auf einen Wert ungleich *'none'* zu setzen. Falls Optimization = *'none'*, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter Optimization reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei find_aniso_shape_model notwendig werden, den Parameter Greediness auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen.

Falls für Optimization *'auto'* übergeben wird, wählt create_aniso_shape_model die Reduktion der Punkte automatisch.

Metric: Der Parameter Metric legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird.

Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist.

Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_aniso_shape_model erhöht sich in diesem Fall geringfügig.

Falls Metric = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Da sich in diesem Fall die Laufzeit von find_aniso_shape_model aber wesentlich erhöht, ist es in solchen Fällen meist sinnvoller, mehrere Modelle mit create_aniso_shape_model zu erzeugen und mit find_aniso_shape_models zu suchen.

Die obigen drei Metriken können nur auf einkanalige Bilder angewendet werden. Falls im Modellbild oder in den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes verwendet.

Falls Metric = *'ignore_color_polarity'*, wird das Modell auch dann gefunden, falls sich die Farbkontraste lokal ändern können. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün. Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von find_aniso_shape_model erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für Metric = *'ignore_color_polarity'* die Anzahl der Kanäle bei der Modellgenerierung mit create_aniso_shape_model und bei der Suche mit find_aniso_shape_model verschieden sein können. Dies erlaubt z.B. die Modellgenerierung aus einem einkanaligen synthetisch generierten Bild. Es ist auch zu beachten, dass die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein.

Contrast: Der Parameter Contrast legt fest, welchen Grauwertkontrast die Punkte des Modells besitzen müssen. Der Kontrast ist ein Maß für die lokalen Grauwertdifferenzen zwischen dem Objekt und dem Hintergrund und zwischen verschiedenen Teilen des Objektes. Contrast sollte so gewählt werden, dass nur die signifikanten Merkmale des Musters für das Modell verwendet werden. Contrast kann auch ein Tupel mit zwei Werten enthalten. In diesem Fall wird bei der Segmentierung des Modells ein Verfahren, das ähnlich zum Hysterese-Schwellenwert-Verfahren in edges_image ist, eingesetzt. Dabei ist der erste Wert des Tupels der untere Schwellenwert und der zweite Wert der obere Schwellenwert. Für weitere Informationen über das Hysterese-Schwellenwert-Verfahren siehe hysteresis_threshold. Optional kann in Contrast

auch ein dritter Wert als letztes Element des Tupels übergeben werden. Dieser Wert dient zur Selektion von signifikanten Komponenten des Modells basierend auf der Größe der Komponenten, d.h. Komponenten, deren Anzahl von Punkten kleiner ist als diese Mindestgröße, werden unterdrückt. Da sich die Mindestgröße auf die Größe der Komponenten bezieht kann es auch vorkommen, dass die daraus abgeleiteten Modellkonturen trotzdem kleiner als die angegebene Mindestgröße sind. Dieser Schwellenwert wird pro Pyramidenstufe halbiert. Falls eine Unterdrückung von kleinen Modellkomponenten gewünscht ist, aber keine Hysterese-Schwellenwert-Operation, müssen dennoch drei Werte in `Contrast` übergeben werden, von denen die ersten zwei auf denselben Wert gesetzt werden. Die Wirkung von `Contrast` kann auch vorab mit `inspect_shape_model` überprüft werden.

Wird `Contrast` auf *'auto'* gesetzt, dann bestimmt `create_aniso_shape_model` die beschriebenen drei Werte automatisch. Es besteht die Möglichkeit, nur den Kontrast (*'auto_contrast'*), die Hysterese-Schwellenwerte (*'auto_contrast_hyst'*) oder die Mindestgröße (*'auto_min_size'*) automatisch bestimmen zu lassen. Die übrigen, nicht automatisch zu bestimmenden Größen, können dann zusätzlich als Tupel übergeben werden. Dabei sind auch Kombinationen zulässig: Wird z.B. *['auto_contrast','auto_min_size']* übergeben, wird sowohl der Kontrast als auch die Mindestgröße bestimmt. Bei *['auto_min_size',20,30]* wird die Mindestgröße automatisch ermittelt, während die Hysterese-Schwellen auf *20* und *30* gesetzt werden, etc. In bestimmten Fällen kann es vorkommen, dass die automatische Bestimmung der Kontrast-Schwellenwerte nicht zum gewünschten Ergebnis führt. So ist eine manuelle Parameterwahl zu bevorzugen, wenn anwendungsbedingt bestimmte Modellkomponenten ausgeschlossen oder ins Modell integriert werden sollen oder wenn das Objekt mehrere unterschiedliche Kontraststufen umfasst. Daher ist es sinnvoll, die Kontrastwerte vor der Modellgenerierung mit `determine_shape_model_params` automatisch zu ermitteln und das Ergebnis mit `inspect_shape_model` zu überprüfen. Es ist zu beachten, dass `MinContrast` die Bestimmung der Kontrast-Schwellenwerte beeinflusst, und daher auch die Schätzung der Mindestgröße.

MinContrast: Mit `MinContrast` wird festgelegt, welchen Grauwertkontrast das Modell später bei der Erkennung mit `find_aniso_shape_model` im Bild mindestens besitzen muss. Mit anderen Worten stellt dieser Parameter somit eine Abgrenzung des Musters von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte `MinContrast` auf 10 gesetzt werden. Falls mehrkanalige Bilder für das Modell und für die Suchbilder verwendet werden, und falls der Parameter `Metric` auf *'ignore_color_polarity'* gesetzt wird (siehe oben), muss das Rauschen in einem Kanal noch mit der Wurzel der Anzahl der Kanäle multipliziert werden, um `MinContrast` zu bestimmen. Falls die Grauwerte in einem Kanal z.B. in einem Bereich von 10 Graustufen schwanken, sollte `MinContrast` bei einem dreikanaligen Bild auf 17 gesetzt werden. Offensichtlich muss `MinContrast` kleiner als `Contrast` sein. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss `MinContrast` entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte `MinContrast` etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells zu gewährleisten.

Wird `MinContrast` auf *'auto'* gesetzt, wählt `create_aniso_shape_model` den Mindestkontrast basierend auf dem Rauschen im Modellbild automatisch. Eine automatische Bestimmung ergibt demzufolge nur Sinn, wenn das bei der Erkennung zu erwartende Bildrauschen dem Rauschen im Modellbild entspricht. Außerdem kann es in manchen Fällen sinnvoll sein, den automatisch ermittelten Wert zu erhöhen, um eine höhere Robustheit gegenüber Verdeckungen zu erzielen (siehe oben). Der automatisch bestimmte Mindestkontrast kann mit `get_shape_model_params` abgefragt werden.

Vollständige Vorabgenerierung des Modells

Optional kann in `Optimization` ein zweiter Wert übergeben werden. Dieser Wert legt fest, ob das Modell vollständig vorab generiert wird oder nicht. Der zweite Wert von `Optimization` ist hierzu auf *'pregeneration'* oder *'no_pregeneration'* zu setzen. Falls der zweite Wert nicht angegeben wird, wird der mit `set_system ('pregenerate_shape_models',...)` eingestellte Modus verwendet. Mit der Standardeinstellung (*'pregenerate_shape_models'* = *'false'*) wird das Modell nicht vollständig vorab generiert. Die vollständige Vorabgenerierung des Modells führt im Normalfall zu leicht geringeren Laufzeiten in `find_aniso_shape_model`, da das Modell nicht zur Laufzeit transformiert werden muss. Allerdings sind in diesem Fall der Speicherbedarf und die Zeit, die zur Modellerzeugung benötigt wird, wesentlich höher. Zu beachten ist auch, dass in den zwei Modi nicht dieselben Ergebnisse erwartet werden können, da intern bei der Transformation zur Laufzeit notwendigerweise andere Daten für die transformierten Modelle entstehen, als bei der Vorabgenerierung. Falls das Modell nicht vorab generiert wird, werden z.B. in `find_aniso_shape_model` typischerweise leicht niedrigere Bewertungen (Scores) zurückgeliefert, so dass es unter Umständen notwendig sein kann, einen niedrigeren Wert für `MinScore` als bei vollständiger Vorabgenerierung zu verwenden. Außerdem können sich die durch Interpolation

gewonnenen Subpixelpositionen geringfügig unterscheiden. Falls höchste Genauigkeit gewünscht wird, sollte die Lage des Modells mittels Ausgleichsrechnung (Least-Squares Adjustment) bestimmt werden.

Falls eine vollständige Vorabgenerierung des Modells gewählt wird, wird das Modell in dem gewählten Winkelbereich und Skalierungsbereich vorab generiert und im Speicher abgelegt. Der Speicherbedarf zur Speicherung des Modells ist also proportional zur Anzahl der Winkelschritte, zur Anzahl der Skalierungsschritte und zur Anzahl der Punkte im Modell. Wenn also `AngleStep`, `ScaleRStep` oder `ScaleCStep` zu klein bzw. `AngleExtent` oder der Skalierungsbereich zu groß gewählt werden, kann es vorkommen, dass das Modell nicht mehr in den (virtuellen) Speicher passt. In diesem Fall muss entweder `AngleStep`, `ScaleRStep` oder `ScaleCStep` größer oder `AngleExtent` oder der Skalierungsbereich kleiner gewählt werden. In jedem Fall ist es aus Laufzeitgründen vorteilhaft, wenn das Modell komplett in den Hauptspeicher passt und somit ein Paging durch das Betriebssystem vermieden werden kann. Da die Möglichkeit zur subpixelgenauen Winkelbestimmung in `find_aniso_shape_model` gegeben ist, kann bei Modellen mit einem Durchmesser von ca. 200 Pixeln `AngleStep` $\geq 1^\circ$ und `ScaleRStep`, `ScaleCStep` ≥ 0.02 gewählt werden.

Falls `AngleStep` $= 'auto'$ oder `ScaleRStep`, `ScaleCStep` $= 'auto'$ (oder jeweils *0* für Rückwärtskompatibilität) übergeben wird, wählt `create_aniso_shape_model` automatisch eine Schrittweite für die Winkel bzw. Skalierungen basierend auf der Größe des Modells aus. Die automatisch gewählten Schrittweiten können mit `get_shape_model_params` abgefragt werden.

Falls eine vollständige Vorabgenerierung des Modells nicht gewählt wird, wird das Modell nur in einer Referenzlage auf jeder Ebene erzeugt. In diesem Fall muss das Modell zur Laufzeit in `find_aniso_shape_model` auf die verschiedenen Winkel und Skalierungen transformiert werden. Dadurch kann es vorkommen, dass das Auffinden des Modells etwas mehr Zeit benötigt.

Es ist zu beachten, dass vorabgenerierte Modelle für eine spezifische Bildgröße bestimmt sind. Aus Laufzeitgründen ist die Verwendung von Bildern mit verschiedenen Größen während der Suche mit dem gleichen Modell nicht parallel möglich. In diesem Fall müssen Kopien des gleichen Modells verwendet werden. Anderenfalls kann das Programm abstürzen.

Parameter

▷ **`Template`** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.

▷ **`NumLevels`** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: `NumLevels` ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **`AngleStart`** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Musters.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **`AngleExtent`** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **`AngleStep`** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: `AngleStep` ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep >= 0 && AngleStep <= pi / 16

▷ **`ScaleRMin`** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Zeilenrichtung.
Defaultwert: 0.9
Wertevorschläge: `ScaleRMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **`ScaleRMax`** (input_control) . . . . . number ⇝ *real*
Größte auftretende Skalierung des Musters in Zeilenrichtung.
Defaultwert: 1.1
Wertevorschläge: `ScaleRMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleRMax >= ScaleRMin

▷ **ScaleRStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleRStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleRStep >= 0

▷ **ScaleCMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 0.9
Wertevorschläge: ScaleCMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number ⇝ *real*
Größte auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 1.1
Wertevorschläge: ScaleCMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleCMax >= ScaleCMin

▷ **ScaleCStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleCStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleCStep >= 0

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung und optional Methode der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: Optimization ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high', 'pregeneration', 'no_pregeneration'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **Contrast** (input_control) . . . . . number(-array) ⇝ *integer* / string
Schwellenwert bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Musterbild und optional Mindestgröße der Objektteile.
Defaultwert: 'auto'
Wertevorschläge: Contrast ∈ {'auto', 'auto_contrast', 'auto_contrast_hyst', 'auto_min_size', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer* / string
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: MinContrast ∈ {'auto', 1, 2, 3, 5, 7, 10, 20, 30, 40}
Restriktion: MinContrast < Contrast

▷ **ModelID** (output_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_aniso_shape_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn die Parameter NumLevels und Contrast so gewählt worden sind, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

draw_region, reduce_domain, threshold

Nachfolger

find_aniso_shape_model, find_aniso_shape_models, get_shape_model_params, clear_shape_model, write_shape_model, set_shape_model_origin, set_shape_model_clutter

Alternativen

create_shape_model, create_scaled_shape_model

Siehe auch

set_system, get_system

Modul

Matching

create_aniso_shape_model_xld (Contours : : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleRMin, ScaleRMax, ScaleRStep, ScaleCMin, ScaleCMax, ScaleCStep, Optimization, Metric, MinContrast : ModelID)

Vorbereiten eines anisotrop skalierten Formmodells für das Matching aus XLD-Konturen.

create_aniso_shape_model_xld erzeugt ein anisotrop skaliertes Formmodell für das Matching aus den XLD Konturen Contours. Die XLD Konturen repräsentieren dabei die Grauwertkanten des zu suchenden Objekts im Bild. Im Gegensatz zu dem Operator create_aniso_shape_model, der ein Formmodell aus einem Bild erzeugt, erzeugt der Operator create_aniso_shape_model_xld das Formmodell aus XLD Konturen, d.h., ohne die Verwendung eines Bildes.

Der Ausgabeparameter ModelID ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von find_aniso_shape_model verwendet wird. Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt des kleinsten umschließenden Rechtecks der Konturen Contours, das parallel zu den Koordinatenachsen liegt, verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit set_shape_model_origin gesetzt werden. Das Modell wird für mehrere Pyramidenebenen generiert und im Speicher abgelegt. Falls eine vollständige Vorabgenerierung des Modells gewählt wird (siehe unten), wird das Modell in mehreren Rotationen und anisotropen Skalierungen (unabhängigen Skalierungen in Zeilen- und Spaltenrichtung) auf jeder Ebene generiert. Das Modell kann mit set_shape_model_clutter um Störparameter erweitert werden.

Eingabeparameter im Detail

NumLevels: Die Anzahl der Pyramidenebenen wird mit NumLevels festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von NumLevels ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens vier). Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entstehen würde, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert create_aniso_shape_model_xld eine Fehlermeldung zurück.

Falls für NumLevels *'auto'* übergeben wird, wählt create_aniso_shape_model_xld die Anzahl der Pyramidenstufen automatisch. Die gewählte Anzahl von Pyramidenstufen kann mit get_shape_model_params abgefragt werden. In seltenen Fällen kann es vorkommen, dass create_aniso_shape_model_xld die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für MinScore oder Greediness in find_aniso_shape_model selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in find_aniso_shape_model kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen manuell gewählt werden.

AngleStart, AngleExtent und AngleStep: Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Objektes im Bild fest. Das Objekt kann also mit find_aniso_shape_model nur in diesem Winkelbereich gefunden werden. Der Parameter AngleStep gibt die Schrittweite der Winkel in dem gewählten Winkelbereich an. Falls bei

find_aniso_shape_model keine Subpixelgenauigkeit spezifiziert wird, gibt AngleStep also die erreichbare Winkelgenauigkeit an. AngleStep sollte aufgrund der Größe des Objektes gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte AngleStep für kleinere Modelle größer gewählt werden. Falls AngleExtent kein ganzzahliges Vielfaches von AngleStep ist, wird AngleStep entsprechend angepasst. Um sicherzustellen, dass find_aniso_shape_model für Modellinstanzen ohne Rotation Winkel von exakt 0.0 zurückgibt, wird der Winkelbereich der möglichen Rotationen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den AngleStart plus n mal AngleStep genau Null ergibt, wird AngleStart um höchstens AngleStep verringert, und AngleExtent wird um AngleStep vergrößert.

ScaleRMin, ScaleRMax, ScaleCMin, ScaleCMax, ScaleRStep und ScaleCStep: Die Parameter ScaleRMin, ScaleRMax, ScaleCMin und ScaleCMax legen den Bereich der möglichen anisotropen Skalierung des Objektes in Zeilen- und Spaltenrichtung im Bild fest. Eine Skalierung von 1 in beiden Skalierungsfaktoren entspricht der Originalgröße des Modells. Die Parameter ScaleRStep und ScaleCStep geben die Schrittweite der Skalierungen an. Falls bei find_aniso_shape_model keine Subpixelgenauigkeit spezifiziert wird, geben ScaleRStep und ScaleCStep also die erreichbare Genauigkeit der Skalierungen an. Wie AngleStep sollten ScaleRStep und ScaleRStep aufgrund der Größe des Objektes gewählt werden. Falls der jeweilige Skalierungsbereich kein ganzzahliges Vielfaches von ScaleRStep und ScaleCStep ist, werden ScaleRStep und ScaleCStep entsprechend angepasst. Um sicherzustellen, dass find_aniso_shape_model für Modellinstanzen ohne Skalierung Skalierungen von exakt 1.0 zurückgibt, wird der Bereich der möglichen Skalierungen folgendermaßen angepasst: Wenn keine positiven ganzzahligen Werte n und m existieren, für die ScaleRMin plus n mal ScaleRStep sowie ScaleCMin plus m mal ScaleCStep genau 1.0 ergeben, werden ScaleRMin und ScaleCMin um höchstens ScaleRStep bzw. ScaleCStep verringert, und ScaleRMax und ScaleCMax werden so erhöht, dass der jeweilige Skalierungsbereich um ScaleRStep bzw. ScaleCStep vergrößert wird.

Es ist zu beachten, dass die Transformationen intern so behandelt werden, dass erst die Skalierungen und dann die Rotation angewendet werden. Daher sollte das Modell im Normalfall so ausgerichtet werden, dass es im Modellbild horizontal oder vertikal erscheint.

Optimization: Bei besonders großen Modellen kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters Optimization auf einen Wert ungleich *'none'* zu setzen. Falls Optimization = *'none'*, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter Optimization reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei find_aniso_shape_model notwendig werden, den Parameter Greediness auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen.

Falls für Optimization *'auto'* übergeben wird, wählt create_aniso_shape_model_xld die Reduktion der Punkte automatisch.

Metric: Der Parameter Metric legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird.

Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist.

Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_aniso_shape_model erhöht sich in diesem Fall geringfügig.

Es ist zu beachten, dass die beiden Werte *'use_polarity'* und *'ignore_global_polarity'* für Metric nur dann verwendet werden können, wenn bei allen Konturen, die in Contours übergeben wurden, das Attribut *'edge_direction'* gesetzt ist, da es die Polarität der Kanten festlegt. Für weitere Informationen zu Konturattributen wie *'edge_direction'* siehe get_contour_attrib_xld. Ansonsten besteht die Möglichkeit, die beiden Werte für Metric mit dem Operator set_shape_model_metric zu setzen, der die Polarität der Kanten aus einem Bild bestimmt.

Falls Metric = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Da sich in diesem Fall die Laufzeit von find_aniso_shape_model aber wesentlich erhöht, ist es in solchen Fällen meist sinnvoller, mehrere Modelle mit create_aniso_shape_model_xld zu erzeugen und mit find_aniso_shape_models zu suchen.

Die obigen drei Metriken können nur auf einkanalige Bilder angewendet werden. Falls im Modellbild oder in den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes

verwendet. Falls Metric = *'ignore_color_polarity'*, wird das Modell auch dann gefunden, falls sich die Farbkontraste lokal ändern können. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün. Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von find_aniso_shape_model erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für Metric = *'ignore_color_polarity'* die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein. Es ist auch zu beachten, dass die beiden ersten Werte für Metric (*'use_polarity'* und *'ignore_global_polarity'*) nur dann verwendet werden können, wenn bei allen Konturen, die in Contours übergeben wurden, das Attribut *'edge_direction'* gesetzt ist, da es die Polarität der Kanten festlegt. Für weitere Informationen zu Konturattributen wie *'edge_direction'* siehe get_contour_attrib_xld. Ansonsten besteht die Möglichkeit, die beiden Werte für Metric mit dem Operator set_shape_model_metric zu setzen, der die Polarität der Kanten aus einem Bild bestimmt.

MinContrast: Mit MinContrast wird festgelegt, welchen Grauwertkontrast die Objektkanten später bei der Erkennung mit find_aniso_shape_model im Bild mindestens besitzen müssen. Mit anderen Worten stellt dieser Parameter somit eine Abgrenzung des Objektes von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte MinContrast auf 10 gesetzt werden. Falls mehrkanalige Bilder für das Modell und für die Suchbilder verwendet werden, und falls der Parameter Metric auf *'ignore_color_polarity'* gesetzt wird (siehe oben), muss das Rauschen in einem Kanal noch mit der Wurzel der Anzahl der Kanäle multipliziert werden, um MinContrast zu bestimmen. Falls die Grauwerte in einem Kanal z.B. in einem Bereich von 10 Graustufen schwanken, sollte MinContrast bei einem dreikanaligen Bild auf 17 gesetzt werden. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss MinContrast entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte MinContrast etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells zu gewährleisten.

Vollständige Vorabgenerierung des Modells

Optional kann in Optimization ein zweiter Wert übergeben werden. Dieser Wert legt fest, ob das Modell vollständig vorab generiert wird oder nicht. Der zweite Wert von Optimization ist hierzu auf *'pregeneration'* oder *'no_pregeneration'* zu setzen. Falls der zweite Wert nicht angegeben wird, wird der mit set_system ('pregenerate_shape_models',...) eingestellte Modus verwendet. Mit der Standardeinstellung (*'pregenerate_shape_models'* = *'false'*) wird das Modell nicht vollständig vorab generiert. Die vollständige Vorabgenerierung des Modells führt im Normalfall zu leicht geringeren Laufzeiten in find_aniso_shape_model, da das Modell nicht zur Laufzeit transformiert werden muss. Allerdings sind in diesem Fall der Speicherbedarf und die Zeit, die zur Modellerzeugung benötigt wird, wesentlich höher. Zu beachten ist auch, dass in den zwei Modi nicht dieselben Ergebnisse erwartet werden können, da intern bei der Transformation zur Laufzeit notwendigerweise andere Daten für die transformierten Modelle entstehen, als bei der Vorabgenerierung. Falls das Modell nicht vorab generiert wird, werden z.B. in find_aniso_shape_model typischerweise leicht niedrigere Bewertungen (Scores) zurückgeliefert, so dass es unter Umständen notwendig sein kann, einen niedrigeren Wert für MinScore als bei vollständiger Vorabgenerierung zu verwenden. Außerdem können sich die durch Interpolation gewonnenen Subpixelpositionen geringfügig unterscheiden. Falls höchste Genauigkeit gewünscht wird, sollte die Lage des Modells mittels Ausgleichsrechnung (Least-Squares Adjustment) bestimmt werden.

Falls eine vollständige Vorabgenerierung des Modells gewählt wird, wird das Modell in dem gewählten Winkelbereich und Skalierungsbereich vorab generiert und im Speicher abgelegt. Der Speicherbedarf zur Speicherung des Modells ist also proportional zur Anzahl der Winkelschritte, zur Anzahl der Skalierungsschritte und zur Anzahl der Punkte im Modell. Wenn also AngleStep, ScaleRStep oder ScaleCStep zu klein bzw. AngleExtent oder der Skalierungsbereich zu groß gewählt werden, kann es vorkommen, dass das Modell nicht mehr in den (virtuellen) Speicher passt. In diesem Fall muss entweder AngleStep, ScaleRStep oder ScaleCStep größer oder AngleExtent oder der Skalierungsbereich kleiner gewählt werden. In jedem Fall ist es aus Laufzeitgründen vorteilhaft, wenn das Modell komplett in den Hauptspeicher passt und somit ein Paging durch das Betriebssystem vermieden werden kann. Da die Möglichkeit zur subpixelgenauen Winkelbestimmung in find_aniso_shape_model gegeben ist, kann bei Modellen mit einem Durchmesser von ca. 200 Pixeln AngleStep $\geq 1°$ und ScaleRStep, ScaleCStep ≥ 0.02 gewählt werden.

Falls AngleStep = *'auto'* oder ScaleRStep, ScaleCStep = *'auto'* übergeben wird, wählt

create_aniso_shape_model_xld automatisch eine Schrittweite für die Winkel bzw. Skalierungen basierend auf der Größe des Modells aus. Die automatisch gewählten Schrittweiten können mit get_shape_model_params abgefragt werden.

Falls eine vollständige Vorabgenerierung des Modells nicht gewählt wird, wird das Modell nur in einer Referenzlage auf jeder Ebene erzeugt. In diesem Fall muss das Modell zur Laufzeit in find_aniso_shape_model auf die verschiedenen Winkel und Skalierungen transformiert werden. Dadurch kann es vorkommen, dass das Auffinden des Modells etwas mehr Zeit benötigt.

Es ist zu beachten, dass vorabgenerierte Modelle für eine spezifische Bildgröße bestimmt sind. Aus Laufzeitgründen ist die Verwendung von Bildern mit verschiedenen Größen während der Suche mit dem gleichen Modell nicht parallel möglich. In diesem Fall müssen Kopien des gleichen Modells verwendet werden. Anderenfalls kann das Programm abstürzen.

Achtung

Die XLD-Konturen, die in Contours übergeben werden, sollten in etwa auf die mittlere Größe skaliert worden sein, in der das Objekt in den Suchbildern erscheint. Die Produkte ScaleRMin × ScaleRMax und ScaleCMin × ScaleCMax sollten also etwa 1 ergeben.

Im Gegensatz zum Operator create_aniso_shape_model ist es nicht möglich die Mindestgröße der Modellkomponenten anzugeben. Um kleine Modellkomponenten im Formmodell zu vermeiden, können kurze Konturen vor dem Aufruf von create_aniso_shape_model_xld mit dem Operator select_contours_xld eliminiert werden.

Parameter

▷ **Contours** (input_object) . . . . . xld_cont(-array) ⇝ *object*
Eingabekonturen, die zum Aufbau des Modells verwendet werden.

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: NumLevels ∈ {'auto', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Musters.
Defaultwert: -0.39
Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: AngleExtent ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: AngleStep ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep > 0 && AngleStep <= pi / 16

▷ **ScaleRMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Zeilenrichtung.
Defaultwert: 0.9
Wertevorschläge: ScaleRMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **ScaleRMax** (input_control) . . . . . number ⇝ *real*
Größte auftretende Skalierung des Musters in Zeilenrichtung.
Defaultwert: 1.1
Wertevorschläge: ScaleRMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleRMax >= ScaleRMin

▷ **ScaleRStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Zeilenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleRStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleRStep > 0

▷ **ScaleCMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 0.9
Wertevorschläge: ScaleCMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number ⇝ *real*
Größte auftretende Skalierung des Musters in Spaltenrichtung.
Defaultwert: 1.1
Wertevorschläge: ScaleCMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleCMax >= ScaleCMin

▷ **ScaleCStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierung (Auflösung) in Spaltenrichtung.
Defaultwert: 'auto'
Wertevorschläge: ScaleCStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleCStep > 0

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung und optional Methode der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: Optimization ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high', 'pregeneration', 'no_pregeneration'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'ignore_local_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer*
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 5
Wertevorschläge: MinContrast ∈ {1, 2, 3, 5, 7, 10, 20, 30, 40}

▷ **ModelID** (output_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_aniso_shape_model_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter NumLevels so gewählt wurde, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_contour_xld_dxf, edges_sub_pix, select_contours_xld

Nachfolger

find_aniso_shape_model, find_aniso_shape_models, get_shape_model_params, clear_shape_model, write_shape_model, set_shape_model_origin, set_shape_model_param, set_shape_model_metric, set_shape_model_clutter

Alternativen

create_shape_model_xld, create_scaled_shape_model_xld

Siehe auch

set_system, get_system

Modul

Matching

create_generic_shape_model (: : : ModelID)

Erstellen eines Formmodells.

Der Operator create_generic_shape_model erstellt ein Formmodell und gibt dessen Handle in ModelID zurück.

Das erstellte Modell ist nicht trainiert. Für Informationen zum Trainieren eines Modells siehe train_generic_shape_model.

Parameter

▷ **ModelID** (output_control) ..... shape_model ⇝ *handle*
Handle des Formmodells.

Ergebnis

Wird create_generic_shape_model erfolgreich ausgeführt, liefert der Operator den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

set_generic_shape_model_param, train_generic_shape_model, write_shape_model, set_shape_model_origin

Modul

Matching

create_scaled_shape_model (Template : : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleMin, ScaleMax, ScaleStep, Optimization, Metric, Contrast, MinContrast : ModelID)

Vorbereiten eines isotrop skalierten Formmodells für das Matching.

create_scaled_shape_model bereitet ein Muster, das als Bild Template übergeben wird, als isotrop skaliertes Formmodell für Matching vor. Die ROI des Modells wird als der Definitionsbereich von Template übergeben.

Der Ausgabeparameter ModelID ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von find_scaled_shape_model verwendet wird. Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt der Region des Modellbildes Template verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit set_shape_model_origin gesetzt werden. Das Modell wird auf mehreren Pyramidenebenen generiert und im Speicher abgelegt. Falls eine vollständige Vorabgenerierung des Modells gewählt wird (siehe unten), wird das Modell in mehreren Rotationen und Skalierungen auf jeder Ebene generiert. Das Modell kann mit set_shape_model_clutter um Störparameter erweitert werden.

Eingabeparameter im Detail

NumLevels: Die Anzahl der Pyramidenebenen wird mit NumLevels festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von NumLevels ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens vier). Dies kann anhand der Ausgabe von inspect_shape_model überprüft werden. Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entstehen

würde, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert create_scaled_shape_model eine Fehlermeldung zurück.

Falls für NumLevels *'auto'* (oder *0* für Rückwärtskompatibilität) übergeben wird, wählt create_scaled_shape_model die Anzahl der Pyramidenstufen automatisch. Die automatisch gewählte Anzahl von Pyramidenstufen kann mit get_shape_model_params abgefragt werden. In seltenen Fällen kann es vorkommen, dass create_scaled_shape_model die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für MinScore oder Greediness in find_scaled_shape_model selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in find_scaled_shape_model kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen mit Hilfe der Ausgabe von inspect_shape_model gewählt werden.

AngleStart, AngleExtent und AngleStep: Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Modells im Bild fest. Das Modell kann also mit find_scaled_shape_model nur in diesem Winkelbereich gefunden werden. Der Parameter AngleStep gibt die Schrittweite der Winkel in dem gewählten Winkelbereich an. Falls bei find_scaled_shape_model keine Subpixelgenauigkeit spezifiziert wird, gibt AngleStep also die erreichbare Winkelgenauigkeit an. AngleStep sollte aufgrund der Größe des Objektes gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte AngleStep für kleinere Modelle größer gewählt werden. Falls AngleExtent kein ganzzahliges Vielfaches von AngleStep ist, wird AngleStep entsprechend angepasst. Um sicherzustellen, dass find_scaled_shape_model für Modellinstanzen ohne Rotation Winkel von exakt 0.0 zurückgibt, wird der Winkelbereich der möglichen Rotationen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den AngleStart plus n mal AngleStep genau Null ergibt, wird AngleStart um höchstens AngleStep verringert, und AngleExtent wird um AngleStep vergrößert.

ScaleMin, ScaleMax und ScaleStep: Die Parameter ScaleMin und ScaleMax legen den Bereich der möglichen Skalierung (Größe) des Modells im Bild fest. Eine Skalierung von 1 entspricht der Originalgröße des Modells. Der Parameter ScaleStep gibt die Schrittweite der Skalierungen an. Falls bei find_scaled_shape_model keine Subpixelgenauigkeit spezifiziert wird, gibt ScaleStep also die erreichbare Genauigkeit der Skalierung an. Wie AngleStep sollte ScaleStep aufgrund der Größe des Objektes gewählt werden. Falls der Skalierungsbereich kein ganzzahliges Vielfaches von ScaleStep ist, wird ScaleStep entsprechend angepasst. Um sicherzustellen, dass find_scaled_shape_model für Modellinstanzen ohne Skalierung Skalierungen von exakt 1.0 zurückgibt, wird der Bereich der möglichen Skalierungen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den ScaleMin plus n mal ScaleStep genau 1.0 ergibt, wird ScaleMin um höchstens ScaleStep verringert, und ScaleMax wird so erhöht, dass der Skalierungsbereich um ScaleStep vergrößert wird.

Optimization: Bei besonders großen Modellen kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters Optimization auf einen Wert ungleich *'none'* zu setzen. Falls Optimization = *'none'*, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter Optimization reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei find_scaled_shape_model notwendig werden, den Parameter Greediness auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen.

Falls für Optimization *'auto'* übergeben wird, wählt create_scaled_shape_model die Reduktion der Punkte automatisch.

Metric: Der Parameter Metric legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird.

Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist.

Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_scaled_shape_model erhöht sich in diesem Fall geringfügig.

Falls Metric = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Da sich in diesem Fall die Laufzeit von find_scaled_shape_model aber wesentlich erhöht, ist es in solchen

Fällen meist sinnvoller, mehrere Modelle mit create_scaled_shape_model zu erzeugen und mit find_scaled_shape_models zu suchen.

Die obigen drei Metriken können nur auf einkanalige Bilder angewendet werden. Falls im Modellbild oder in den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes verwendet.

Falls Metric = *'ignore_color_polarity'*, wird das Modell auch dann gefunden, falls sich die Farbkontraste lokal ändern können. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün. Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von find_scaled_shape_model erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für Metric = *'ignore_color_polarity'* die Anzahl der Kanäle bei der Modellgenerierung mit create_scaled_shape_model und bei der Suche mit find_scaled_shape_model verschieden sein können. Dies erlaubt z.B. die Modellgenerierung aus einem einkanaligen synthetisch generierten Bild. Es ist auch zu beachten, dass die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein.

Contrast: Der Parameter Contrast legt fest, welchen Grauwertkontrast die Punkte des Modells besitzen müssen. Der Kontrast ist ein Maß für die lokalen Grauwertdifferenzen zwischen dem Objekt und dem Hintergrund und zwischen verschiedenen Teilen des Objektes. Contrast sollte so gewählt werden, dass nur die signifikanten Merkmale des Musters für das Modell verwendet werden. Contrast kann auch ein Tupel mit zwei Werten enthalten. In diesem Fall wird bei der Segmentierung des Modells ein Verfahren, das ähnlich zum Hysterese-Schwellenwert-Verfahren in edges_image ist, eingesetzt. Dabei ist der erste Wert des Tupels der untere Schwellenwert und der zweite Wert der obere Schwellenwert. Für weitere Informationen über das Hysterese-Schwellenwert-Verfahren siehe hysteresis_threshold. Optional kann in Contrast auch ein dritter Wert als letztes Element des Tupels übergeben werden. Dieser Wert dient zur Selektion von signifikanten Komponenten des Modells basierend auf der Größe der Komponenten, d.h. Komponenten, deren Anzahl von Punkten kleiner ist als diese Mindestgröße, werden unterdrückt. Da sich die Mindestgröße auf die Größe der Komponenten bezieht kann es auch vorkommen, dass die daraus abgeleiteten Modellkonturen trotzdem kleiner als die angegebene Mindestgröße sind. Dieser Schwellenwert wird pro Pyramidenstufe halbiert. Falls eine Unterdrückung von kleinen Modellkomponenten gewünscht ist, aber keine Hysterese-Schwellenwert-Operation, müssen dennoch drei Werte in Contrast übergeben werden, von denen die ersten zwei auf denselben Wert gesetzt werden. Die Wirkung von Contrast kann auch vorab mit inspect_shape_model überprüft werden.

Wird Contrast auf *'auto'* gesetzt, dann bestimmt create_scaled_shape_model die beschriebenen drei Werte automatisch. Es besteht die Möglichkeit, nur den Kontrast (*'auto_contrast'*), die Hysterese-Schwellenwerte (*'auto_contrast_hyst'*) oder die Mindestgröße (*'auto_min_size'*) automatisch bestimmen zu lassen. Die übrigen, nicht automatisch zu bestimmenden Größen, können dann zusätzlich als Tupel übergeben werden. Dabei sind auch Kombinationen zulässig: Wird z.B. *['auto_contrast','auto_min_size']* übergeben, wird sowohl der Kontrast als auch die Mindestgröße bestimmt. Bei *['auto_min_size',20,30]* wird die Mindestgröße automatisch ermittelt, während die Hysterese-Schwellen auf *20* und *30* gesetzt werden, etc. In bestimmten Fällen kann es vorkommen, dass die automatische Bestimmung der Kontrast-Schwellenwerte nicht zum gewünschten Ergebnis führt. So ist eine manuelle Parameterwahl zu bevorzugen, wenn anwendungsbedingt bestimmte Modellkomponenten ausgeschlossen oder ins Modell integriert werden sollen oder wenn das Objekt mehrere unterschiedliche Kontraststufen umfasst. Daher ist es sinnvoll, die Kontrastwerte vor der Modellgenerierung mit determine_shape_model_params automatisch zu ermitteln und das Ergebnis mit inspect_shape_model zu überprüfen. Es ist zu beachten, dass MinContrast die Bestimmung der Kontrast-Schwellenwerte beeinflusst, und daher auch die Schätzung der Mindestgröße.

MinContrast: Mit MinContrast wird festgelegt, welchen Grauwertkontrast das Modell später bei der Erkennung mit find_scaled_shape_model im Bild mindestens besitzen muss. Mit anderen Worten stellt dieser Parameter somit eine Abgrenzung des Musters von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte MinContrast auf 10 gesetzt werden. Falls mehrkanalige Bilder für das Modell und für die Suchbilder verwendet werden, und falls der Parameter Metric auf *'ignore_color_polarity'* gesetzt wird (siehe oben), muss das Rauschen in einem Kanal noch mit der Wurzel der Anzahl der Kanäle multipliziert werden, um MinContrast zu bestimmen. Falls die Grauwerte in einem Kanal z.B. in einem Bereich von 10 Graustufen schwanken, sollte MinContrast bei einem dreikanaligen Bild auf 17 gesetzt werden. Offensichtlich muss MinContrast

kleiner als Contrast sein. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss MinContrast entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte MinContrast etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells zu gewährleisten.

Wird MinContrast auf *'auto'* gesetzt, wählt create_scaled_shape_model den Mindestkontrast basierend auf dem Rauschen im Modellbild automatisch. Eine automatische Bestimmung ergibt demzufolge nur Sinn, wenn das bei der Erkennung zu erwartende Bildrauschen dem Rauschen im Modellbild entspricht. Außerdem kann es in manchen Fällen sinnvoll sein, den automatisch ermittelten Wert zu erhöhen, um eine höhere Robustheit gegenüber Verdeckungen zu erzielen (siehe oben). Der automatisch bestimmte Mindestkontrast kann mit get_shape_model_params abgefragt werden.

Vollständige Vorabgenerierung des Modells

Optional kann in Optimization ein zweiter Wert übergeben werden. Dieser Wert legt fest, ob das Modell vollständig vorab generiert wird oder nicht. Der zweite Wert von Optimization ist hierzu auf *'pregeneration'* oder *'no_pregeneration'* zu setzen. Falls der zweite Wert nicht angegeben wird, wird der mit set_system ('pregenerate_shape_models',...) eingestellte Modus verwendet. Mit der Standardeinstellung (*'pregenerate_shape_models'* = *'false'*) wird das Modell nicht vollständig vorab generiert. Die vollständige Vorabgenerierung des Modells führt im Normalfall zu leicht geringeren Laufzeiten in find_scaled_shape_model, da das Modell nicht zur Laufzeit transformiert werden muss. Allerdings sind in diesem Fall der Speicherbedarf und die Zeit, die zur Modellerzeugung benötigt wird, wesentlich höher. Zu beachten ist auch, dass in den zwei Modi nicht dieselben Ergebnisse erwartet werden können, da intern bei der Transformation zur Laufzeit notwendigerweise andere Daten für die transformierten Modelle entstehen, als bei der Vorabgenerierung. Falls das Modell nicht vorab generiert wird, werden z.B. in find_scaled_shape_model typischerweise leicht niedrigere Bewertungen (Scores) zurückgeliefert, so dass es unter Umständen notwendig sein kann, einen niedrigeren Wert für MinScore als bei vollständiger Vorabgenerierung zu verwenden. Außerdem können sich die durch Interpolation gewonnenen Subpixelpositionen geringfügig unterscheiden. Falls höchste Genauigkeit gewünscht wird, sollte die Lage des Modells mittels Ausgleichsrechnung (Least-Squares Adjustment) bestimmt werden.

Falls eine vollständige Vorabgenerierung des Modells gewählt wird, wird das Modell in dem gewählten Winkelbereich und Skalierungsbereich vorab generiert und im Speicher abgelegt. Der Speicherbedarf zur Speicherung des Modells ist also proportional zur Anzahl der Winkelschritte, zur Anzahl der Skalierungsschritte und zur Anzahl der Punkte im Modell. Wenn also AngleStep oder ScaleStep zu klein bzw. AngleExtent oder der Skalierungsbereich zu groß gewählt werden, kann es vorkommen, dass das Modell nicht mehr in den (virtuellen) Speicher passt. In diesem Fall muss entweder AngleStep oder ScaleStep größer oder AngleExtent oder der Skalierungsbereich kleiner gewählt werden. In jedem Fall ist es aus Laufzeitgründen vorteilhaft, wenn das Modell komplett in den Hauptspeicher passt und somit ein Paging durch das Betriebssystem vermieden werden kann. Da die Möglichkeit zur subpixelgenauen Winkelbestimmung in find_scaled_shape_model gegeben ist, kann bei Modellen mit einem Durchmesser von ca. 200 Pixeln AngleStep $\geq 1°$ und ScaleStep ≥ 0.02 gewählt werden.

Falls AngleStep $= 'auto'$ oder ScaleStep $= 'auto'$ (oder jeweils *0* für Rückwärtskompatibilität) übergeben wird, wählt create_scaled_shape_model automatisch eine Schrittweite für die Winkel bzw. Skalierungen basierend auf der Größe des Modells aus. Die automatisch gewählten Schrittweiten können mit get_shape_model_params abgefragt werden.

Falls eine vollständige Vorabgenerierung des Modells nicht gewählt wird, wird das Modell nur in einer Referenzlage auf jeder Ebene erzeugt. In diesem Fall muss das Modell zur Laufzeit in find_scaled_shape_model auf die verschiedenen Winkel und Skalierungen transformiert werden. Dadurch kann es vorkommen, dass das Auffinden des Modells etwas mehr Zeit benötigt.

Es ist zu beachten, dass vorabgenerierte Modelle für eine spezifische Bildgröße bestimmt sind. Aus Laufzeitgründen ist die Verwendung von Bildern mit verschiedenen Größen während der Suche mit dem gleichen Modell nicht parallel möglich. In diesem Fall müssen Kopien des gleichen Modells verwendet werden. Anderenfalls kann das Programm abstürzen.

Parameter

▷ **Template** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: NumLevels ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Musters.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: `AngleStep` ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep >= 0 && AngleStep <= pi / 16

▷ **ScaleMin** (input_control) . . . . . number ⇝ *real*
Kleinste auftretende Skalierung des Musters.
Defaultwert: 0.9
Wertevorschläge: `ScaleMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleMin > 0

▷ **ScaleMax** (input_control) . . . . . number ⇝ *real*
Größte auftretende Skalierung des Musters.
Defaultwert: 1.1
Wertevorschläge: `ScaleMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleMax >= ScaleMin

▷ **ScaleStep** (input_control) . . . . . number ⇝ *real* / string
Schrittweite der Skalierungen (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: `ScaleStep` ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
Restriktion: ScaleStep >= 0

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung und optional Methode der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: `Optimization` ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high', 'pregeneration', 'no_pregeneration'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **Contrast** (input_control) . . . . . number(-array) ⇝ *integer* / string
Schwellenwert bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Musterbild und optional Mindestgröße der Objektteile.
Defaultwert: 'auto'
Wertevorschläge: `Contrast` ∈ {'auto', 'auto_contrast', 'auto_contrast_hyst', 'auto_min_size', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer* / string
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: `MinContrast` ∈ {'auto', 1, 2, 3, 5, 7, 10, 20, 30, 40}
Restriktion: MinContrast < Contrast

▷ **ModelID** (output_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_scaled_shape_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn die Parameter `NumLevels` und `Contrast` so gewählt worden sind, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`draw_region`, `reduce_domain`, `threshold`

Nachfolger

`find_scaled_shape_model`, `find_scaled_shape_models`, `get_shape_model_params`, `clear_shape_model`, `write_shape_model`, `set_shape_model_origin`, `set_shape_model_clutter`

Alternativen

`create_shape_model`, `create_aniso_shape_model`

Siehe auch

`set_system`, `get_system`

Modul

Matching

`create_scaled_shape_model_xld` `( Contours : : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleMin, ScaleMax, ScaleStep, Optimization, Metric, MinContrast : ModelID )`

Vorbereiten eines isotrop skalierten Formmodells für das Matching aus XLD-Konturen.

`create_scaled_shape_model_xld` erzeugt ein isotrop skaliertes Formmodell für das Matching aus den XLD Konturen `Contours`. Die XLD Konturen repräsentieren dabei die Grauwertkanten des zu suchenden Objekts im Bild. Im Gegensatz zu dem Operator `create_scaled_shape_model`, der ein Formmodell aus einem Bild erzeugt, erzeugt der Operator `create_scaled_shape_model_xld` das Formmodell aus XLD Konturen, d.h., ohne die Verwendung eines Bildes.

Der Ausgabeparameter `ModelID` ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von `find_scaled_shape_model` verwendet wird. Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt des kleinsten umschließenden Rechtecks der Konturen `Contours`, das parallel zu den Koordinatenachsen liegt, verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit `set_shape_model_origin` gesetzt werden. Das Modell wird für mehrere Pyramidenebenen generiert und im Speicher abgelegt. Falls eine vollständige Vorabgenerierung des Modells gewählt wird (siehe unten), wird das Modell in mehreren Rotationen und Skalierungen auf jeder Ebene generiert. Das Modell kann mit `set_shape_model_clutter` um Störparameter erweitert werden.

Eingabeparameter im Detail

NumLevels: Die Anzahl der Pyramidenebenen wird mit `NumLevels` festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von `NumLevels` ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens vier). Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entstehen würde, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert `create_scaled_shape_model_xld` eine Fehlermeldung zurück.

Falls für `NumLevels` *'auto'* übergeben wird, wählt `create_scaled_shape_model_xld` die Anzahl der Pyramidenstufen automatisch. Die gewählte Anzahl von Pyramidenstufen kann mit `get_shape_model_params` abgefragt werden. In seltenen Fällen kann es vorkommen, dass `create_scaled_shape_model_xld` die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für `MinScore` oder `Greediness` in `find_scaled_shape_model` selektiert werden müssen, damit das Modell gefunden

wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in `find_scaled_shape_model` kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen manuell gewählt werden.

AngleStart, AngleExtent und AngleStep: Die Parameter `AngleStart` und `AngleExtent` legen den Winkelbereich für die möglichen Rotationen des Objektes im Bild fest. Das Objekt kann also mit `find_scaled_shape_model` nur in diesem Winkelbereich gefunden werden. Der Parameter `AngleStep` gibt die Schrittweite der Winkel in dem gewählten Winkelbereich an. Falls bei `find_scaled_shape_model` keine Subpixelgenauigkeit spezifiziert wird, gibt `AngleStep` also die erreichbare Winkelgenauigkeit an. `AngleStep` sollte aufgrund der Größe des Objektes gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte `AngleStep` für kleinere Modelle größer gewählt werden. Falls `AngleExtent` kein ganzzahliges Vielfaches von `AngleStep` ist, wird `AngleStep` entsprechend angepasst. Um sicherzustellen, dass `find_scaled_shape_model` für Modellinstanzen ohne Rotation Winkel von exakt 0.0 zurückgibt, wird der Winkelbereich der möglichen Rotationen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den `AngleStart` plus n mal `AngleStep` genau Null ergibt, wird `AngleStart` um höchstens `AngleStep` verringert, und `AngleExtent` wird um `AngleStep` vergrößert.

ScaleMin, ScaleMax und ScaleStep: Die Parameter `ScaleMin` und `ScaleMax` legen den Bereich der möglichen Skalierung (Größe) des Objektes im Bild fest. Eine Skalierung von 1 entspricht der Originalgröße des Modells. Der Parameter `ScaleStep` gibt die Schrittweite der Skalierungen an. Falls bei `find_scaled_shape_model` keine Subpixelgenauigkeit spezifiziert wird, gibt `ScaleStep` also die erreichbare Genauigkeit der Skalierung an. Wie `AngleStep` sollte `ScaleStep` aufgrund der Größe des Objektes gewählt werden. Falls der Skalierungsbereich kein ganzzahliges Vielfaches von `ScaleStep` ist, wird `ScaleStep` entsprechend angepasst. Um sicherzustellen, dass `find_scaled_shape_model` für Modellinstanzen ohne Skalierung Skalierungen von exakt 1.0 zurückgibt, wird der Bereich der möglichen Skalierungen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den `ScaleMin` plus n mal `ScaleStep` genau 1.0 ergibt, wird `ScaleMin` um höchstens `ScaleStep` verringert, und `ScaleMax` wird so erhöht, dass der Skalierungsbereich um `ScaleStep` vergrößert wird.

Optimization: Bei besonders großen Modellen kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters `Optimization` auf einen Wert ungleich *'none'* zu setzen. Falls `Optimization` = *'none'*, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter `Optimization` reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei `find_scaled_shape_model` notwendig werden, den Parameter `Greediness` auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen.

Falls für `Optimization` *'auto'* übergeben wird, wählt `create_scaled_shape_model_xld` die Reduktion der Punkte automatisch.

Metric: Der Parameter `Metric` legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird.

Falls `Metric` = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist.

Falls `Metric` = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von `find_scaled_shape_model` erhöht sich in diesem Fall geringfügig.

Es ist zu beachten, dass die beiden Werte *'use_polarity'* und *'ignore_global_polarity'* für `Metric` nur dann verwendet werden können, wenn bei allen Konturen, die in `Contours` übergeben wurden, das Attribut *'edge_direction'* gesetzt ist, da es die Polarität der Kanten festlegt. Für weitere Informationen zu Konturattributen wie *'edge_direction'* siehe `get_contour_attrib_xld`. Ansonsten besteht die Möglichkeit, die beiden Werte für `Metric` mit dem Operator `set_shape_model_metric` zu setzen, der die Polarität der Kanten aus einem Bild bestimmt.

Falls `Metric` = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Da sich in diesem Fall die Laufzeit von `find_scaled_shape_model` aber wesentlich erhöht, ist es in solchen Fällen meist sinnvoller, mehrere Modelle mit `create_scaled_shape_model_xld` zu erzeugen und mit `find_scaled_shape_models` zu suchen.

Die obigen drei Metriken können nur auf einkanalige Bilder angewendet werden. Falls im Modellbild oder in

den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes verwendet.

Falls `Metric` = *'ignore_color_polarity'*, wird das Modell auch dann gefunden, falls sich die Farbkontraste lokal ändern können. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün. Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von `find_scaled_shape_model` erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für `Metric` = *'ignore_color_polarity'* die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein.

Es ist auch zu beachten, dass die beiden ersten Werte für `Metric` (*'use_polarity'* und *'ignore_global_polarity'*) nur dann verwendet werden können, wenn bei allen Konturen, die in `Contours` übergeben wurden, das Attribut *'edge_direction'* gesetzt ist, da es die Polarität der Kanten festlegt. Für weitere Informationen zu Konturattributen wie *'edge_direction'* siehe `get_contour_attrib_xld`. Ansonsten besteht die Möglichkeit, die beiden Werte für `Metric` mit dem Operator `set_shape_model_metric` zu setzen, der die Polarität der Kanten aus einem Bild bestimmt.

MinContrast: Mit `MinContrast` wird festgelegt, welchen Grauwertkontrast die Objektkanten später bei der Erkennung mit `find_scaled_shape_model` im Suchbild mindestens besitzen müssen. Mit anderen Worten stellt dieser Parameter somit eine Abgrenzung des Objektes von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte `MinContrast` auf 10 gesetzt werden. Falls mehrkanalige Bilder für das Modell und für die Suchbilder verwendet werden, und falls der Parameter `Metric` auf *'ignore_color_polarity'* gesetzt wird (siehe oben), muss das Rauschen in einem Kanal noch mit der Wurzel der Anzahl der Kanäle multipliziert werden, um `MinContrast` zu bestimmen. Falls die Grauwerte in einem Kanal z.B. in einem Bereich von 10 Graustufen schwanken, sollte `MinContrast` bei einem dreikanaligen Bild auf 17 gesetzt werden. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss `MinContrast` entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte `MinContrast` etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells zu gewährleisten.

Vollständige Vorabgenerierung des Modells

Optional kann in `Optimization` ein zweiter Wert übergeben werden. Dieser Wert legt fest, ob das Modell vollständig vorab generiert wird oder nicht. Der zweite Wert von `Optimization` ist hierzu auf *'pregeneration'* oder *'no_pregeneration'* zu setzen. Falls der zweite Wert nicht angegeben wird, wird der mit `set_system ('pregenerate_shape_models',...)` eingestellte Modus verwendet. Mit der Standardeinstellung (*'pregenerate_shape_models'* = *'false'*) wird das Modell nicht vollständig vorab generiert. Die vollständige Vorabgenerierung des Modells führt im Normalfall zu leicht geringeren Laufzeiten in `find_scaled_shape_model`, da das Modell nicht zur Laufzeit transformiert werden muss. Allerdings sind in diesem Fall der Speicherbedarf und die Zeit, die zur Modellerzeugung benötigt wird, wesentlich höher. Zu beachten ist auch, dass in den zwei Modi nicht dieselben Ergebnisse erwartet werden können, da intern bei der Transformation zur Laufzeit notwendigerweise andere Daten für die transformierten Modelle entstehen, als bei der Vorabgenerierung. Falls das Modell nicht vorab generiert wird, werden z.B. in `find_scaled_shape_model` typischerweise leicht niedrigere Bewertungen (Scores) zurückgeliefert, so dass es unter Umständen notwendig sein kann, einen niedrigeren Wert für `MinScore` als bei vollständiger Vorabgenerierung zu verwenden. Außerdem können sich die durch Interpolation gewonnenen Subpixelpositionen geringfügig unterscheiden. Falls höchste Genauigkeit gewünscht wird, sollte die Lage des Modells mittels Ausgleichsrechnung (Least-Squares Adjustment) bestimmt werden.

Falls eine vollständige Vorabgenerierung des Modells gewählt wird, wird das Modell in dem gewählten Winkelbereich und Skalierungsbereich vorab generiert und im Speicher abgelegt. Der Speicherbedarf zur Speicherung des Modells ist also proportional zur Anzahl der Winkelschritte, zur Anzahl der Skalierungsschritte und zur Anzahl der Punkte im Modell. Wenn also `AngleStep` oder `ScaleStep` zu klein bzw. `AngleExtent` oder der Skalierungsbereich zu groß gewählt werden, kann es vorkommen, dass das Modell nicht mehr in den (virtuellen) Speicher passt. In diesem Fall muss entweder `AngleStep` oder `ScaleStep` größer oder `AngleExtent` oder der Skalierungsbereich kleiner gewählt werden. In jedem Fall ist es aus Laufzeitgründen vorteilhaft, wenn das Modell komplett in den Hauptspeicher passt und somit ein Paging durch das Betriebssystem vermieden werden kann. Da die Möglichkeit zur subpixelgenauen Winkelbestimmung in `find_scaled_shape_model` gegeben ist, kann bei Modellen mit einem Durchmesser von ca. 200 Pixeln `AngleStep` $\geq 1^\circ$ und `ScaleStep` ≥ 0.02 gewählt werden.

Falls AngleStep = 'auto' oder ScaleStep = 'auto' übergeben wird, wählt create_scaled_shape_model_xld automatisch eine Schrittweite für die Winkel bzw. Skalierungen basierend auf der Größe des Modells aus. Die automatisch gewählten Schrittweiten können mit get_shape_model_params abgefragt werden.

Falls eine vollständige Vorabgenerierung des Modells nicht gewählt wird, wird das Modell nur in einer Referenzlage auf jeder Ebene erzeugt. In diesem Fall muss das Modell zur Laufzeit in find_scaled_shape_model auf die verschiedenen Winkel und Skalierungen transformiert werden. Dadurch kann es vorkommen, dass das Auffinden des Modells etwas mehr Zeit benötigt.

Es ist zu beachten, dass vorabgenerierte Modelle für eine spezifische Bildgröße bestimmt sind. Aus Laufzeitgründen ist die Verwendung von Bildern mit verschiedenen Größen während der Suche mit dem gleichen Modell nicht parallel möglich. In diesem Fall müssen Kopien des gleichen Modells verwendet werden. Anderenfalls kann das Programm abstürzen.

Achtung

Die XLD-Konturen, die in Contours übergeben werden, sollten in etwa auf die mittlere Größe skaliert worden sein, in der das Objekt in den Suchbildern erscheint. Das Produkt ScaleMin × ScaleMax sollte also etwa 1 ergeben.

Im Gegensatz zum Operator create_scaled_shape_model ist es nicht möglich die Mindestgröße der Modellkomponenten anzugeben. Um kleine Modellkomponenten im Formmodell zu vermeiden, können kurze Konturen vor dem Aufruf von create_scaled_shape_model_xld mit dem Operator select_contours_xld eliminiert werden.

Parameter

- ▷ **Contours** (input_object) . . . . . xld_cont(-array) ⇝ *object*
  Eingabekonturen, die zum Aufbau des Modells verwendet werden.
- ▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
  Maximale Anzahl von Pyramidenebenen.
  Defaultwert: 'auto'
  Werteliste: NumLevels ∈ {'auto', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
  Kleinste auftretende Rotation des Musters.
  Defaultwert: -0.39
  Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Defaultwert: 0.79
  Wertevorschläge: AngleExtent ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
  Restriktion: AngleExtent >= 0
- ▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real* / string
  Schrittweite der Winkel (Auflösung).
  Defaultwert: 'auto'
  Wertevorschläge: AngleStep ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
  Restriktion: AngleStep > 0 && AngleStep <= pi / 16
- ▷ **ScaleMin** (input_control) . . . . . number ⇝ *real*
  Kleinste auftretende Skalierung des Musters.
  Defaultwert: 0.9
  Wertevorschläge: ScaleMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: ScaleMin > 0
- ▷ **ScaleMax** (input_control) . . . . . number ⇝ *real*
  Größte auftretende Skalierung des Musters.
  Defaultwert: 1.1
  Wertevorschläge: ScaleMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
  Restriktion: ScaleMax >= ScaleMin
- ▷ **ScaleStep** (input_control) . . . . . number ⇝ *real* / string
  Schrittweite der Skalierungen (Auflösung).
  Defaultwert: 'auto'
  Wertevorschläge: ScaleStep ∈ {'auto', 0.01, 0.02, 0.05, 0.1, 0.15, 0.2}
  Restriktion: ScaleStep > 0

▷ **Optimization** (input_control) ..... string(-array) ⇝ *string*
Art der Optimierung und optional Methode der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: Optimization ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high', 'pregeneration', 'no_pregeneration'}

▷ **Metric** (input_control) ..... string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'ignore_local_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **MinContrast** (input_control) ..... number ⇝ *integer*
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 5
Wertevorschläge: MinContrast ∈ {1, 2, 3, 5, 7, 10, 20, 30, 40}

▷ **ModelID** (output_control) ..... shape_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_scaled_shape_model_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter NumLevels so gewählt wurde, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_contour_xld_dxf, edges_sub_pix, select_contours_xld

Nachfolger

find_scaled_shape_model, find_scaled_shape_models, get_shape_model_params, clear_shape_model, write_shape_model, set_shape_model_origin, set_shape_model_param, set_shape_model_metric, set_shape_model_clutter

Alternativen

create_shape_model_xld, create_aniso_shape_model_xld

Siehe auch

set_system, get_system

Modul

Matching

```
create_shape_model ( Template : : NumLevels, AngleStart,
    AngleExtent, AngleStep, Optimization, Metric, Contrast,
    MinContrast : ModelID )
```

Vorbereiten eines Formmodells für das Matching.

create_shape_model bereitet ein Muster, das als Bild Template übergeben wird, als Formmodell für das Matching vor. Die ROI des Modells wird als der Definitionsbereich von Template übergeben.

Der Ausgabeparameter ModelID ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von find_shape_model verwendet wird. Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt der Region des Modellbildes Template verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit set_shape_model_origin gesetzt werden. Das Modell wird auf mehreren Pyramidenebenen generiert und

im Speicher abgelegt. Falls eine vollständige Vorabgenerierung des Modells gewählt wird (siehe unten), wird das Modell in mehreren Rotationen auf jeder Ebene generiert. Das Modell kann mit set_shape_model_clutter um Störparameter erweitert werden.

Eingabeparameter im Detail

NumLevels: Die Anzahl der Pyramidenebenen wird mit NumLevels festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von NumLevels ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens vier). Dies kann anhand der Ausgabe von inspect_shape_model überprüft werden. Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entstehen würde, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert create_shape_model eine Fehlermeldung zurück.

Falls für NumLevels *'auto'* (oder *0* für Rückwärtskompatibilität) übergeben wird, wählt create_shape_model die Anzahl der Pyramidenstufen automatisch. Die automatisch gewählte Anzahl von Pyramidenstufen kann mit get_shape_model_params abgefragt werden. In seltenen Fällen kann es vorkommen, dass create_shape_model die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für MinScore oder Greediness in find_shape_model selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in find_shape_model kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen mit Hilfe der Ausgabe von inspect_shape_model gewählt werden.

AngleStart, AngleExtent und AngleStep: Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Modells im Bild fest. Das Modell kann also mit find_shape_model nur in diesem Winkelbereich gefunden werden. Der Parameter AngleStep gibt die Schrittweite der Winkel in dem gewählten Winkelbereich an. Falls bei find_shape_model keine Subpixelgenauigkeit spezifiziert wird, gibt AngleStep also die erreichbare Winkelgenauigkeit an. AngleStep sollte aufgrund der Größe des Objektes gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte AngleStep für kleinere Modelle größer gewählt werden. Falls AngleExtent kein ganzzahliges Vielfaches von AngleStep ist, wird AngleStep entsprechend angepasst. Um sicherzustellen, dass find_shape_model für Modellinstanzen ohne Rotation Winkel von exakt 0.0 zurückgibt, wird der Winkelbereich der möglichen Rotationen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den AngleStart plus n mal AngleStep genau Null ergibt, wird AngleStart um höchstens AngleStep verringert, und AngleExtent wird um AngleStep vergrößert.

Optimization: Bei besonders großen Modellen kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters Optimization auf einen Wert ungleich *'none'* zu setzen. Falls Optimization = *'none'*, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter Optimization reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei find_shape_model notwendig werden, den Parameter Greediness auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen.

Falls für Optimization *'auto'* übergeben wird, wählt create_shape_model die Reduktion der Punkte automatisch.

Metric: Der Parameter Metric legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird.

Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist.

Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_shape_model erhöht sich in diesem Fall geringfügig.

Falls Metric = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Da sich in diesem Fall

die Laufzeit von find_shape_model aber wesentlich erhöht, ist es in solchen Fällen meist sinnvoller, mehrere Modelle mit create_shape_model zu erzeugen und mit find_shape_models zu suchen.

Die obigen drei Metriken können nur auf einkanalige Bilder angewendet werden. Falls im Modellbild oder in den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes verwendet.

Falls Metric = *'ignore_color_polarity'*, wird das Modell auch dann gefunden, falls sich die Farbkontraste lokal ändern können. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün. Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von find_shape_model erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für Metric = *'ignore_color_polarity'* die Anzahl der Kanäle bei der Modellgenerierung mit create_shape_model und bei der Suche mit find_shape_model verschieden sein können. Dies erlaubt z.B. die Modellgenerierung aus einem einkanaligen synthetisch generierten Bild. Es ist auch zu beachten, dass die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein.

Contrast: Der Parameter Contrast legt fest, welchen Grauwertkontrast die Punkte des Modells besitzen müssen. Der Kontrast ist ein Maß für die lokalen Grauwertdifferenzen zwischen dem Objekt und dem Hintergrund und zwischen verschiedenen Teilen des Objektes. Contrast sollte so gewählt werden, dass nur die signifikanten Merkmale des Musters für das Modell verwendet werden. Contrast kann auch ein Tupel mit zwei Werten enthalten. In diesem Fall wird bei der Segmentierung des Modells ein Verfahren, das ähnlich zum Hysterese-Schwellenwert-Verfahren in edges_image ist, eingesetzt. Dabei ist der erste Wert des Tupels der untere Schwellenwert und der zweite Wert der obere Schwellenwert. Für weitere Informationen über das Hysterese-Schwellenwert-Verfahren siehe hysteresis_threshold. Optional kann in Contrast auch ein dritter Wert als letztes Element des Tupels übergeben werden. Dieser Wert dient zur Selektion von signifikanten Komponenten des Modells basierend auf der Größe der Komponenten, d.h. Komponenten, deren Anzahl von Punkten kleiner ist als diese Mindestgröße, werden unterdrückt. Da sich die Mindestgröße auf die Größe der Komponenten bezieht kann es auch vorkommen, dass die daraus abgeleiteten Modellkonturen trotzdem kleiner als die angegebene Mindestgröße sind. Dieser Schwellenwert wird pro Pyramidenstufe halbiert. Falls eine Unterdrückung von kleinen Modellkomponenten gewünscht ist, aber keine Hysterese-Schwellenwert-Operation, müssen dennoch drei Werte in Contrast übergeben werden, von denen die ersten zwei auf denselben Wert gesetzt werden. Die Wirkung von Contrast kann auch vorab mit inspect_shape_model überprüft werden.

Wird Contrast auf *'auto'* gesetzt, dann bestimmt create_shape_model die beschriebenen drei Werte automatisch. Es besteht die Möglichkeit, nur den Kontrast (*'auto_contrast'*), die Hysterese-Schwellenwerte (*'auto_contrast_hyst'*) oder die Mindestgröße (*'auto_min_size'*) automatisch bestimmen zu lassen. Die übrigen, nicht automatisch zu bestimmenden Größen, können dann zusätzlich als Tupel übergeben werden. Dabei sind auch Kombinationen zulässig: Wird z.B. *['auto_contrast','auto_min_size']* übergeben, wird sowohl der Kontrast als auch die Mindestgröße bestimmt. Bei *['auto_min_size',20,30]* wird die Mindestgröße automatisch ermittelt, während die Hysterese-Schwellen auf

20 und *30* gesetzt werden, etc. In bestimmten Fällen kann es vorkommen, dass die automatische Bestimmung der Kontrast-Schwellenwerte nicht zum gewünschten Ergebnis führt. So ist eine manuelle Parameterwahl zu bevorzugen, wenn anwendungsbedingt bestimmte Modellkomponenten ausgeschlossen oder ins Modell integriert werden sollen oder wenn das Objekt mehrere unterschiedliche Kontraststufen umfasst. Daher ist es sinnvoll, die Kontrastwerte vor der Modellgenerierung mit determine_shape_model_params automatisch zu ermitteln und das Ergebnis mit inspect_shape_model zu überprüfen. Es ist zu beachten, dass MinContrast die Bestimmung der Kontrast-Schwellenwerte beeinflusst, und daher auch die Schätzung der Mindestgröße.

MinContrast: Mit MinContrast wird festgelegt, welchen Grauwertkontrast das Modell später bei der Erkennung mit find_shape_model im Bild mindestens besitzen muss. Mit anderen Worten stellt dieser Parameter somit eine Abgrenzung des Musters von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte MinContrast auf 10 gesetzt werden. Falls mehrkanalige Bilder für das Modell und für die Suchbilder verwendet werden, und falls der Parameter Metric auf *'ignore_color_polarity'* gesetzt wird (siehe oben), muss das Rauschen in einem Kanal noch mit der Wurzel der Anzahl der Kanäle multipliziert werden, um MinContrast zu bestimmen. Falls die Grauwerte in einem Kanal z.B. in einem Bereich von 10 Graustufen schwanken, sollte MinContrast bei

einem dreikanaligen Bild auf 17 gesetzt werden. Offensichtlich muss MinContrast kleiner als Contrast sein. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss MinContrast entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte MinContrast etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells zu gewährleisten.

Wird MinContrast auf *'auto'* gesetzt, wählt create_shape_model den Mindestkontrast basierend auf dem Rauschen im Modellbild automatisch. Eine automatische Bestimmung ergibt demzufolge nur Sinn, wenn das bei der Erkennung zu erwartende Bildrauschen dem Rauschen im Modellbild entspricht. Außerdem kann es in manchen Fällen sinnvoll sein, den automatisch ermittelten Wert zu erhöhen, um eine höhere Robustheit gegenüber Verdeckungen zu erzielen (siehe oben). Der automatisch bestimmte Mindestkontrast kann mit get_shape_model_params abgefragt werden.

Vollständige Vorabgenerierung des Modells

Optional kann in Optimization ein zweiter Wert übergeben werden. Dieser Wert legt fest, ob das Modell vollständig vorab generiert wird oder nicht. Der zweite Wert von Optimization ist hierzu auf *'pregeneration'* oder *'no_pregeneration'* zu setzen. Falls der zweite Wert nicht angegeben wird, wird der mit set_system ('pregenerate_shape_models',...) eingestellte Modus verwendet. Mit der Standardeinstellung (*'pregenerate_shape_models'* = *'false'*) wird das Modell nicht vollständig vorab generiert. Die vollständige Vorabgenerierung des Modells führt im Normalfall zu leicht geringeren Laufzeiten in find_shape_model, da das Modell nicht zur Laufzeit transformiert werden muss. Allerdings sind in diesem Fall der Speicherbedarf und die Zeit, die zur Modellerzeugung benötigt wird, wesentlich höher. Zu beachten ist auch, dass in den zwei Modi nicht dieselben Ergebnisse erwartet werden können, da intern bei der Transformation zur Laufzeit notwendigerweise andere Daten für die transformierten Modelle entstehen, als bei der Vorabgenerierung. Falls das Modell nicht vorab generiert wird, werden z.B. in find_shape_model typischerweise leicht niedrigere Bewertungen (Scores) zurückgeliefert, so dass es unter Umständen notwendig sein kann, einen niedrigeren Wert für MinScore als bei vollständiger Vorabgenerierung zu verwenden. Außerdem können sich die durch Interpolation gewonnenen Subpixelpositionen geringfügig unterscheiden. Falls höchste Genauigkeit gewünscht wird, sollte die Lage des Modells mittels Ausgleichsrechnung (Least-Squares Adjustment) bestimmt werden.

Falls eine vollständige Vorabgenerierung des Modells gewählt wird, wird das Modell in dem gewählten Winkelbereich vorab generiert und im Speicher abgelegt. Der Speicherbedarf zur Speicherung des Modells ist also proportional zur Anzahl der Winkelschritte und zur Anzahl der Punkte im Modell. Wenn also AngleStep zu klein bzw. AngleExtent zu groß gewählt wird, kann es vorkommen, dass das Modell nicht mehr in den (virtuellen) Speicher passt. In diesem Fall muss entweder AngleStep größer oder AngleExtent kleiner gewählt werden. In jedem Fall ist es aus Laufzeitgründen vorteilhaft, wenn das Modell komplett in den Hauptspeicher passt und somit ein Paging durch das Betriebssystem vermieden werden kann. Da die Möglichkeit zur subpixelgenauen Winkelbestimmung in find_shape_model gegeben ist, kann bei Modellen mit einem Durchmesser von ca. 200 Pixeln AngleStep ≥ 1 gewählt werden.

Falls AngleStep = *'auto'* (oder *0* für Rückwärtskompatibilität) übergeben wird, wählt create_shape_model automatisch eine Schrittweite basierend auf der Größe des Modells aus. Die automatisch gewählte Schrittweite kann mit get_shape_model_params abgefragt werden.

Falls eine vollständige Vorabgenerierung des Modells nicht gewählt wird, wird das Modell nur in einer Referenzlage auf jeder Ebene erzeugt. In diesem Fall muss das Modell zur Laufzeit in find_shape_model auf die verschiedenen Winkel transformiert werden. Dadurch kann es vorkommen, dass das Auffinden des Modells etwas mehr Zeit benötigt.

Es ist zu beachten, dass vorabgenerierte Modelle für eine spezifische Bildgröße bestimmt sind. Aus Laufzeitgründen ist die Verwendung von Bildern mit verschiedenen Größen während der Suche mit dem gleichen Modell nicht parallel möglich. In diesem Fall müssen Kopien des gleichen Modells verwendet werden. Anderenfalls kann das Programm abstürzen.

Parameter

▷ **Template** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: NumLevels ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Musters.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: `AngleStep` ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep >= 0 && AngleStep <= pi / 16

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung und optional Methode der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: `Optimization` ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high', 'pregeneration', 'no_pregeneration'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **Contrast** (input_control) . . . . . number(-array) ⇝ *integer* / string
Schwellenwert bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Musterbild und optional Mindestgröße der Objektteile.
Defaultwert: 'auto'
Wertevorschläge: `Contrast` ∈ {'auto', 'auto_contrast', 'auto_contrast_hyst', 'auto_min_size', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer* / string
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: `MinContrast` ∈ {'auto', 1, 2, 3, 5, 7, 10, 20, 30, 40}
Restriktion: MinContrast < Contrast

▷ **ModelID** (output_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_shape_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn die Parameter `NumLevels` und `Contrast` so gewählt worden sind, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`draw_region`, `reduce_domain`, `threshold`

Nachfolger

`find_shape_model`, `find_shape_models`, `get_shape_model_params`, `clear_shape_model`, `write_shape_model`, `set_shape_model_origin`, `set_shape_model_clutter`

Alternativen

create_scaled_shape_model, create_aniso_shape_model

Siehe auch

set_system, get_system

Modul

Matching

create_shape_model_xld (Contours : : NumLevels, AngleStart, AngleExtent, AngleStep, Optimization, Metric, MinContrast : ModelID)

Vorbereiten eines Formmodells für das Matching aus XLD-Konturen.

create_shape_model_xld erzeugt ein Formmodell für das Matching aus den XLD Konturen Contours. Die XLD Konturen repräsentieren dabei die Grauwertkanten des zu suchenden Objekts im Bild. Im Gegensatz zu dem Operator create_shape_model, der ein Formmodell aus einem Bild erzeugt, erzeugt der Operator create_shape_model_xld das Formmodell aus XLD Konturen, d.h., ohne die Verwendung eines Bildes.

Der Ausgabeparameter ModelID ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von find_shape_model verwendet wird. Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt des kleinsten umschließenden Rechtecks der Konturen Contours, das parallel zu den Koordinatenachsen liegt, verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit set_shape_model_origin gesetzt werden. Das Modell wird für mehrere Pyramidenebenen generiert und im Speicher abgelegt. Falls eine vollständige Vorabgenerierung des Modells gewählt wird (siehe unten), wird das Modell in mehreren Rotationen auf jeder Ebene generiert. Das Modell kann mit set_shape_model_clutter um Störparameter erweitert werden.

Eingabeparameter im Detail

NumLevels: Die Anzahl der Pyramidenebenen wird mit NumLevels festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von NumLevels ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens vier). Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entstehen würde, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert create_shape_model_xld eine Fehlermeldung zurück.

Falls für NumLevels *'auto'* übergeben wird, wählt create_shape_model_xld die Anzahl der Pyramidenstufen automatisch. Die gewählte Anzahl von Pyramidenstufen kann mit get_shape_model_params abgefragt werden. In seltenen Fällen kann es vorkommen, dass create_shape_model_xld die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für MinScore oder Greediness in find_shape_model selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in find_shape_model kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen manuell gewählt werden.

AngleStart, AngleExtent und AngleStep: Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Objekts im Suchbild fest. Das Objekt kann also mit find_shape_model nur in diesem Winkelbereich gefunden werden. Der Parameter AngleStep gibt die Schrittweite der Winkel in dem gewählten Winkelbereich an. Falls bei find_shape_model keine Subpixelgenauigkeit spezifiziert wird, gibt AngleStep also die erreichbare Winkelgenauigkeit an. AngleStep sollte aufgrund der Größe des Objektes gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte AngleStep für kleinere Modelle größer gewählt werden. Falls AngleExtent kein ganzzahliges Vielfaches von AngleStep ist, wird AngleStep entsprechend angepasst. Um sicherzustellen, dass find_shape_model für Modellinstanzen ohne Rotation Winkel von exakt 0.0 zurückgibt, wird der Winkelbereich der möglichen Rotationen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den AngleStart plus n mal AngleStep genau Null ergibt, wird AngleStart um höchstens AngleStep verringert, und AngleExtent wird um AngleStep vergrößert.

Optimization: Bei besonders großen Modellen kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters `Optimization` auf einen Wert ungleich *'none'* zu setzen. Falls `Optimization` = *'none'*, werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter `Optimization` reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei `find_shape_model` notwendig werden, den Parameter `Greediness` auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen. Falls für `Optimization` *'auto'* übergeben wird, wählt `create_shape_model_xld` die Reduktion der Punkte automatisch.

Metric: Der Parameter `Metric` legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird.

Falls `Metric` = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist.

Falls `Metric` = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von `find_shape_model` erhöht sich in diesem Fall geringfügig.

Falls `Metric` = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Da sich in diesem Fall die Laufzeit von `find_shape_model` aber wesentlich erhöht, ist es in solchen Fällen meist sinnvoller, mehrere Modelle mit `create_shape_model_xld` zu erzeugen und mit `find_shape_models` zu suchen.

Die obigen drei Metriken können nur auf einkanalige Bilder angewendet werden. Falls im Modellbild oder in den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes verwendet.

Falls `Metric` = *'ignore_color_polarity'*, wird das Modell auch dann gefunden, falls sich die Farbkontraste lokal ändern können. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün. Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von `find_shape_model` erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für `Metric` = *'ignore_color_polarity'* die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein.

Es ist auch zu beachten, dass die beiden ersten Werte für `Metric` (*'use_polarity'* und *'ignore_global_polarity'*) nur dann verwendet werden können, wenn bei allen Konturen, die in `Contours` übergeben wurden, das Attribut *'edge_direction'* gesetzt ist, da es die Polarität der Kanten festlegt. Für weitere Informationen zu Konturattributen wie *'edge_direction'* siehe `get_contour_attrib_xld`. Ansonsten besteht die Möglichkeit, die beiden Werte für `Metric` mit dem Operator `set_shape_model_metric` zu setzen, der die Polarität der Kanten aus einem Bild bestimmt.

Falls `Metric` = *'ignore_local_polarity'*, wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Da sich in diesem Fall die Laufzeit von `find_shape_model` aber wesentlich erhöht, ist es in solchen Fällen meist sinnvoller, mehrere Modelle mit `create_shape_model_xld` zu erzeugen und mit `find_shape_models` zu suchen.

Die obigen drei Metriken können nur auf einkanalige Bilder angewendet werden. Falls im Modellbild oder in den Suchbildern ein mehrkanaliges Bild übergeben wird, wird stillschweigend nur der erste Kanal des Bildes verwendet.

Falls `Metric` = *'ignore_color_polarity'*, wird das Modell auch dann gefunden, falls sich die Farbkontraste lokal ändern können. Dies ist z.B. der Fall, wenn Teile des Objektes ihre Farbe ändern können, z.B. von rot auf grün. Dieser Modus ist insbesondere dann sinnvoll, wenn nicht vorab bekannt ist, in welchen Kanälen das Objekt zu erkennen ist. Auch hier kann sich die Laufzeit von `find_shape_model` erheblich erhöhen. Die Metrik *'ignore_color_polarity'* kann auf Bilder mit einer beliebigen Anzahl von Kanälen angewendet werden. Falls sie mit einkanaligen Bildern verwendet wird, hat sie dieselbe Wirkung wie *'ignore_local_polarity'*. Es ist zu beachten, dass für `Metric` = *'ignore_color_polarity'* die Kanäle nicht einer spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Die Kanäle können z.B. auch durch Beleuchtung des Objektes aus unterschiedlichen Richtungen entstanden sein.

MinContrast: Mit MinContrast wird festgelegt, welchen Grauwertkontrast die Objektkanten später bei der Erkennung mit find_shape_model im Bild mindestens besitzen müssen. Mit anderen Worten stellt dieser Parameter somit eine Abgrenzung des Objekts von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte MinContrast auf 10 gesetzt werden. Falls mehrkanalige Bilder für das Modell und für die Suchbilder verwendet werden, und falls der Parameter Metric auf *'ignore_color_polarity'* gesetzt wird (siehe oben), muss das Rauschen in einem Kanal noch mit der Wurzel der Anzahl der Kanäle multipliziert werden, um MinContrast zu bestimmen. Falls die Grauwerte in einem Kanal z.B. in einem Bereich von 10 Graustufen schwanken, sollte MinContrast bei einem dreikanaligen Bild auf 17 gesetzt werden. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muss MinContrast entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte MinContrast etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells zu gewährleisten.

Vollständige Vorabgenerierung des Modells

Optional kann in Optimization ein zweiter Wert übergeben werden. Dieser Wert legt fest, ob das Modell vollständig vorab generiert wird oder nicht. Der zweite Wert von Optimization ist hierzu auf *'pregeneration'* oder *'no_pregeneration'* zu setzen. Falls der zweite Wert nicht angegeben wird, wird der mit set_system ('pregenerate_shape_models',...) eingestellte Modus verwendet. Mit der Standardeinstellung (*'pregenerate_shape_models'* = *'false'*) wird das Modell nicht vollständig vorab generiert. Die vollständige Vorabgenerierung des Modells führt im Normalfall zu leicht geringeren Laufzeiten in find_shape_model, da das Modell nicht zur Laufzeit transformiert werden muss. Allerdings sind in diesem Fall der Speicherbedarf und die Zeit, die zur Modellerzeugung benötigt wird, wesentlich höher. Zu beachten ist auch, dass in den zwei Modi nicht dieselben Ergebnisse erwartet werden können, da intern bei der Transformation zur Laufzeit notwendigerweise andere Daten für die transformierten Modelle entstehen, als bei der Vorabgenerierung. Falls das Modell nicht vorab generiert wird, werden z.B. in find_shape_model typischerweise leicht niedrigere Bewertungen (Scores) zurückgeliefert, so dass es unter Umständen notwendig sein kann, einen niedrigeren Wert für MinScore als bei vollständiger Vorabgenerierung zu verwenden. Außerdem können sich die durch Interpolation gewonnenen Subpixelpositionen geringfügig unterscheiden. Falls höchste Genauigkeit gewünscht wird, sollte die Lage des Modells mittels Ausgleichsrechnung (Least-Squares Adjustment) bestimmt werden.

Falls eine vollständige Vorabgenerierung des Modells gewählt wird (siehe unten), wird das Modell in dem gewählten Winkelbereich vorab generiert und im Speicher abgelegt. Der Speicherbedarf zur Speicherung des Modells ist also proportional zur Anzahl der Winkelschritte und zur Anzahl der Punkte im Modell. Wenn also AngleStep zu klein bzw. AngleExtent zu groß gewählt wird, kann es vorkommen, dass das Modell nicht mehr in den (virtuellen) Speicher passt. In diesem Fall muss entweder AngleStep größer oder AngleExtent kleiner gewählt werden. In jedem Fall ist es aus Laufzeitgründen vorteilhaft, wenn das Modell komplett in den Hauptspeicher passt und somit ein Paging durch das Betriebssystem vermieden werden kann. Da die Möglichkeit zur subpixelgenauen Winkelbestimmung in find_shape_model gegeben ist, kann bei Modellen mit einem Durchmesser von ca. 200 Pixeln AngleStep ≥ 1 gewählt werden.

Falls AngleStep $=$ *'auto'* übergeben wird, wählt create_shape_model_xld automatisch eine Schrittweite basierend auf der Größe des Modells aus. Die automatisch gewählte Schrittweite kann mit get_shape_model_params abgefragt werden.

Falls eine vollständige Vorabgenerierung des Modells nicht gewählt wird, wird das Modell nur in einer Referenzlage auf jeder Ebene erzeugt. In diesem Fall muss das Modell zur Laufzeit in find_shape_model auf die verschiedenen Winkel transformiert werden. Dadurch kann es vorkommen, dass das Auffinden des Modells etwas mehr Zeit benötigt.

Es ist zu beachten, dass vorabgenerierte Modelle für eine spezifische Bildgröße bestimmt sind. Aus Laufzeitgründen ist die Verwendung von Bildern mit verschiedenen Größen während der Suche mit dem gleichen Modell nicht parallel möglich. In diesem Fall müssen Kopien des gleichen Modells verwendet werden. Anderenfalls kann das Programm abstürzen.

Achtung

Im Gegensatz zum Operator create_shape_model ist es nicht möglich die Mindestgröße der Modellkomponenten anzugeben. Um kleine Modellkomponenten im Formmodell zu vermeiden, können kurze Konturen vor dem Aufruf von create_shape_model_xld mit dem Operator select_contours_xld eliminiert werden.

Parameter

▷ **Contours** (input_object) . . . . . xld_cont(-array) ⇝ *object*
Eingabekonturen, die zum Aufbau des Modells verwendet werden.

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: NumLevels ∈ {'auto', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Musters.
Defaultwert: -0.39
Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: AngleExtent ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: AngleStep ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep > 0 && AngleStep <= pi / 16

▷ **Optimization** (input_control) . . . . . string(-array) ⇝ *string*
Art der Optimierung und optional Methode der Modellgenerierung.
Defaultwert: 'auto'
Werteliste: Optimization ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high', 'pregeneration', 'no_pregeneration'}

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'ignore_local_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **MinContrast** (input_control) . . . . . number ⇝ *integer*
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 5
Wertevorschläge: MinContrast ∈ {1, 2, 3, 5, 7, 10, 20, 30, 40}

▷ **ModelID** (output_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_shape_model_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter NumLevels so gewählt wurde, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

read_contour_xld_dxf, edges_sub_pix, select_contours_xld

Nachfolger

find_shape_model, find_shape_models, get_shape_model_params, clear_shape_model, write_shape_model, set_shape_model_origin, set_shape_model_param, set_shape_model_metric, set_shape_model_clutter

Alternativen

create_scaled_shape_model_xld, create_aniso_shape_model_xld

Siehe auch

set_system, get_system

Modul

Matching

deserialize_shape_model (: : SerializedItemHandle : ModelID)

Deserialisieren eines Formmodells.

deserialize_shape_model deserialisiert ein Formmodell, welches mit serialize_shape_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Formmodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Formmodell mit dem Handle ModelID gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **ModelID** (output_control) ..... shape_model ⇝ *handle*
  Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_shape_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_shape_model

Nachfolger

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models, get_shape_model_clutter

Siehe auch

create_shape_model, create_scaled_shape_model, create_aniso_shape_model, clear_shape_model

Modul

Matching

determine_shape_model_params (Template : : NumLevels, AngleStart, AngleExtent, ScaleMin, ScaleMax, Optimization, Metric, Contrast, MinContrast, Parameters : ParameterName, ParameterValue)

Berechnung der Parameter eines Formmodells.

determine_shape_model_params berechnet bestimmte Parameter eines Formmodells automatisch aus dem Modellbild Template. Die zu berechnenden Parameter werden in Parameters angegeben. Mit determine_shape_model_params lassen sich dieselben Parameter automatisch bestimmen, die auch mit create_shape_model, create_scaled_shape_model und create_aniso_shape_model durch

Setzen des jeweiligen Parameters auf *'auto'* automatisch bestimmt werden können: die Anzahl der Pyramidenebenen des Modells (Parameters = *'num_levels'*), die Winkelschrittweite (Parameters = *'angle_step'*), die Skalierungsschrittweite (Parameters = *'scale_step'* für isotrope Skalierungen bzw. *'scale_r_step'* und/oder *'scale_c_step'* für anisotrope Skalierungen), die Art der Optimierung (Parameters = *'optimization'*), der Schwellenwert (Parameters = *'contrast'*) bzw. die Hystereseschwellenwerte (Parameters = *'contrast_hyst'*) für den Kontrast, die Mindestgröße der Objektteile (Parameters = *'min_size'*) und der minimale Kontrast (Parameters = *'min_contrast'*). Es kann eine beliebige Kombination dieser Parameter bestimmt werden, indem ein Tupel von Werten in Parameters übergeben wird. Falls alle Parameter automatisch bestimmt werden sollen, kann zur Vereinfachung auch der Wert *'all'* übergeben werden. In diesem Fall werden beide Hystereseschwellenwerte bestimmt, d.h. der Operator verhält sich so, als ob *'contrast_hyst'* anstatt *'contrast'* übergeben worden wäre.

determine_shape_model_params kann hauptsächlich dazu verwendet werden, die obigen Parameter automatisch zu bestimmen, bevor das Modell erzeugt wird. Dies ist vor allem in interaktiven Systemen nützlich, in denen dem Benutzer ein Vorschlag für diese Werte präsentiert wird, die der Benutzer aber modifizieren kann.

Die berechneten Parameter werden in ParameterName und ParameterValue als Name des Parameters und als jeweiliger Wert zurückgeliefert. Dabei haben die Parameter in ParameterName dieselben Namen wie in Parameters, wobei natürlich der Wert *'all'* durch die entsprechenden Parameternamen ersetzt wird. Eine Ausnahme besteht jedoch für den Parameter *'contrast_hyst'*, für den zwei Werte *'contrast_low'* und *'contrast_high'* zurückgegeben werden.

Die restlichen Parameter (NumLevels, AngleStart, AngleExtent, ScaleMin, ScaleMax, Optimization, Metric, Contrast und MinContrast) haben dieselbe Bedeutung wie bei create_shape_model, create_scaled_shape_model und create_aniso_shape_model. Die Beschreibung der Parameter kann dort nachgelesen werden. Diese Parameter werden von determine_shape_model_params dazu verwendet, die zu bestimmenden Parameter auf dieselbe Art zu berechnen wie in create_shape_model, create_scaled_shape_model und create_aniso_shape_model. Es ist zu beachten, dass in ScaleMin und ScaleMax jeweils ein Wert übergeben werden muss, falls die Parameter eines Formmodells mit isotroper Skalierung bestimmt werden sollen, falls also Parameters *'scale_step'* explizit oder implizit über *'all'* enthält. Falls die Parameter eines Formmodells mit anisotroper Skalierung bestimmt werden sollen, d.h. falls Parameters *'scale_r_step'* oder *'scale_c_step'* explizit oder implizit enthält, müssen in ScaleMin und ScaleMax jeweils zwei Werte angegeben werden. Dabei bezieht sich der jeweils erste Parameter auf die Skalierung in Zeilenrichtung, der jeweils zweite Parameter auf die Skalierung in Spaltenrichtung.

Zu beachten ist, dass determine_shape_model_params einige Parameter enthält, die auch automatisch bestimmt werden können (NumLevels, Optimization, Contrast, MinContrast). Falls diese Parameter nicht automatisch bestimmt werden sollen, d.h. ihr Name wird nicht in Parameters übergeben, müssen für die entsprechenden Parameter zulässige Werte übergeben werden. Insbesondere dürfen die entsprechenden Parameter nicht auf *'auto'* gesetzt sein. Sollen diese Parameter dagegen automatisch bestimmt werden, so werden die in den entsprechenden Parametern übergebenen Werte wie folgt behandelt: Sollen die Art der Optimierung oder die (Hysterese)-Schwellenwerte für den Kontrast automatisch bestimmt werden, d.h. Parameters enthält die Werte *'optimization'* oder *'contrast'* (*'contrast_hyst'*), dann werden die in Optimization bzw. Contrast übergebenen Werte ignoriert. Sollen dagegen die Anzahl an Pyramidenebenen oder der minimale Kontrast automatisch bestimmt werden, d.h. Parameters enthält die Werte *'num_levels'* oder *'min_contrast'*, dann kann HALCON zwar geeignete Werte ermitteln, zusätzlich besteht jedoch die Möglichkeit, eine obere bzw. untere Grenze anzugeben:

Falls die maximale Anzahl der Pyramidenstufen vorgegeben werden soll, kann NumLevels auf den betreffenden Wert gesetzt werden. Falls in diesem Fall Parameters den Wert *'num_levels'* enthält, kann der berechnete Wert der Anzahl der Pyramidenebenen höchstens NumLevels betragen. Falls NumLevels = *'auto'* (oder *0* für Rückwärtskompatibilität) gesetzt wird, wird die Anzahl der Pyramidenebenen ohne Beschränkung so hoch wie möglich bestimmt.

Falls der minimale Kontrast vorgegeben werden soll, kann MinContrast auf den betreffenden Wert gesetzt werden. Falls in diesem Fall Parameters den Wert *'min_contrast'* enthält, beträgt der berechnete Wert des minimalen Kontrastes mindestens MinContrast. Falls MinContrast = *'auto'* gesetzt wird, wird der minimale Kontrast ohne Beschränkung bestimmt.

Achtung

In manchen Fällen ist der Wert für die maximale Anzahl an Pyramidenebenen, der von determine_shape_model_params zurückgegebenen wird, höher als der tatsächlich im Formmodell

verwendete Wert. Letzterer kann mit `get_shape_model_params` nach der Modellgenerierung abgefragt werden. Dieser Fall kann auftreten, wenn das Modell mit `create_scaled_shape_model` oder `create_aniso_shape_model` generiert wird und ScaleMin, ScaleRMin oder ScaleCMin auf einen Wert unter *1.0* gesetzt wird.

Parameter

▷ **`Template`** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, dessen Definitionsbereich für das Modell verwendet wird.

▷ **`NumLevels`** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: `NumLevels` ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **`AngleStart`** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Modells.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **`AngleExtent`** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **`ScaleMin`** (input_control) . . . . . number(-array) ⇝ *real*
Kleinste auftretende Skalierung des Modells.
Defaultwert: 0.9
Wertevorschläge: `ScaleMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleMin > 0

▷ **`ScaleMax`** (input_control) . . . . . number(-array) ⇝ *real*
Größte auftretende Skalierung des Modells.
Defaultwert: 1.1
Wertevorschläge: `ScaleMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleMax >= ScaleMin

▷ **`Optimization`** (input_control) . . . . . string ⇝ *string*
Art der Optimierung.
Defaultwert: 'auto'
Werteliste: `Optimization` ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **`Metric`** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **`Contrast`** (input_control) . . . . . number(-array) ⇝ *integer* / string
Schwellenwert bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Musterbild und optional Mindestgröße der Objektteile.
Defaultwert: 'auto'
Wertevorschläge: `Contrast` ∈ {'auto', 'auto_contrast', 'auto_contrast_hyst', 'auto_min_size', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

▷ **`MinContrast`** (input_control) . . . . . integer ⇝ *integer* / string
Minimaler Kontrast des Objektes in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: `MinContrast` ∈ {'auto', 1, 2, 3, 5, 7, 10, 20, 30, 40}
Restriktion: MinContrast < Contrast

▷ **`Parameters`** (input_control) . . . . . string(-array) ⇝ *string*
Zu berechnende Parameter.
Defaultwert: 'all'
Werteliste: `Parameters` ∈ {'all', 'num_levels', 'angle_step', 'scale_step', 'scale_r_step', 'scale_c_step', 'optimization', 'contrast', 'contrast_hyst', 'min_size', 'min_contrast'}

▷ **ParameterName** (output_control) . . . . . string-array ⇝ *string*
Name des berechneten Parameters.

▷ **ParameterValue** (output_control) . . . . . number-array ⇝ *real* / integer
Wert des berechneten Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `determine_shape_model_params` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn die Parameter `NumLevels` und `Contrast` so gewählt worden sind, dass das Modell zu wenige Punkte besitzt, oder nicht genügend signifikante Merkmale im Eingabebild vorhanden sind, wird die Fehlermeldung 8510 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`draw_region`, `reduce_domain`, `threshold`

Nachfolger

`create_shape_model`, `create_scaled_shape_model`, `create_aniso_shape_model`

Siehe auch

`find_shape_model`, `find_scaled_shape_model`, `find_aniso_shape_model`, `find_shape_models`, `find_scaled_shape_models`, `find_aniso_shape_models`

Modul

Matching

find_aniso_shape_model (Image : : ModelID, AngleStart, AngleExtent, ScaleRMin, ScaleRMax, ScaleCMin, ScaleCMax, MinScore, NumMatches, MaxOverlap, SubPixel, NumLevels, Greediness : Row, Column, Angle, ScaleR, ScaleC, Score)

Suche der besten Matches eines anisotrop skalierten Formmodells in einem Bild.

`find_aniso_shape_model` findet die besten `NumMatches` Instanzen des anisotrop skalierten Formmodells `ModelID` im Eingabebild `Image`. Das Modell muss zuvor mit `create_aniso_shape_model` erzeugt oder mit `read_shape_model` eingelesen worden sein.

Die Position, Rotation und Skalierung in Zeilen- und Spaltenrichtung der gefundenen Instanzen des Modells werden in `Row`, `Column`, `Angle`, `ScaleR` und `ScaleC` zurückgeliefert. In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Nähere Informationen finden Sie in den parameterspezifischen Abschnitten.

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes `Image` gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit `create_aniso_shape_model` verwendet wurde. Ein eventuell mit `set_shape_model_origin` anders gesetzter Ursprung wird nicht berücksichtigt. Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als `MinScore` erreichen würde (siehe unten). Außerdem kann es sein, dass das Modell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit `set_system('border_shape_models','true')` für alle Modelle oder mit `set_shape_model_param(ModelID,'border_shape_models','true')` speziell für ein Modell umgestellt werden, so dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als `MinScore` erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der

Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit `dilation_circle` zu vergrößern.

AngleStart, AngleExtent, ScaleRMin, ScaleRMax, ScaleCMin, ScaleCMax: Die Parameter `AngleStart` und `AngleExtent` legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Die Parameter `ScaleRMin`, `ScaleRMax`, `ScaleCMin` und `ScaleCMax` legen den Bereich der Skalierungen in Zeilen- und Spaltenrichtung fest, in dem nach dem Modell gesucht wird. Der Winkel- und Skalierungsbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit `create_aniso_shape_model` angegeben worden ist. Dies bedeutet insbesondere, dass die Winkelbereiche des Modells und der Suche sich überlappen müssen.

Wird in `ModelID` ein Modell übergeben, das mit `create_shape_model` oder mit `create_scaled_shape_model` erzeugt wurde, wird das Modell mit einer isotropen Skalierung von 1.0 bzw. mit einer isotropen Skalierung zwischen `ScaleRMin` und `ScaleRMax` gesucht. In diesem Fall werden in `ScaleR` und `ScaleC` identische Werte zurückgeliefert.

Beachten Sie, dass in einigen Fällen Instanzen mit einer Rotation oder Skalierung gefunden werden, die geringfügig außerhalb des angegebenen Winkel- und Skalierungsbereichs liegen. Dies kann vorkommen, wenn der angegebene Bereich kleiner ist als der bei der Erstellung des Modells angegebene Bereich. `AngleStart` und `AngleExtent` sowie `ScaleRMin`/`ScaleCMin` und `ScaleRMax`/`ScaleCMax` werden nur auf der höchsten Pyramidenstufe überprüft. Matches, die auf der höchsten Pyramidenstufe gefunden werden, werden bis zur untersten Pyramidenstufe verfeinert. Aus Leistungsgründen wird bei der Verfeinerung jedoch nicht mehr überprüft, ob die Matches immer noch innerhalb der angegebenen Bereiche liegen.

MinScore: Der Parameter `MinScore` legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von `MinScore` gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, dass das Modell niemals verdeckt wird, kann `MinScore` auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden. Werden die Matches nicht bis zur untersten Pyramidenebene verfolgt (siehe unten) kann es in manchen Fällen vorkommen, dass Instanzen gefunden werden, deren Score geringfügig unter dem Wert von `MinScore` liegen.

Falls das Formmodell mit `set_shape_model_clutter` um Störparameter erweitert wurde und dadurch *'use_clutter'* aktiviert wurde, erwartet `MinScore` einen zweiten Wert, welcher den maximalen Störwert eines potentiellen Matches angibt, um als Instanz des Modells im Bild betrachtet werden zu können. Die Laufzeit mit Nutzung von Störparametern ist mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter `NumMatches`.

NumMatches: Mit `NumMatches` kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als `NumMatches` Instanzen eine Bewertung größer als `MinScore` erreichen, werden nur die besten `NumMatches` Instanzen zurückgeliefert. Falls weniger als `NumMatches` Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter `MinScore` hat Vorrang vor `NumMatches`. Sollen alle Modellinstanzen, deren Bewertung `MinScore` übersteigt, im Bild gefunden werden, muss `NumMatches` auf *0* gesetzt werden.

Beim Tracken der Matches durch die Bildpyramide werden auf jeder Ebene Matches verworfen, die wenig vielversprechend sind. Dadurch ist es möglich, dass Matches verworfen werden, die auf der niedrigsten Pyramidenebene einen höheren Score gehabt hätten. Beispielsweise kann das dazu führen, dass der gefundene Match für `NumMatches` gleich *1* nicht mit dem Match mit dem höchsten Score übereinstimmt, der von `NumMatches` gleich *0* oder > *1* zurückgegeben wird.

Falls erwartet wird, dass mehrere Objekte mit einem ähnlichen Score gefunden werden können, aber nur das Modell mit dem höchsten Score zurückgegeben werden soll, kann es vorteilhaft sein, `NumMatches` zu erhöhen, und dann den Match mit dem höchsten Score auszuwählen.

Für den Fall, dass das Formmodell mit `set_shape_model_clutter` um Störparameter erweitert wurde, berücksichtigt `NumMatches` auch den zweiten Wert, der in `MinScore` übergeben wird. Gefundene Instanzen weisen in diesem Fall einen Score auf, der größer als der erste Eintrag ist, sowie einen Störwert der niedriger als der zweite Eintrag ist. Ist die Anzahl gefundener Instanzen größer als `NumMatches`, werden die besten `NumMatches` Instanzen in Bezug auf ihren Störwert zurückgegeben. `MinScore` hat weiterhin Vorrang vor `NumMatches` und `NumMatches` muss weiterhin auf *0* gesetzt werden um alle Modellinstanzen zu erhalten, die die Bedingungen von `MinScore` erfüllen. Es ist zu beachten, dass beim Tracken der Matches durch die Bildpyramide keine Matches verworfen werden, wenn Störparameter verwendet werden. Die Laufzeit mit Nutzung von Störparametern ist daher mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter `NumMatches`.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter `MaxOverlap` kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als `MaxOverlap` überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe `smallest_rectangle2`). Bei `MaxOverlap` $= 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei `MaxOverlap` $= 1$ werden alle gefundenen Instanzen zurückgeliefert.

SubPixel: Der Parameter `SubPixel` gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls `SubPixel` auf *'none'* (oder *'false'* für Rückwärtskompatibilität) gesetzt wird, wird die Lage des Modells nur pixelgenau bzw. mit der bei `create_aniso_shape_model` angegebenen Winkel- und Skalierungsauflösung bestimmt. Falls `SubPixel` auf *'interpolation'* (oder *'true'*) gesetzt wird, werden sowohl die Position als auch die Rotation und Skalierung subpixelgenau bestimmt. Dabei wird die Lage des Modells anhand der Score-Funktion interpoliert. Dieser Modus kostet fast keine Rechenzeit und liefert in den meisten Anwendungen eine ausreichende Genauigkeit. In manchen Anwendungen ist eine möglichst hohe Genauigkeit wichtig. In diesen Fällen kann die Lage des Modells durch Ausgleichsrechnung, d.h., durch Minimierung der Abstände der Modellpunkte und der zugehörigen Bildpunkte (Least-Squares Adjustment), bestimmt werden. Im Gegensatz zu *'interpolation'* kostet dieser Modus zusätzliche Rechenzeit. Mit den verschiedenen Modi (*'least_squares'*, *'least_squares_high'* und *'least_squares_very_high'*) kann dabei die Genauigkeit festgelegt werden, mit der der minimale Abstand gesucht wird. Je höher die Genauigkeit gewählt wird, desto länger dauert allerdings auch die Subpixel-Extraktion. Im Normalfall sollte für `SubPixel` *'interpolation'* gewählt werden. Falls die Ausgleichsrechnung gewünscht wird, sollte *'least_squares'* gewählt werden, da sich so der optimale Kompromiss zwischen Laufzeit und Genauigkeit ergibt.

Objekte, die im Vergleich zum Modell leicht deformiert im Bild erscheinen, können in manchen Fällen entweder nicht oder nur mit einer geringen Genauigkeit gefunden werden. Für solche Objekte besteht die Möglichkeit, im Parameter `SubPixel` zusätzlich die maximal zulässige Objektdeformation in Pixeln anzugeben. Dies kann mit Hilfe des optionalen Parameterwertes *'max_deformation '* und einer nachfolgenden Ganzzahl zwischen *0* und *32* (im selben String), die die maximale Deformation beschreibt, erfolgen. Kann die Form des Objektes z.B. bis zu 2 Pixel von der im Modell gespeicherten Form abweichen, muss im Parameter `SubPixel` der Wert *'max_deformation 2'* zusätzlich zum oben beschriebenen Modus für die subpixelgenaue Extraktion übergeben werden, also z.B. *['least_squares', 'max_deformation 2']*. Mit *'max_deformation 0'* wird das Objekt ohne Deformationen gesucht, also so als ob man kein *'max_deformation '* übergeben hätte. Zu beachten ist, dass größere Werte für die maximale Deformation oft zu erhöhten Laufzeiten führen. Außerdem besteht bei einem zu großen Wert die Gefahr, dass falsche Kandidaten gefunden werden. Beide Probleme können hauptsächlich bei kleinen Objekten oder Objekten mit feinen Strukturen auftreten. Der Grund hierfür ist, dass solche Objekte durch große erlaubte Deformationen ihre für die Suche wichtige charakteristische Form verlieren. Zu beachten ist auch, dass die Genauigkeit von teilweise verdeckten Objekten für höhere Deformationen abnehmen kann wenn Störobjekte in der Nähe des Objektes vorhanden sind. Die maximale Deformation sollte demnach immer nur so groß wie nötig und so gering wie möglich gewählt werden. Näherungsweise rotationssymmetrische Objekt könnten nicht gefunden werden, falls *'max_deformation'* und `AngleExtent` beide auf einen Wert größer 0 gesetzt sind. In dem Fall kann es zu Mehrdeutigkeiten kommen, die nicht aufgelöst werden können, sodass der Match als falsch verworfen wird. Wenn das passiert, versuchen Sie, entweder *'max_deformation'* oder `AngleExtent` auf 0 zu setzen oder das Modell so zu verändern, dass weniger Symmetrien auftreten. Die Score-Berechnung bei Deformationswerten größer *0* hängt von der Art der Subpixel-Extraktion ab. In den meisten Fällen wird sich der Score eines Matches verändern, wenn man *'least_squares'*, *'least_squares_high'* oder *'least_squares_very_high'* (siehe oben) wählt (im Vergleich zu *'none'* oder *'interpolation'*). Außerdem wird sich der Score in der Regel erhöhen wenn man bei einer der Least-Squares-Adjustment-Methoden die maximale Deformation erhöht, da dann zu den Modellpunkten mehr zugehörige Bildpunkte gefunden werden können. Für einen aussagekräftigen Score und zum Vermeiden falscher Matches wird empfohlen, das Erlauben einer Deformation mit einer Subpixel-Extraktion mit einer der Least-Squares-Adjustment-Methoden zu kombinieren.

NumLevels: Mit `NumLevels` wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit `create_aniso_shape_model` angegebenen Bereich beschnitten. Falls `NumLevels` als *0* angegeben wird, wird die mit `create_aniso_shape_model` angegebene Anzahl verwendet.

In manchen Fällen kann es sein, dass die Anzahl der Pyramidenebenen, die beispielsweise automatisch mit `create_aniso_shape_model` ermittelt wurde, zu hoch ist. Dann werden eventuell Instanzen, die eine

sehr hohe finale Bewertung gehabt hätten, bereits auf der höchsten Pyramidenebene ausgeschlossen und damit nicht gefunden. Anstatt nun MinScore auf einen sehr geringen Wert zu setzen, um alle Matches zu finden, kann der Wert von NumLevels mit get_shape_model_params ermittelt werden und dann ein etwas geringerer Wert in find_aniso_shape_model genutzt werden. Diese Herangehensweise führt oft zu besseren Ergebnissen hinsichtlich Geschwindigkeit und Robustheit.

Optional kann NumLevels einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, Laufzeit einzusparen. Allerdings ist in diesem Modus im Allgemeinen die Genauigkeit der gefundenen Lageparameter geringer als im Normalfall, in dem die Matches bis auf die unterste Pyramidenebene verfolgt werden. Falls eine hohe Genauigkeit erzielt werden soll, empfiehlt es sich also, mindestens *'least_squares'* für SubPixel zu verwenden. Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden.

In mangelhaften Eingabebildern, d.h., in Bildern die z.B. unscharf, deformiert oder verrauscht sind, kann auf der untersten Pyramidenebene oft keine Instanz des Formmodells gefunden werden, da aufgrund der mangelhaften Bildqualität nicht genügend Kanteninformation vorhanden ist, oder die Kanten zu stark deformiert sind. Auf höheren Pyramidenebenen dagegen kann die Kanteninformation noch ausreichend sein um Instanzen des Modells zu finden, wobei auch hierbei die oben beschriebenen Einschränkungen bezüglich Genauigkeit und Robustheit gelten. Die Wahl der passenden Pyramidenebene, d.h., der untersten Pyramidenebene auf der noch eine Instanz des Modells gefunden wird, hängt sowohl vom Modell als auch vom Eingabebild ab. Diese Pyramidenebene kann also von Bild zu Bild variieren. Um das Matching auf mangelhaften Bildern zu erleichtern, kann die unterste Pyramidenebene, auf der noch eine Instanz des Modells gefunden wird, während des Matchings automatisch bestimmt werden. Um diesen als 'increased tolerance mode' bezeichneten Mechanismus zu aktivieren, muss die unterste Pyramidenebene in NumLevels negativ angegeben werden. Wird z.B. NumLevels $= [4, -2]$ gesetzt, so wird das Matching auf der vierten Pyramidenebene begonnen und auf der zweituntersten Pyramidenebene beendet. Es wird also versucht, eine Instanz des Formmodells auf der Pyramidenebene 2 zu finden. Kann auf dieser Pyramidenebene keine Instanz gefunden werden, so wird die unterste Pyramidenebene ermittelt, auf der noch Instanzen des Modells gefunden wurden. Die Instanzen dieser Pyramidenebene werden dann als Ergebnis zurückgegeben.

Wenn ModelID mit adapt_shape_model_high_noise angepasst wurde, wird die geschätzte untere Pyramidenstufe per Standardeinstellung verwendet. Der Benutzer kann den geschätzten Wert dennoch überschreiben, indem er zwei Werte an NumLevels übergibt und damit explizit die untere Pyramidenstufe setzt.

Greediness: Der Parameter Greediness bestimmt, wie „gierig“ die Suche durchgeführt werden soll. Für Greediness $= 0$ wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden, immer findet, wenn die anderen Parameter passend gesetzt sind. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für Greediness $= 1$ wird eine unsicherere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, dass das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für Greediness $= 1$ wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten Fällen wird das Formmodell für Greediness $= 0.9$ immer sicher gefunden.

Ausgabeparameter im Detail

Row, Column, Angle, ScaleR, ScaleC: Die Position, Rotation und Skalierung in Zeilen- und Spaltenrichtung der gefundenen Instanzen des Modells werden in Row, Column, Angle, ScaleR und ScaleC zurückgeliefert. Die Koordinaten Row und Column beschreiben die Position des Ursprungs des Modells im Suchbild. Die Werte sind allerdings bereits geringfügig angepasst, so dass sie direkt zur Erstellung einer Transformationsmatrix verwendet werden können, die für ein Alignment oder die Visualisierung der Modellkanten benötigt wird. Diese Anpassung der Koordinaten hat mit der Art und Weise zu tun, in der HALCON ikonische Objekte transformiert, siehe hierzu affine_trans_pixel. Das Beispiel weiter unten zeigt, wie die Transformationsmatrix erzeugt wird und wie man damit das Modell an der gefundenen Position im Suchbild visualisiert.

Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_aniso_shape_model verwendet wurde. Ein anderer Ursprung kann mit set_shape_model_origin festgelegt werden.

Score: In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Falls das Formmodell mit `set_shape_model_clutter` um Störparameter erweitert wurde, gibt `Score`, nach den oben genannten Werten, auch den Störwert jeder gefundenen Instanz zurück. Falls beispielsweise die Hälfte der Störregion Störkanten aufweist entspricht der Störwert 0.5. Werden z.B. zwei Instanzen gefunden, wobei für Erstere eine Bewertung von 0.9 und für die zweite 0.8 erreicht wird und der Störwert der ersten Instanz 0.2, der zweiten Instanz 0.1 ist, wird `Score` $= [0.9, 0.8, 0.2, 0.1]$ zurückgegeben. Es ist zu beachten, dass von sämtlichen Ergebnissen des formbasierten Matchings die Störwerte am stärksten von Beleuchtungsänderungen beeinflusst werden.

Setzen eines Timeout

Mittels des Operators `set_shape_model_param` können Sie einen *'timeout'* für `find_aniso_shape_model` angeben. Wenn `find_aniso_shape_model` diesen *'timeout'* erreicht endet er ohne Ergebnis und gibt den Fehlercode 9400 (H_ERR_TIMEOUT) zurück. Abhängig von der Größe der Skalierungsintervalle, die durch `ScaleRMin`, `ScaleRMax`, `ScaleCMin` und `ScaleCMax` festgelegt werden, benötigt `find_aniso_shape_model` eine signifikante Zeitspanne um den Speicher für gecachte Transformationen freizugeben, falls das Modell nicht vorberechnet wurde. Da diese Transformationen auch freigegeben werden müssen nachdem ein Timeout auftritt, überschreitet `find_aniso_shape_model` den in *'timeout'* angegebenen Wert um diese Zeitspanne.

Darstellung der Resultate

Die Verwendung der Prozedur `dev_display_shape_matching_results` wird zur Darstellung der Resultate formbasierten Matchings stark empfohlen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild, in dem das Modell gefunden werden soll.
- ▷ **`ModelID`** (input_control) ..... shape_model ⇝ *handle*
  Handle des Modells.
- ▷ **`AngleStart`** (input_control) ..... angle.rad ⇝ *real*
  Kleinste auftretende Rotation des Modells.
  Defaultwert: -0.39
  Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **`AngleExtent`** (input_control) ..... angle.rad ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Defaultwert: 0.79
  Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
  Restriktion: AngleExtent >= 0
- ▷ **`ScaleRMin`** (input_control) ..... number ⇝ *real*
  Kleinste auftretende Skalierung des Modells in Zeilenrichtung.
  Defaultwert: 0.9
  Wertevorschläge: `ScaleRMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: ScaleRMin > 0
- ▷ **`ScaleRMax`** (input_control) ..... number ⇝ *real*
  Größte auftretende Skalierung des Modells in Zeilenrichtung.
  Defaultwert: 1.1
  Wertevorschläge: `ScaleRMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
  Restriktion: ScaleRMax >= ScaleRMin
- ▷ **`ScaleCMin`** (input_control) ..... number ⇝ *real*
  Kleinste auftretende Skalierung des Modells in Spaltenrichtung.
  Defaultwert: 0.9
  Wertevorschläge: `ScaleCMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: ScaleCMin > 0

▷ **ScaleCMax** (input_control) . . . . . number ⇝ *real*
Größte auftretende Skalierung des Modells in Spaltenrichtung.
Defaultwert: 1.1
Wertevorschläge: ScaleCMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleCMax >= ScaleCMin

▷ **MinScore** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: MinScore ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MinScore ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumMatches** (input_control) . . . . . integer ⇝ *integer*
Anzahl der zu findenden Instanzen des Modells (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: NumMatches ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Überlappung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: MaxOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MaxOverlap ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **SubPixel** (input_control) . . . . . string(-array) ⇝ *string*
Subpixelgenauigkeit falls ungleich *'none'*.
Defaultwert: 'least_squares'
Wertevorschläge: SubPixel ∈ {'none', 'interpolation', 'least_squares', 'least_squares_high', 'least_squares_very_high', 'max_deformation 1', 'max_deformation 2', 'max_deformation 3', 'max_deformation 4', 'max_deformation 5', 'max_deformation 6'}

▷ **NumLevels** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls |NumLevels| = 2).
Defaultwert: 0
Werteliste: NumLevels ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real ⇝ *real*
„Gierigkeit“ der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen“ werden).
Defaultwert: 0.9
Wertevorschläge: Greediness ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ Greediness ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate der gefundenen Instanzen des Modells.

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate der gefundenen Instanzen des Modells.

▷ **Angle** (output_control) . . . . . angle.rad-array ⇝ *real*
Rotationswinkel der gefundenen Instanzen des Modells.

▷ **ScaleR** (output_control) . . . . . number-array ⇝ *real*
Skalierung der gefundenen Instanzen des Modells in Zeilenrichtung.

▷ **ScaleC** (output_control) . . . . . number-array ⇝ *real*
Skalierung der gefundenen Instanzen des Modells in Spaltenrichtung.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Instanzen des Modells.

Beispiel

```
create_aniso_shape_model (ImageReduced, 0, rad(-15), rad(30), 0, \
                          0.9, 1.1, 0, 0.9, 1.1, 0, 'none', \
```

```
                          'use_polarity', 30, 10, ModelID)
get_shape_model_contours (ModelXLD, ModelID, 1)
find_aniso_shape_model (SearchImage, ModelID, rad(-15), rad(30), \
                        0.9, 1.1, 0.9, 1.1, 0.5, 1, 0.5, 'interpolation', \
                        0, 0, Row, Column, Angle, ScaleR, ScaleC, Score)
* Create transformation matrix
hom_mat2d_identity (HomMat2DIdentity)
hom_mat2d_scale (HomMat2DIdentity, ScaleR, ScaleC, 0, 0, HomMat2DScale)
hom_mat2d_rotate (HomMat2DScale, Angle, 0, 0, HomMat2DRotate)
hom_mat2d_translate (HomMat2DRotate, Row, Column, HomMat2DObject)
* Calculate true position of the model origin in the search image
affine_trans_pixel (HomMat2DObject, 0, 0, RowObject, ColObject)
* Display results
dev_display_shape_matching_results (ModelID, 'red', Row, Column, Angle, \
                    ScaleR, ScaleC, 0)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_aniso_shape_model` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_aniso_shape_model`, `read_shape_model`, `set_shape_model_origin`, `set_shape_model_clutter`

Nachfolger

`clear_shape_model`

Alternativen

`find_shape_model`, `find_scaled_shape_model`, `find_shape_models`, `find_scaled_shape_models`, `find_aniso_shape_models`, `find_ncc_model`, `find_ncc_models`

Siehe auch

`set_system`, `get_system`

Modul

Matching

find_aniso_shape_models (Image : : ModelIDs, AngleStart, AngleExtent, ScaleRMin, ScaleRMax, ScaleCMin, ScaleCMax, MinScore, NumMatches, MaxOverlap, SubPixel, NumLevels, Greediness : Row, Column, Angle, ScaleR, ScaleC, Score, Model)

Suche der besten Matches mehrerer anisotrop skalierter Formmodelle.

`find_aniso_shape_models` findet die besten `NumMatches` Instanzen der anisotrop skalierten Formmodelle, die in `ModelIDs` übergeben werden, im Eingabebild `Image`. Die Modelle müssen zuvor mit `create_aniso_shape_model` erzeugt oder mit `read_shape_model` eingelesen worden sein. Im Gegensatz zu `find_aniso_shape_model` kann also eine Mehrzahl von Modellen gleichzeitig im selben Bild gesucht werden.

Die Position, Rotation und Skalierung in Zeilen- und Spaltenrichtung der gefundenen Instanzen des Modells werden in Row, Column, Angle, ScaleR und ScaleC zurückgeliefert. In Score wird die Bewertung der gefundenen Instanzen zurückgegeben. Der gefundene Modelltyp wird in Model zurückgeliefert. Nähere Informationen finden Sie in den parameterspezifischen Abschnitten.

Besonderheiten der Parametersemantik

Gegenüber find_aniso_shape_model unterscheidet sich die Semantik aller Eingabeparameter etwas. Alle Eingabeparameter müssen entweder genau ein Element enthalten, oder dieselbe Anzahl von Elementen wie ModelIDs. (NumLevels kann auch zwei oder zweimal die Anzahl von Elementen wie ModelIDs enthalten; siehe unten.) Im ersten Fall wird der Wert des Eingabeparameters für alle Modelle gleich verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für das entsprechende Modell in ModelIDs verwendet. Zusätzliche Details können in den parameterspezifischen Abschnitten weiter unten gefunden werden. Ein Aufruf von find_aniso_shape_models mit mehreren Werten für ModelIDs, NumMatches und MaxOverlap entspricht von der Wirkung her also mehreren unabhängigen Aufrufen von find_aniso_shape_model mit den jeweiligen Parametern, ist allerdings wesentlich effizienter.

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes Image gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_aniso_shape_model verwendet wurde. Ein eventuell mit set_shape_model_origin anders gesetzter Ursprung wird nicht berücksichtigt. Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als MinScore erreichen würde (siehe unten). Außerdem kann es sein, dass das Modell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit set_system('border_shape_models','true') für alle Modelle oder mit set_shape_model_param(ModelID,'border_shape_models','true') speziell für ein Modell umgestellt werden, so dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als MinScore erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit dilation_circle zu vergrößern.

Wie üblich wird die Region des Eingabebildes Image als Suchbereich für den Referenzpunkt der Modelle ModelIDs verwendet. Konsistent mit der obigen Semantik kann Image also ein einzelnes Bildobjekt beinhalten oder ein Bildobjekttupel. Falls Image ein einzelnes Bildobjekt enthält, wird dessen Region als Suchbereich für alle Modelle in ModelIDs verwendet. Falls Image mehrere Bildobjekte enthält, wird die jeweilige Region als Suchbereich für das entsprechende Modell aus ModelIDs verwendet. In diesem Fall müssen die Bilder, abgesehen von ihrer Region, identisch sein. D.h. Image kann nicht in beliebiger Weise mit concat_obj aufgebaut werden, sondern muss aus demselben Bild mit add_channels oder äquivalenten Aufrufen erzeugt werden. Falls das nicht der Fall ist, wird eine Fehlermeldung zurückgeliefert.

AngleStart, AngleExtent, ScaleRMin, ScaleRMax, ScaleCMin, ScaleCMax: Die Parameter AngleStart und AngleExtent legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Die Parameter ScaleRMin, ScaleRMax, ScaleCMin und ScaleCMax legen den Bereich der Skalierungen in Zeilen- und Spaltenrichtung fest, in dem nach dem Modell gesucht wird. Der Winkel- und Skalierungsbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit create_aniso_shape_model angegeben worden ist. Dies bedeutet insbesondere, dass die Winkelbereiche des Modells und der Suche sich überlappen müssen.

Wird in ModelIDs ein Modell übergeben, das mit create_shape_model oder mit create_scaled_shape_model erzeugt wurde, wird das Modell mit einer isotropen Skalierung von 1.0 bzw. mit einer isotropen Skalierung zwischen ScaleRMin und ScaleRMax gesucht. In diesem Fall werden in ScaleR und ScaleC identische Werte zurückgeliefert.

Beachten Sie, dass in einigen Fällen Instanzen mit einer Rotation oder Skalierung gefunden werden, die geringfügig außerhalb des angegebenen Winkel- und Skalierungsbereichs liegen. Dies kann vorkommen, wenn der angegebene Bereich kleiner ist als der bei der Erstellung des Modells angegebene Bereich. AngleStart und AngleExtent sowie ScaleRMin/ScaleCMin und ScaleRMax/ScaleCMax werden nur auf der

höchsten Pyramidenstufe überprüft. Matches, die auf der höchsten Pyramidenstufe gefunden werden, werden bis zur untersten Pyramidenstufe verfeinert. Aus Leistungsgründen wird bei der Verfeinerung jedoch nicht mehr überprüft, ob die Matches immer noch innerhalb der angegebenen Bereiche liegen.

MinScore: Der Parameter `MinScore` legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von `MinScore` gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, dass das Modell niemals verdeckt wird, kann `MinScore` auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden. Werden die Matches nicht bis zur untersten Pyramidenebene verfolgt (siehe unten) kann es in manchen Fällen vorkommen, dass Instanzen gefunden werden, deren Score geringfügig unter dem Wert von `MinScore` liegen.

Falls ein einzelner Wert in `MinScore` übergeben wird, wird dieser Wert auf alle Instanzen angewandt, die mit allen Modellen gefunden werden. Falls `MinScore` mehrere Einträge enthält, werden die Werte einzeln für die entsprechenden Modelle verwendet.

Falls die Formmodelle mit `set_shape_model_clutter` um Störparameter erweitert wurden und dadurch *'use_clutter'* aktiviert wurde, wird in `MinScore` zu jedem der vorhandenen Werte für den minimalen Score ein weiterer Wert erwartet, welcher den maximalen Störwert eines potentiellen Matches angibt, um als Instanz des Modells im Bild betrachtet werden zu können. Die Laufzeit mit Nutzung von Störparametern ist mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter `NumMatches`. Die verschiedenen Formmodelle müssen den gleichen Wert für *'use_clutter'* haben.

Wird der maximale Störwert separat für jedes Modell spezifiziert, was immer dann nötig ist, wenn auch der minimale Score für jedes Modell gesetzt wird, muss in `MinScore` die doppelte Anzahl an Einträgen enthalten sein wie in `ModelIDs`. Dabei müssen der minimale Score und der maximale Störwert verschränkt in `MinScore` angegeben werden. Werden in `ModelIDs` beispielsweise zwei Modelle angegeben, wobei für Ersteres ein minimaler Score von 0.9 und für das zweite 0.8 festgelegt wird und der maximale Störwert des ersten Modells 0.1, für das zweite Modell 0.2 betragen soll, muss `MinScore` $= [0.9, 0.1, 0.8, 0.2]$ angegeben werden.

NumMatches: Mit `NumMatches` kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als `NumMatches` Instanzen eine Bewertung größer als `MinScore` erreichen, werden nur die besten `NumMatches` Instanzen zurückgeliefert. Falls weniger als `NumMatches` Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter `MinScore` hat Vorrang vor `NumMatches`. Sollen alle Modellinstanzen, deren Bewertung `MinScore` übersteigt, im Bild gefunden werden, muss `NumMatches` auf *0* gesetzt werden.

Beim Tracken der Matches durch die Bildpyramide werden auf jeder Ebene Matches verworfen, die wenig vielversprechend sind. Dadurch ist es möglich, dass Matches verworfen werden, die auf der niedrigsten Pyramidenebene einen höheren Score gehabt hätten. Beispielsweise kann das dazu führen, dass der gefundene Match für `NumMatches` gleich *1* nicht mit dem Match mit dem höchsten Score übereinstimmt, der von `NumMatches` gleich *0* oder > *1* zurückgegeben wird.

Falls erwartet wird, dass mehrere Objekte mit einem ähnlichen Score gefunden werden können, aber nur das Modell mit dem höchsten Score zurückgegeben werden soll, kann es vorteilhaft sein, `NumMatches` zu erhöhen, und dann den Match mit dem höchsten Score auszuwählen.

Für den Fall, dass das Formmodell mit `set_shape_model_clutter` um Störparameter erweitert wurde, berücksichtigt `NumMatches` auch den zweiten Wert, der in `MinScore` übergeben wird. Gefundene Instanzen weisen in diesem Fall einen Score auf, der größer als der erste Eintrag ist, sowie einen Störwert der niedriger als der zweite Eintrag ist. Ist die Anzahl gefundener Instanzen größer als `NumMatches`, werden die besten `NumMatches` Instanzen in Bezug auf ihren Störwert zurückgegeben. `MinScore` hat weiterhin Vorrang vor `NumMatches` und `NumMatches` muss weiterhin auf *0* gesetzt werden um alle Modellinstanzen zu erhalten, die die Bedingungen von `MinScore` erfüllen. Es ist zu beachten, dass beim Tracken der Matches durch die Bildpyramide keine Matches verworfen werden, wenn Störparameter verwendet werden. Die Laufzeit mit Nutzung von Störparametern ist daher mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter `NumMatches`.

Enthält `NumMatches` einen Wert, liefert `find_aniso_shape_models` die `NumMatches` besten Instanzen des Modells unabhängig von der Art des Modells zurück. Falls z.B. in `ModelIDs` zwei Modelle übergeben werden und `NumMatches` $= 2$ gewählt wird, kann es sein, dass zwei Instanzen des ersten Modells und keine des zweiten Modells, eine Instanz des ersten Modells und eine des zweiten Modells oder keine Instanz des ersten Modells und zwei des zweiten Modells zurückgeliefert werden. Falls hingegen `NumMatches` mehrere Werte enthält, werden so viele Instanzen des jeweiligen Modells in `ModelIDs` zurückgeliefert, wie durch das entsprechende Element von `NumMatches` angegeben. Falls z.B. `NumMatches` $= [1, 1]$ gewählt wird, wird eine Instanz des ersten Modells und eine des zweiten Modells zurückgeliefert.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter `MaxOverlap` kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als `MaxOverlap` überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe `smallest_rectangle2`). Bei `MaxOverlap` $= 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei `MaxOverlap` $= 1$ werden alle gefundenen Instanzen zurückgeliefert.

Falls `MaxOverlap` einen Wert enthält, wird die Überlappung für alle gefundenen Instanzen des Modells unabhängig von der Art des Modells berechnet. D.h., dass sich zu viel überlappende Instanzen verschiedener und gleicher Modelle eliminiert werden. Falls hingegen mehrere Werte in `MaxOverlap` übergeben werden, wird die Überlappung nur innerhalb der gefundenen Instanzen des jeweiligen Typs der Modelle in `ModelIDs` berechnet. D.h., es werden nur sich zu viel überlappende Instanzen gleicher Modelle eliminiert. Modelle verschiedenen Typs können sich in diesem Modus vollständig überlappen.

SubPixel: Der Parameter `SubPixel` gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls `SubPixel` auf *'none'* (oder *'false'* für Rückwärtskompatibilität) gesetzt wird, wird die Lage des Modells nur pixelgenau bzw. mit der bei `create_aniso_shape_model` angegebenen Winkel- und Skalierungsauflösung bestimmt. Falls `SubPixel` auf *'interpolation'* (oder *'true'*) gesetzt wird, werden sowohl die Position als auch die Rotation und Skalierung subpixelgenau bestimmt. Dabei wird die Lage des Modells anhand der Score-Funktion interpoliert. Dieser Modus kostet fast keine Rechenzeit und liefert in den meisten Anwendungen eine ausreichende Genauigkeit. In manchen Anwendungen ist eine möglichst hohe Genauigkeit wichtig. In diesen Fällen kann die Lage des Modells durch Ausgleichsrechnung, d.h., durch Minimierung der Abstände der Modellpunkte und der zugehörigen Bildpunkte (Least-Squares Adjustment), bestimmt werden. Im Gegensatz zu *'interpolation'* kostet dieser Modus zusätzliche Rechenzeit. Mit den verschiedenen Modi (*'least_squares'*, *'least_squares_high'* und *'least_squares_very_high'*) kann dabei die Genauigkeit festgelegt werden, mit der der minimale Abstand gesucht wird. Je höher die Genauigkeit gewählt wird, desto länger dauert allerdings auch die Subpixel-Extraktion. Im Normalfall sollte für `SubPixel` *'interpolation'* gewählt werden. Falls die Ausgleichsrechnung gewünscht wird, sollte *'least_squares'* gewählt werden, da sich so der optimale Kompromiss zwischen Laufzeit und Genauigkeit ergibt.

Objekte, die im Vergleich zum Modell leicht deformiert im Bild erscheinen, können in manchen Fällen entweder nicht oder nur mit einer geringen Genauigkeit gefunden werden. Für solche Objekte besteht die Möglichkeit, im Parameter `SubPixel` zusätzlich die maximal zulässige Objektdeformation in Pixeln anzugeben. Dies kann mit Hilfe des optionalen Parameterwertes *'max_deformation '* und einer nachfolgenden Ganzzahl zwischen *0* und *32* (im selben String), die die maximale Deformation beschreibt, erfolgen. Kann die Form des Objektes z.B. bis zu 2 Pixel von der im Modell gespeicherten Form abweichen, muss im Parameter `SubPixel` der Wert *'max_deformation 2'* zusätzlich zum oben beschriebenen Modus für die subpixelgenaue Extraktion übergeben werden, also z.B. *['least_squares', 'max_deformation 2']*. Mit *'max_deformation 0'* wird das Objekt ohne Deformationen gesucht, also so als ob man kein *'max_deformation '* übergeben hätte. Zu beachten ist, dass größere Werte für die maximale Deformation oft zu erhöhten Laufzeiten führen. Außerdem besteht bei einem zu großen Wert die Gefahr, dass falsche Kandidaten gefunden werden. Beide Probleme können hauptsächlich bei kleinen Objekten oder Objekten mit feinen Strukturen auftreten. Der Grund hierfür ist, dass solche Objekte durch große erlaubte Deformationen ihre für die Suche wichtige charakteristische Form verlieren. Zu beachten ist auch, dass die Genauigkeit von teilweise verdeckten Objekten für höhere Deformationen abnehmen kann wenn Störobjekte in der Nähe des Objektes vorhanden sind. Die maximale Deformation sollte demnach immer nur so groß wie nötig und so gering wie möglich gewählt werden. Näherungsweise rotationssymmetrische Objekt könnten nicht gefunden werden, falls *'max_deformation'* und `AngleExtent` beide auf einen Wert größer 0 gesetzt sind. In dem Fall kann es zu Mehrdeutigkeiten kommen, die nicht aufgelöst werden können, sodass der Match als falsch verworfen wird. Wenn das passiert, versuchen Sie, entweder *'max_deformation'* oder `AngleExtent` auf 0 zu setzen oder das Modell so zu verändern, dass weniger Symmetrien auftreten. Die Score-Berechnung bei Deformationswerten größer *0* hängt von der Art der Subpixel-Extraktion ab. In den meisten Fällen wird sich der Score eines Matches verändern, wenn man *'least_squares'*, *'least_squares_high'* oder *'least_squares_very_high'* (siehe oben) wählt (im Vergleich zu *'none'* oder *'interpolation'*). Außerdem wird sich der Score in der Regel erhöhen wenn man bei einer der Least-Squares-Adjustment-Methoden die maximale Deformation erhöht, da dann zu den Modellpunkten mehr zugehörige Bildpunkte gefunden werden können. Für einen aussagekräftigen Score und zum Vermeiden falscher Matches wird empfohlen, das Erlauben einer Deformation mit einer Subpixel-Extraktion mit einer der Least-Squares-Adjustment-Methoden zu kombinieren.

Falls die Subpixel-Extraktion und/oder die maximale Objektdeformation für jedes Modell einzeln spezifiziert

werden sollen, muss in SubPixel die Subpixel-Extraktion genauso oft angegeben werden, wie die Anzahl von Elementen in ModelIDs. Nach jedem übergebenen Wert für die Subpixel-Extraktion kann optional ein zweiter Wert übergeben werden, der die maximale Objektdeformation für das jeweilige Modell beschreibt. Wird nach dem Wert für die Subpixel-Extraktion kein Wert für die Objektdeformation übergeben, wird das entsprechende Modell ohne Deformation gesucht. Falls z.B. zwei Modelle in ModelIDs angegeben sind und für das erste Modell die Subpixel-Extraktion *'interpolation'* gewählt werden soll sowie keine Objektdeformation erlaubt ist und für das zweite Modell die Subpixel-Extraktion *'least_squares'* gewünscht ist sowie eine maximale Objektdeformation von *3* Pixeln erlaubt ist, dann kann für den Parameter SubPixel das Tupel *['interpolation', 'least_squares', 'max_deformation 3']* übergeben werden. Alternativ kann auch das äquivalente Tupel *['interpolation', 'max_deformation 0', 'least_squares', 'max_deformation 3']* übergeben werden.

NumLevels: Mit NumLevels wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit create_aniso_shape_model angegebenen Bereich beschnitten. Falls NumLevels als *0* angegeben wird, wird die mit create_aniso_shape_model angegebene Anzahl verwendet.

In manchen Fällen kann es sein, dass die Anzahl der Pyramidenebenen, die beispielsweise automatisch mit create_aniso_shape_model ermittelt wurde, zu hoch ist. Dann werden eventuell Instanzen, die eine sehr hohe finale Bewertung gehabt hätten, bereits auf der höchsten Pyramidenebene ausgeschlossen und damit nicht gefunden. Anstatt nun MinScore auf einen sehr geringen Wert zu setzen, um alle Matches zu finden, kann der Wert von NumLevels mit get_shape_model_params ermittelt werden und dann ein etwas geringerer Wert in find_aniso_shape_models genutzt werden. Diese Herangehensweise führt oft zu besseren Ergebnissen hinsichtlich Geschwindigkeit und Robustheit.

Optional kann NumLevels einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, Laufzeit einzusparen. Allerdings ist in diesem Modus im Allgemeinen die Genauigkeit der gefundenen Lageparameter geringer als im Normalfall, in dem die Matches bis auf die unterste Pyramidenebene verfolgt werden. Falls eine hohe Genauigkeit erzielt werden soll, empfiehlt es sich also, mindestens *'least_squares'* für SubPixel zu verwenden.

Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden. Falls die unterste Pyramidenebene für jedes Modell einzeln spezifiziert werden soll, müssen in NumLevels zweimal die Anzahl von Elementen in ModelIDs angegeben werden. Dabei sind die Anzahl der Pyramidenebenen und der untersten Pyramidenebenen verschränkt anzugeben. Falls z.B. zwei Modelle in ModelIDs angegeben sind, die oberste Pyramidenebene für das erste Modell 5 und für das zweite Modell 4 sein soll und die unterste Pyramidenebene für das erste Modell 2 und für das zweite 1 sein soll, so ist NumLevels $= [5, 2, 4, 1]$ zu wählen. Falls genau zwei Modelle in ModelIDs angegeben sind, ergibt sich als Spezialfall, dass, falls die unterste Pyramidenebene spezifiziert werden soll, die oberste und unterste Pyramidenebene für beide Modelle explizit spezifiziert werden muss, selbst wenn sie gleich sind. Ein Tupel der Länge zwei in NumLevels wird in diesem Fall nämlich als Spezifikation der obersten Pyramidenebene für die zwei Modelle interpretiert.

In mangelhaften Eingabebildern, d.h., in Bildern die z.B. unscharf, deformiert oder verrauscht sind, kann auf der untersten Pyramidenebene oft keine Instanz des Formmodells gefunden werden, da aufgrund der mangelhaften Bildqualität nicht genügend Kanteninformation vorhanden ist, oder die Kanten zu stark deformiert sind. Auf höheren Pyramidenebenen dagegen kann die Kanteninformation noch ausreichend sein um Instanzen des Modells zu finden, wobei auch hierbei die oben beschriebenen Einschränkungen bezüglich Genauigkeit und Robustheit gelten. Die Wahl der passenden Pyramidenebene, d.h., der untersten Pyramidenebene auf der noch eine Instanz des Modells gefunden wird, hängt sowohl vom Modell als auch vom Eingabebild ab. Diese Pyramidenebene kann also von Bild zu Bild variieren. Um das Matching auf mangelhaften Bildern zu erleichtern, kann die unterste Pyramidenebene, auf der noch eine Instanz des Modells gefunden wird, während des Matchings automatisch bestimmt werden. Um diesen als 'increased tolerance mode' bezeichneten Mechanismus zu aktivieren, muss die unterste Pyramidenebene in NumLevels negativ angegeben werden. Wird z.B. NumLevels $= [5, 2, 4, -1]$ gesetzt, so ist die unterste Pyramidenebene für das erste Modell 2. Kann auf der Pyramidenebene 2 keine Instanz des ersten Formmodells gefunden werden, so wird für dieses Modell kein Ergebnis zurückgegeben. Für das zweite Modell ist in diesem Beispiel die unterste Pyramidene-

bene auf *-1* gesetzt. Daher wird versucht, eine Instanz des Formmodells auf der Pyramidenebene 1 zu finden. Kann auf dieser Pyramidenebene keine Instanz gefunden werden, so wird die unterste Pyramidenebene ermittelt, auf der noch Instanzen des Modells gefunden wurden. Die Instanzen dieser Pyramidenebene werden dann als Ergebnis zurückgegeben.

Wenn ein Modell mit `adapt_shape_model_high_noise` angepasst wurde wird die geschätzte untere Pyramidenstufe per Standardeinstellung verwendet. Der Benutzer kann den geschätzten Wert dennoch überschreiben, indem er jeweils zwei Werte an `NumLevels` übergibt und damit explizit die untere Pyramidenstufe setzt.

Greediness: Der Parameter `Greediness` bestimmt, wie „gierig“ die Suche durchgeführt werden soll. Für `Greediness` $= 0$ wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden, immer findet, wenn die anderen Parameter passend gesetzt sind. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für `Greediness` $= 1$ wird eine unsicherere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, dass das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für `Greediness` $= 1$ wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten Fällen wird das Formmodell für `Greediness` $= 0.9$ immer sicher gefunden.

Ausgabeparameter im Detail

Row, Column, Angle, ScaleR, ScaleC: Die Position, Rotation und Skalierung in Zeilen- und Spaltenrichtung der gefundenen Instanzen des Modells werden in `Row`, `Column`, `Angle`, `ScaleR` und `ScaleC` zurückgeliefert. Die Koordinaten `Row` und `Column` sind die Koordinaten des Ursprungs des Modells im Suchbild. Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit `create_aniso_shape_model` verwendet wurde. Ein anderer Ursprung kann mit `set_shape_model_origin` festgelegt werden.

Beachten Sie, dass die Koordinaten `Row` und `Column` nicht exakt mit dem Ursprung des Modells übereinstimmen und daher nicht direkt verwendet werden sollten. Die Werte sind dafür optimiert, die Transformationsmatrix zu erzeugen, mit der die Matching-Ergebnisse für verschiedene Aufgaben verwendet werden können, z.B. um ROIs für andere Bildverarbeitungsschritte einem Objekt nachzuführen. Das bei `find_aniso_shape_model` angegebene Beispiel zeigt, wie diese Matrix erzeugt wird und wie man damit das Modell an der gefundenen Position im Suchbild visualisiert.

Beachten Sie, dass die Visualisierung aber auch mit der Prozedur dev_display_shape_matching_results realisiert werden kann.

Score: In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Falls das Formmodell mit `set_shape_model_clutter` um Störparameter erweitert wurde, gibt `Score`, nach den oben genannten Werten, auch den Störwert jeder gefundenen Instanz zurück. Falls beispielsweise die Hälfte der Störregion Störkanten aufweist entspricht der Störwert 0.5. Werden z.B. zwei Instanzen gefunden, wobei für Erstere eine Bewertung von 0.9 und für die zweite 0.8 erreicht wird und der Störwert der ersten Instanz 0.2, der zweiten Instanz 0.1 ist, wird `Score` $= [0.9, 0.8, 0.2, 0.1]$ zurückgegeben. Es ist zu beachten, dass von sämtlichen Ergebnissen des formbasierten Matchings die Störwerte am stärksten von Beleuchtungsänderungen beeinflusst werden.

Model: Der gefundene Modelltyp wird in `Model` zurückgeliefert. Die Elemente von `Model` sind ein Index in das Tupel `ModelIDs`, d.h. sie können Werte von 0 bis $|$`ModelIDs`$| - 1$ enthalten. Ein Wert von 0 in einem Element von `Model` entspricht also einer Instanz des ersten in `ModelIDs` übergebenen Modells.

Setzen eines Timeout

Mittels des Operators `set_shape_model_param` können Sie einen *'timeout'* für `find_aniso_shape_models` angeben. Falls die durch `ModelIDs` referenzierten Modelle unterschiedliche Werte für *'timeout'* besitzen, wählt `find_aniso_shape_models` den niedrigsten aus. Wenn `find_aniso_shape_models` diesen *'timeout'* erreicht endet er ohne Ergebnis und gibt den Fehlercode 9400 (H_ERR_TIMEOUT) zurück. Abhängig von der Größe der Skalierungsintervalle, die durch `ScaleRMin`, `ScaleRMax`, `ScaleCMin` und `ScaleCMax` festgelegt werden, benötigt `find_aniso_shape_models` eine signifikante Zeitspanne um den Speicher für gecachte Transformationen freizugeben, falls das Modell nicht vorberechnet wurde. Da diese Transformationen auch freigegeben werden müssen nachdem ein Timeout auftritt, überschreitet `find_aniso_shape_models` den in *'timeout'* angegebenen Wert um diese Zeitspanne.

MinContrast bei mehreren Modellen

Wenn über den Parameter `ModelIDs` mehrere Modelle übergeben werden, sollten diese mit dem gleichen Wert für `MinContrast` erzeugt worden sein. Falls sie mit unterschiedlichen Werten für `MinContrast` erzeugt worden sind, wird in `find_aniso_shape_models` der kleinste dieser Werte verwendet.

Darstellung der Resultate

Die Verwendung der Prozedur `dev_display_shape_matching_results` wird zur Darstellung der Resultate formbasierten Matchings stark empfohlen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Eingabebild, in dem die Modelle gefunden werden sollen.

▷ **`ModelIDs`** (input_control) . . . . . shape_model(-array) ⇝ *handle*
Handle der Modelle.

▷ **`AngleStart`** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Kleinste auftretende Rotation der Modelle.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **`AngleExtent`** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
Restriktion: AngleExtent >= 0

▷ **`ScaleRMin`** (input_control) . . . . . number(-array) ⇝ *real*
Kleinste auftretende Skalierung der Modelle in Zeilenrichtung.
Defaultwert: 0.9
Wertevorschläge: `ScaleRMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleRMin > 0

▷ **`ScaleRMax`** (input_control) . . . . . number(-array) ⇝ *real*
Größte auftretende Skalierung der Modelle in Zeilenrichtung.
Defaultwert: 1.1
Wertevorschläge: `ScaleRMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleRMax >= ScaleRMin

▷ **`ScaleCMin`** (input_control) . . . . . number(-array) ⇝ *real*
Kleinste auftretende Skalierung der Modelle in Spaltenrichtung.
Defaultwert: 0.9
Wertevorschläge: `ScaleCMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleCMin > 0

▷ **`ScaleCMax`** (input_control) . . . . . number(-array) ⇝ *real*
Größte auftretende Skalierung der Modelle in Spaltenrichtung.
Defaultwert: 1.1
Wertevorschläge: `ScaleCMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleCMax >= ScaleCMin

▷ **`MinScore`** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen der Modelle.
Defaultwert: 0.5
Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **`NumMatches`** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der zu findenden Instanzen der Modelle (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real(-array) ⇝ *real*
Maximale Überlappung der zu findenden Instanzen der Modelle.
Defaultwert: 0.5
Wertevorschläge: MaxOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MaxOverlap ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **SubPixel** (input_control) . . . . . string(-array) ⇝ *string*
Subpixelgenauigkeit falls ungleich *'none'*.
Defaultwert: 'least_squares'
Wertevorschläge: SubPixel ∈ {'none', 'interpolation', 'least_squares', 'least_squares_high', 'least_squares_very_high', 'max_deformation 1', 'max_deformation 2', 'max_deformation 3', 'max_deformation 4', 'max_deformation 5', 'max_deformation 6'}

▷ **NumLevels** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls |NumLevels| = 2).
Defaultwert: 0
Werteliste: NumLevels ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real(-array) ⇝ *real*
„Gierigkeit" der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen" werden).
Defaultwert: 0.9
Wertevorschläge: Greediness ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ Greediness ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate der gefundenen Instanzen der Modelle.

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate der gefundenen Instanzen der Modelle.

▷ **Angle** (output_control) . . . . . angle.rad-array ⇝ *real*
Rotationswinkel der gefundenen Instanzen der Modelle.

▷ **ScaleR** (output_control) . . . . . number-array ⇝ *real*
Skalierung der gefundenen Instanzen der Modelle in Zeilenrichtung.

▷ **ScaleC** (output_control) . . . . . number-array ⇝ *real*
Skalierung der gefundenen Instanzen der Modelle in Spaltenrichtung.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Instanzen der Modelle.

▷ **Model** (output_control) . . . . . integer-array ⇝ *integer*
Index der gefundenen Instanzen der Modelle.

Beispiel

```
read_image (Image, 'pcb_focus/pcb_focus_telecentric_061')
gen_rectangle1 (ROI_0, 236, 241, 313, 321)
gen_circle (ROI_1, 281, 653, 41)
reduce_domain (Image, ROI_0, ImageReduced1)
reduce_domain (Image, ROI_1, ImageReduced2)
create_aniso_shape_model (ImageReduced1,'auto', -0.39, 0.79, 'auto', 0.9, \
                          1.1, 'auto', 0.9, 1.1, 'auto', 'auto', \
                          'use_polarity', 'auto', 'auto', ModelID1)
create_aniso_shape_model (ImageReduced2,'auto', -0.39, 0.79, 'auto', 0.9, \
                          1.1, 'auto', 0.9, 1.1, 'auto', 'auto', \
                          'use_polarity', 'auto', 'auto', ModelID2)
ModelIDs:=[ModelID1, ModelID2]
find_aniso_shape_models(Image, ModelIDs, -0.39, 0.79, 0.9, 1.1, 0.9, 1.1, \
                        0.5, 1, 0.5, 'least_squares', 0, 0.9, Row, Column, \
                        Angle, ScaleR, ScaleC, Score, Model)
* Display results
```

```
dev_display_shape_matching_results (ModelIDs, 'red', Row, Column, Angle, \
                                    ScaleR, ScaleC, Model)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_aniso_shape_models` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`add_channels`, `create_aniso_shape_model`, `read_shape_model`, `set_shape_model_origin`, `set_shape_model_clutter`

Nachfolger

`clear_shape_model`

Alternativen

`find_shape_models`, `find_scaled_shape_models`, `find_shape_model`, `find_scaled_shape_model`, `find_aniso_shape_model`, `find_ncc_model`, `find_ncc_models`

Siehe auch

`set_system`, `get_system`, `set_shape_model_param`

Modul

Matching

find_generic_shape_model

```
find_generic_shape_model ( SearchImage : : ModelID : MatchResultID,
    NumMatchResult )
```

Suche der besten Matches eines oder mehrerer Formmodelle in einem Bild.

Der Operator `find_generic_shape_model` findet die besten Instanzen eines oder mehrerer Formmodelle, die in `ModelID` übergeben werden, im Eingabebild `SearchImage`. Die gefundenen Matches werden in `MatchResultID` zurückgegeben und können mit `get_generic_shape_model_result` und `get_generic_shape_model_result_object` abgefragt werden. Die Anzahl gefundener Matches wird in `NumMatchResult` zurückgegeben. Das Modell kann mit `set_generic_shape_model_param` und `set_generic_shape_model_object` parametrisiert werden, wodurch der Suchvorgang gesteuert werden kann.

Die Domäne des Eingabebildes `SearchImage` bestimmt die ROI für die Suche. Sie schränkt den Suchbereich ein, da sie als Begrenzung für den Schwerpunkt des Modells `ModelID` gilt, wodurch der Matching-Vorgang beschleunigt wird. Für Informationen zum Übergeben von Bild-Tupeln an `SearchImage` siehe unten. Desweiteren ist der Suchbereich durch die Größe von `SearchImage` begrenzt. Standardmäßig wird `ModelID` nur an den Stellen der Domäne gesucht an denen `ModelID` komplett hineinpasst. Dementsprechend wird `ModelID` nicht gefunden wenn es über den Bildrand hinausragt. Dies gilt für alle Stufen der Bildpyramide.

In seltenen Fällen, welche typischerweise in künstlich erzeugten Bildern auftreten, können Instanzen von `ModelID` nicht im Bild gefunden werden wenn sie den Rand der Domäne auf irgendeiner Stufe der Bildpyramide berühren. Als Daumenregel kann man annehmen, dass `ModelID` nicht gefunden werden könnte wenn dessen Entfernung zum Bildrand kleiner als $2^{NumLevels-1}$ Pixel ist (wobei $NumLevels$ für die Anzahl an Pyramidenstufen steht). Dieses Verhalten kann mittels *'border_shape_models'* geändert werden, siehe `set_generic_shape_model_param`.

Werden mehrere Modelle gesucht, dann kann der Suchbereich für alle Modelle gleichzeitig eingeschränkt werden indem ein einzelnes Bild mit reduzierter Domäne in `SearchImage` übergeben wird. Alternativ kann der

Suchbereich auch für jedes Modell individuell eingeschränkt werden indem ein Objekt mit mehreren Bildobjekten übergeben wird, eins für jedes Modell in ModelID. In diesem Fall müssen die Bilder, abgesehen von deren Domänen, identisch sein. Werden mehrere Modelle angewendet, müssen diese unterschiedliche Identifikatoren (*'model_identifier'*) aufweisen. Dadurch kann jede gefundene Instanz dem entsprechenden Formmodell mit dem es gefunden wurde zugeordnet werden. Ist dies nicht der Fall wird eine Fehlerbehandlung durchgeführt.

Parameter

- ▷ **SearchImage** (input_object) ..... (multichannel-)object(-array) ⇝ *object*
  Bild, in dem das Modell gesucht wird.
- ▷ **ModelID** (input_control) ..... shape_model(-array) ⇝ *handle*
  Handle des Formmodells.
- ▷ **MatchResultID** (output_control) ..... generic_shape_model_result ⇝ *handle*
  Handle der gefundenen Matches.
- ▷ **NumMatchResult** (output_control) ..... integer ⇝ *integer*
  Anzahl gefundener Matches.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_generic_shape_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`train_generic_shape_model`, `set_generic_shape_model_param`

Modul

Matching

```
find_scaled_shape_model ( Image : : ModelID, AngleStart,
    AngleExtent, ScaleMin, ScaleMax, MinScore, NumMatches,
    MaxOverlap, SubPixel, NumLevels, Greediness : Row, Column, Angle,
    Scale, Score )
```

Suche der besten Matches eines isotrop skalierten Formmodells in einem Bild.

`find_scaled_shape_model` findet die besten `NumMatches` Instanzen des isotrop skalierten Formmodells `ModelID` im Eingabebild `Image`. Das Modell muss zuvor mit `create_scaled_shape_model` erzeugt oder mit `read_shape_model` eingelesen worden sein.

Die Position, Rotation und Skalierung der gefundenen Instanzen des Modells werden in `Row`, `Column`, `Angle` und `Scale` zurückgeliefert. In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Nähere Informationen finden Sie in den parameterspezifischen Abschnitten.

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes `Image` gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit `create_scaled_shape_model` verwendet wurde. Ein eventuell mit

set_shape_model_origin anders gesetzter Ursprung wird nicht berücksichtigt. Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als MinScore erreichen würde (siehe unten). Außerdem kann es sein, dass das Modell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit set_system('border_shape_models','true') für alle Modelle oder mit set_shape_model_param(ModelID,'border_shape_models','true') speziell für ein Modell umgestellt werden, so dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als MinScore erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit dilation_circle zu vergrößern.

AngleStart, AngleExtent, ScaleMin, ScaleMax: Die Parameter AngleStart und AngleExtent legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Die Parameter ScaleMin und ScaleMax legen den Bereich der Skalierungen fest, in dem nach dem Modell gesucht wird. Der Winkel- und Skalierungsbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit create_scaled_shape_model angegeben worden ist. Dies bedeutet insbesondere, dass die Winkelbereiche des Modells und der Suche sich überlappen müssen.

Beachten Sie, dass in einigen Fällen Instanzen mit einer Rotation oder Skalierung gefunden werden, die geringfügig außerhalb des angegebenen Winkel- und Skalierungsbereichs liegen. Dies kann vorkommen, wenn der angegebene Bereich kleiner ist als der bei der Erstellung des Modells angegebene Bereich. AngleStart und AngleExtent sowie ScaleMin und ScaleMax werden nur auf der höchsten Pyramidenstufe überprüft. Matches, die auf der höchsten Pyramidenstufe gefunden werden, werden bis zur untersten Pyramidenstufe verfeinert. Aus Leistungsgründen wird bei der Verfeinerung jedoch nicht mehr überprüft, ob die Matches immer noch innerhalb der angegebenen Bereiche liegen.

MinScore: Der Parameter MinScore legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von MinScore gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, dass das Modell niemals verdeckt wird, kann MinScore auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden. Werden die Matches nicht bis zur untersten Pyramidenebene verfolgt (siehe unten) kann es in manchen Fällen vorkommen, dass Instanzen gefunden werden, deren Score geringfügig unter dem Wert von MinScore liegen.

Falls das Formmodell mit set_shape_model_clutter um Störparameter erweitert wurde und dadurch *'use_clutter'* aktiviert wurde, erwartet MinScore einen zweiten Wert, welcher den maximalen Störwert eines potentiellen Matches angibt, um als Instanz des Modells im Bild betrachtet werden zu können. Die Laufzeit mit Nutzung von Störparametern ist mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter NumMatches.

NumMatches: Mit NumMatches kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als NumMatches Instanzen eine Bewertung größer als MinScore erreichen, werden nur die besten NumMatches Instanzen zurückgeliefert. Falls weniger als NumMatches Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter MinScore hat Vorrang vor NumMatches. Sollen alle Modellinstanzen, deren Bewertung MinScore übersteigt, im Bild gefunden werden, muss NumMatches auf *0* gesetzt werden.

Beim Tracken der Matches durch die Bildpyramide werden auf jeder Ebene Matches verworfen, die wenig vielversprechend sind. Dadurch ist es möglich, dass Matches verworfen werden, die auf der niedrigsten Pyramidenebene einen höheren Score gehabt hätten. Beispielsweise kann das dazu führen, dass der gefundene Match für NumMatches gleich *1* nicht mit dem Match mit dem höchsten Score übereinstimmt, der von NumMatches gleich *0* oder > *1* zurückgegeben wird.

Falls erwartet wird, dass mehrere Objekte mit einem ähnlichen Score gefunden werden können, aber nur das Modell mit dem höchsten Score zurückgegeben werden soll, kann es vorteilhaft sein, NumMatches zu erhöhen, und dann den Match mit dem höchsten Score auszuwählen.

Für den Fall, dass das Formmodell mit set_shape_model_clutter um Störparameter erweitert wurde, berücksichtigt NumMatches auch den zweiten Wert, der in MinScore übergeben wird. Gefundene Instanzen weisen in diesem Fall einen Score auf, der größer als der erste Eintrag ist, sowie einen Störwert der niedriger als der zweite Eintrag ist. Ist die Anzahl gefundener Instanzen größer als NumMatches, werden

die besten NumMatches Instanzen in Bezug auf ihren Störwert zurückgegeben. MinScore hat weiterhin Vorrang vor NumMatches und NumMatches muss weiterhin auf *0* gesetzt werden um alle Modellinstanzen zu erhalten, die die Bedingungen von MinScore erfüllen. Es ist zu beachten, dass beim Tracken der Matches durch die Bildpyramide keine Matches verworfen werden, wenn Störparameter verwendet werden. Die Laufzeit mit Nutzung von Störparametern ist daher mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter NumMatches.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter MaxOverlap kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als MaxOverlap überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe smallest_rectangle2). Bei MaxOverlap $= 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei MaxOverlap $= 1$ werden alle gefundenen Instanzen zurückgeliefert.

SubPixel: Der Parameter SubPixel gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls SubPixel auf *'none'* (oder *'false'* für Rückwärtskompatibilität) gesetzt wird, wird die Lage des Modells nur pixelgenau bzw. mit der bei create_scaled_shape_model angegebenen Winkel- und Skalierungsauflösung bestimmt. Falls SubPixel auf *'interpolation'* (oder *'true'*) gesetzt wird, werden sowohl die Position als auch die Rotation und Skalierung subpixelgenau bestimmt. Dabei wird die Lage des Modells anhand der Score-Funktion interpoliert. Dieser Modus kostet fast keine Rechenzeit und liefert in den meisten Anwendungen eine ausreichende Genauigkeit. In manchen Anwendungen ist eine möglichst hohe Genauigkeit wichtig. In diesen Fällen kann die Lage des Modells durch Ausgleichsrechnung, d.h., durch Minimierung der Abstände der Modellpunkte und der zugehörigen Bildpunkte (Least-Squares Adjustment), bestimmt werden. Im Gegensatz zu *'interpolation'* kostet dieser Modus zusätzliche Rechenzeit. Mit den verschiedenen Modi (*'least_squares'*, *'least_squares_high'* und *'least_squares_very_high'*) kann dabei die Genauigkeit festgelegt werden, mit der der minimale Abstand gesucht wird. Je höher die Genauigkeit gewählt wird, desto länger dauert allerdings auch die Subpixel-Extraktion. Im Normalfall sollte für SubPixel *'interpolation'* gewählt werden. Falls die Ausgleichsrechnung gewünscht wird, sollte *'least_squares'* gewählt werden, da sich so der optimale Kompromiss zwischen Laufzeit und Genauigkeit ergibt.

Objekte, die im Vergleich zum Modell leicht deformiert im Bild erscheinen, können in manchen Fällen entweder nicht oder nur mit einer geringen Genauigkeit gefunden werden. Für solche Objekte besteht die Möglichkeit, im Parameter SubPixel zusätzlich die maximal zulässige Objektdeformation in Pixeln anzugeben. Dies kann mit Hilfe des optionalen Parameterwertes *'max_deformation '* und einer nachfolgenden Ganzzahl zwischen *0* und *32* (im selben String), die die maximale Deformation beschreibt, erfolgen. Kann die Form des Objektes z.B. bis zu 2 Pixel von der im Modell gespeicherten Form abweichen, muss im Parameter SubPixel der Wert *'max_deformation 2'* zusätzlich zum oben beschriebenen Modus für die subpixelgenaue Extraktion übergeben werden, also z.B. *['least_squares', 'max_deformation 2']*. Mit *'max_deformation 0'* wird das Objekt ohne Deformationen gesucht, also so als ob man kein *'max_deformation '* übergeben hätte. Zu beachten ist, dass größere Werte für die maximale Deformation oft zu erhöhten Laufzeiten führen. Außerdem besteht bei einem zu großen Wert die Gefahr, dass falsche Kandidaten gefunden werden. Beide Probleme können hauptsächlich bei kleinen Objekten oder Objekten mit feinen Strukturen auftreten. Der Grund hierfür ist, dass solche Objekte durch große erlaubte Deformationen ihre für die Suche wichtige charakteristische Form verlieren. Zu beachten ist auch, dass die Genauigkeit von teilweise verdeckten Objekten für höhere Deformationen abnehmen kann wenn Störobjekte in der Nähe des Objektes vorhanden sind. Die maximale Deformation sollte demnach immer nur so groß wie nötig und so gering wie möglich gewählt werden. Näherungsweise rotationssymmetrische Objekt könnten nicht gefunden werden, falls *'max_deformation'* und AngleExtent beide auf einen Wert größer 0 gesetzt sind. In dem Fall kann es zu Mehrdeutigkeiten kommen, die nicht aufgelöst werden können, sodass der Match als falsch verworfen wird. Wenn das passiert, versuchen Sie, entweder *'max_deformation'* oder AngleExtent auf 0 zu setzen oder das Modell so zu verändern, dass weniger Symmetrien auftreten. Die Score-Berechnung bei Deformationswerten größer *0* hängt von der Art der Subpixel-Extraktion ab. In den meisten Fällen wird sich der Score eines Matches verändern, wenn man *'least_squares'*, *'least_squares_high'* oder *'least_squares_very_high'* (siehe oben) wählt (im Vergleich zu *'none'* oder *'interpolation'*). Außerdem wird sich der Score in der Regel erhöhen wenn man bei einer der Least-Squares-Adjustment-Methoden die maximale Deformation erhöht, da dann zu den Modellpunkten mehr zugehörige Bildpunkte gefunden werden können. Für einen aussagekräftigen Score und zum Vermeiden falscher Matches wird empfohlen, das Erlauben einer Deformation mit einer Subpixel-Extraktion mit einer der Least-Squares-Adjustment-Methoden zu kombinieren.

NumLevels: Mit NumLevels wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit create_scaled_shape_model angegebenen Bereich beschnitten. Falls NumLevels als *0* angegeben wird, wird die mit create_scaled_shape_model angegebene Anzahl verwendet.

In manchen Fällen kann es sein, dass die Anzahl der Pyramidenebenen, die beispielsweise automatisch mit create_scaled_shape_model ermittelt wurde, zu hoch ist. Dann werden eventuell Instanzen, die eine sehr hohe finale Bewertung gehabt hätten, bereits auf der höchsten Pyramidenebene ausgeschlossen und damit nicht gefunden. Anstatt nun MinScore auf einen sehr geringen Wert zu setzen, um alle Matches zu finden, kann der Wert von NumLevels mit get_shape_model_params ermittelt werden und dann ein etwas geringerer Wert in find_scaled_shape_model genutzt werden. Diese Herangehensweise führt oft zu besseren Ergebnissen hinsichtlich Geschwindigkeit und Robustheit.

Optional kann NumLevels einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, Laufzeit einzusparen. Allerdings ist in diesem Modus im Allgemeinen die Genauigkeit der gefundenen Lageparameter geringer als im Normalfall, in dem die Matches bis auf die unterste Pyramidenebene verfolgt werden. Falls eine hohe Genauigkeit erzielt werden soll, empfiehlt es sich also, mindestens *'least_squares'* für SubPixel zu verwenden. Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden.

In mangelhaften Eingabebildern, d.h., in Bildern die z.B. unscharf, deformiert oder verrauscht sind, kann auf der untersten Pyramidenebene oft keine Instanz des Formmodells gefunden werden, da aufgrund der mangelhaften Bildqualität nicht genügend Kanteninformation vorhanden ist, oder die Kanten zu stark deformiert sind. Auf höheren Pyramidenebenen dagegen kann die Kanteninformation noch ausreichend sein um Instanzen des Modells zu finden, wobei auch hierbei die oben beschriebenen Einschränkungen bezüglich Genauigkeit und Robustheit gelten. Die Wahl der passenden Pyramidenebene, d.h., der untersten Pyramidenebene auf der noch eine Instanz des Modells gefunden wird, hängt sowohl vom Modell als auch vom Eingabebild ab. Diese Pyramidenebene kann also von Bild zu Bild variieren. Um das Matching auf mangelhaften Bildern zu erleichtern, kann die unterste Pyramidenebene, auf der noch eine Instanz des Modells gefunden wird, während des Matchings automatisch bestimmt werden. Um diesen als 'increased tolerance mode' bezeichneten Mechanismus zu aktivieren, muss die unterste Pyramidenebene in NumLevels negativ angegeben werden. Wird z.B. NumLevels $= [4, -2]$ gesetzt, so wird das Matching auf der vierten Pyramidenebene begonnen und auf der zweituntersten Pyramidenebene beendet. Es wird also versucht, eine Instanz des Formmodells auf der Pyramidenebene 2 zu finden. Kann auf dieser Pyramidenebene keine Instanz gefunden werden, so wird die unterste Pyramidenebene ermittelt, auf der noch Instanzen des Modells gefunden wurden. Die Instanzen dieser Pyramidenebene werden dann als Ergebnis zurückgegeben.

Wenn ModelID mit adapt_shape_model_high_noise angepasst wurde, wird die geschätzte untere Pyramidenstufe per Standardeinstellung verwendet. Der Benutzer kann den geschätzten Wert dennoch überschreiben, indem er zwei Werte an NumLevels übergibt und damit explizit die untere Pyramidenstufe setzt.

Greediness: Der Parameter Greediness bestimmt, wie „gierig" die Suche durchgeführt werden soll. Für Greediness $= 0$ wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden, immer findet, wenn die anderen Parameter passend gesetzt sind. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für Greediness $= 1$ wird eine unsicherere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, dass das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für Greediness $= 1$ wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten Fällen wird das Formmodell für Greediness $= 0.9$ immer sicher gefunden.

Ausgabeparameter im Detail

Row, Column, Angle und Scale: Die Position, Rotation und Skalierung der gefundenen Instanzen des Modells werden in Row, Column, Angle und Scale zurückgeliefert. Die Koordinaten Row und Column beschreiben die Position des Ursprungs des Modells im Suchbild. Die Werte sind allerdings bereits geringfügig angepasst, so dass sie direkt zur Erstellung einer Transformationsmatrix verwendet werden können, die für ein Alignment oder die Visualisierung der Modellkanten benötigt wird. Diese Anpassung der Koordinaten hat mit der Art und Weise zu tun, in der HALCON ikonische Objekte transformiert, siehe hierzu

affine_trans_pixel. Das Beispiel weiter unten zeigt, wie die Transformationsmatrix erzeugt wird und wie man damit das Modell an der gefundenen Position im Suchbild visualisiert.

Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_scaled_shape_model verwendet wurde. Ein anderer Ursprung kann mit set_shape_model_origin festgelegt werden.

Score: In Score wird die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Falls das Formmodell mit set_shape_model_clutter um Störparameter erweitert wurde, gibt Score, nach den oben genannten Werten, auch den Störwert jeder gefundenen Instanz zurück. Falls beispielsweise die Hälfte der Störregion Störkanten aufweist entspricht der Störwert 0.5. Werden z.B. zwei Instanzen gefunden, wobei für Erstere eine Bewertung von 0.9 und für die zweite 0.8 erreicht wird und der Störwert der ersten Instanz 0.2, der zweiten Instanz 0.1 ist, wird Score $= [0.9, 0.8, 0.2, 0.1]$ zurückgegeben. Es ist zu beachten, dass von sämtlichen Ergebnissen des formbasierten Matchings die Störwerte am stärksten von Beleuchtungsänderungen beeinflusst werden.

Setzen eines Timeout

Mittels des Operators set_shape_model_param können Sie einen *'timeout'* für find_scaled_shape_model angeben. Wenn find_scaled_shape_model diesen *'timeout'* erreicht endet er ohne Ergebnis und gibt den Fehlercode 9400 (H_ERR_TIMEOUT) zurück. Abhängig von der Größe des Skalierungsintervalls, das durch ScaleMin und ScaleMax festgelegt wird, benötigt find_scaled_shape_model eine signifikante Zeitspanne um den Speicher für gecachte Transformationen freizugeben, falls das Modell nicht vorberechnet wurde. Da diese Transformationen auch freigegeben werden müssen nachdem ein Timeout auftritt, überschreitet find_scaled_shape_model den in *'timeout'* angegebenen Wert um diese Zeitspanne.

Darstellung der Resultate

Die Verwendung der Prozedur dev_display_shape_matching_results wird zur Darstellung der Resultate formbasierten Matchings stark empfohlen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild, in dem das Modell gefunden werden soll.
- ▷ **ModelID** (input_control) . . . . . shape_model ⇝ *handle*
  Handle des Modells.
- ▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
  Kleinste auftretende Rotation des Modells.
  Defaultwert: -0.39
  Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Defaultwert: 0.78
  Wertevorschläge: AngleExtent ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
  Restriktion: AngleExtent >= 0
- ▷ **ScaleMin** (input_control) . . . . . number ⇝ *real*
  Kleinste auftretende Skalierung des Modells.
  Defaultwert: 0.9
  Wertevorschläge: ScaleMin ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Restriktion: ScaleMin > 0
- ▷ **ScaleMax** (input_control) . . . . . number ⇝ *real*
  Größte auftretende Skalierung des Modells.
  Defaultwert: 1.1
  Wertevorschläge: ScaleMax ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
  Restriktion: ScaleMax >= ScaleMin

▷ **MinScore** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: MinScore ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MinScore ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumMatches** (input_control) . . . . . integer ⇝ *integer*
Anzahl der zu findenden Instanzen des Modells (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: NumMatches ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Überlappung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: MaxOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MaxOverlap ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **SubPixel** (input_control) . . . . . string(-array) ⇝ *string*
Subpixelgenauigkeit falls ungleich *'none'*.
Defaultwert: 'least_squares'
Wertevorschläge: SubPixel ∈ {'none', 'interpolation', 'least_squares', 'least_squares_high', 'least_squares_very_high', 'max_deformation 1', 'max_deformation 2', 'max_deformation 3', 'max_deformation 4', 'max_deformation 5', 'max_deformation 6'}

▷ **NumLevels** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls |NumLevels| = 2).
Defaultwert: 0
Werteliste: NumLevels ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real ⇝ *real*
„Gierigkeit" der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen" werden).
Defaultwert: 0.9
Wertevorschläge: Greediness ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ Greediness ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate der gefundenen Instanzen des Modells.

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate der gefundenen Instanzen des Modells.

▷ **Angle** (output_control) . . . . . angle.rad-array ⇝ *real*
Rotationswinkel der gefundenen Instanzen des Modells.

▷ **Scale** (output_control) . . . . . number-array ⇝ *real*
Skalierung der gefundenen Instanzen des Modells.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Instanzen des Modells.

Beispiel

```
create_scaled_shape_model (ImageReduced, 0, rad(-45), rad(180), 0, \
                           0.9, 1.1, 0, 'none', 'use_polarity', \
                           30, 10, ModelID)
get_shape_model_contours (ModelXLD, ModelID, 1)
find_scaled_shape_model (SearchImage, ModelID, rad(-45), rad(180), \
                         0.9, 1.1, 0.5, 1, 0.5, 'interpolation', \
                         0, 0, Row, Column, Angle, Scale, Score)
* Create transformation matrix
vector_angle_to_rigid (0, 0, 0, Row, Column, Angle, HomMat2DTmp)
```

```
hom_mat2d_scale (HomMat2DTmp, Scale, Scale, Row, Column, HomMat2DObject)
* Calculate true position of the model origin in the search image
affine_trans_pixel (HomMat2DObject, 0, 0, RowObject, ColObject)
* Display results
dev_display_shape_matching_results (ModelID, 'red', Row, Column, Angle, \
                    Scale, 1, 0)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_scaled_shape_model` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_scaled_shape_model`, `read_shape_model`, `set_shape_model_origin`, `set_shape_model_clutter`

Nachfolger

`clear_shape_model`

Alternativen

`find_shape_model`, `find_aniso_shape_model`, `find_shape_models`, `find_scaled_shape_models`, `find_aniso_shape_models`, `find_ncc_model`, `find_ncc_models`

Siehe auch

`set_system`, `get_system`, `set_shape_model_param`

Modul

Matching

find_scaled_shape_models (Image : : ModelIDs, AngleStart, AngleExtent, ScaleMin, ScaleMax, MinScore, NumMatches, MaxOverlap, SubPixel, NumLevels, Greediness : Row, Column, Angle, Scale, Score, Model)

Suche der besten Matches mehrerer isotrop skalierter Formmodelle.

`find_scaled_shape_models` findet die besten `NumMatches` Instanzen der isotrop skalierten Formmodelle, die in `ModelIDs` übergeben werden, im Eingabebild `Image`. Die Modelle müssen zuvor mit `create_scaled_shape_model` erzeugt oder mit `read_shape_model` eingelesen worden sein. Im Gegensatz zu `find_scaled_shape_model` kann also eine Mehrzahl von Modellen gleichzeitig im selben Bild gesucht werden.

Die Position, Rotation und Skalierung der gefundenen Instanzen des Modells werden in `Row`, `Column`, `Angle` und `Scale` zurückgeliefert. In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Der gefundene Modelltyp wird in `Model` zurückgeliefert. Nähere Informationen finden Sie in den parameterspezifischen Abschnitten.

Besonderheiten der Parametersemantik

Gegenüber `find_scaled_shape_model` unterscheidet sich die Semantik aller Eingabeparameter etwas. Alle Eingabeparameter müssen entweder genau ein Element enthalten, oder dieselbe Anzahl von Elementen wie `ModelIDs`. (`NumLevels` kann auch zwei oder zweimal die Anzahl von Elementen wie `ModelIDs` enthalten; siehe unten.) Im ersten Fall wird der Wert des Eingabeparameters für alle Modelle gleich verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für das entsprechende Modell in `ModelIDs` verwendet.

Zusätzliche Details können in den parameterspezifischen Abschnitten weiter unten gefunden werden. Ein Aufruf von find_scaled_shape_models mit mehreren Werten für ModelIDs, NumMatches und MaxOverlap entspricht von der Wirkung her also mehreren unabhängigen Aufrufen von find_scaled_shape_model mit den jeweiligen Parametern, ist allerdings wesentlich effizienter.

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes Image gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_scaled_shape_model verwendet wurde. Ein eventuell mit set_shape_model_origin anders gesetzter Ursprung wird nicht berücksichtigt. Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als MinScore erreichen würde (siehe unten). Außerdem kann es sein, dass das Modell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit set_system('border_shape_models','true') für alle Modelle oder mit set_shape_model_param(ModelID,'border_shape_models','true') speziell für ein Modell umgestellt werden, so dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als MinScore erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit dilation_circle zu vergrößern.

Wie üblich wird die Region des Eingabebildes Image als Suchbereich für den Referenzpunkt der Modelle ModelIDs verwendet. Konsistent mit der obigen Semantik kann Image also ein einzelnes Bildobjekt beinhalten oder ein Bildobjekttupel. Falls Image ein einzelnes Bildobjekt enthält, wird dessen Region als Suchbereich für alle Modelle in ModelIDs verwendet. Falls Image mehrere Bildobjekte enthält, wird die jeweilige Region als Suchbereich für das entsprechende Modell aus ModelIDs verwendet. In diesem Fall müssen die Bilder, abgesehen von ihrer Region, identisch sein. D.h. Image kann nicht in beliebiger Weise mit concat_obj aufgebaut werden, sondern muss aus demselben Bild mit add_channels oder äquivalenten Aufrufen erzeugt werden. Falls das nicht der Fall ist, wird eine Fehlermeldung zurückgeliefert.

AngleStart, AngleExtent, ScaleMin, ScaleMax: Die Parameter AngleStart und AngleExtent legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Die Parameter ScaleMin und ScaleMax legen den Bereich der Skalierungen fest, in dem nach dem Modell gesucht wird. Der Winkel- und Skalierungsbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit create_scaled_shape_model angegeben worden ist. Dies bedeutet insbesondere, dass die Winkelbereiche des Modells und der Suche sich überlappen müssen.

Beachten Sie, dass in einigen Fällen Instanzen mit einer Rotation oder Skalierung gefunden werden, die geringfügig außerhalb des angegebenen Winkel- und Skalierungsbereichs liegen. Dies kann vorkommen, wenn der angegebene Bereich kleiner ist als der bei der Erstellung des Modells angegebene Bereich. AngleStart und AngleExtent sowie ScaleMin und ScaleMax werden nur auf der höchsten Pyramidenstufe überprüft. Matches, die auf der höchsten Pyramidenstufe gefunden werden, werden bis zur untersten Pyramidenstufe verfeinert. Aus Leistungsgründen wird bei der Verfeinerung jedoch nicht mehr überprüft, ob die Matches immer noch innerhalb der angegebenen Bereiche liegen.

MinScore: Der Parameter MinScore legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von MinScore gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, dass das Modell niemals verdeckt wird, kann MinScore auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden. Werden die Matches nicht bis zur untersten Pyramidenebene verfolgt (siehe unten) kann es in manchen Fällen vorkommen, dass Instanzen gefunden werden, deren Scores geringfügig unter dem Wert von MinScore liegen.

Falls ein einzelner Wert in MinScore übergeben wird, wird dieser Wert auf alle Instanzen angewandt, die mit allen Modellen gefunden werden. Falls MinScore mehrere Einträge enthält, werden die Werte einzeln für die entsprechenden Modelle verwendet.

Falls die Formmodelle mit set_shape_model_clutter um Störparameter erweitert wurden und dadurch *'use_clutter'* aktiviert wurde, wird in MinScore zu jedem der vorhandenen Werte für den minimalen Score ein weiterer Wert erwartet, welcher den maximalen Störwert eines potentiellen Matches angibt, um als

Instanz des Modells im Bild betrachtet werden zu können. Die Laufzeit mit Nutzung von Störparametern ist mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter NumMatches. Die verschiedenen Formmodelle müssen den gleichen Wert für *'use_clutter'* haben.

Wird der maximale Störwert separat für jedes Modell spezifiziert, was immer dann nötig ist, wenn auch der minimale Score für jedes Modell gesetzt wird, muss in MinScore die doppelte Anzahl an Einträgen enthalten sein wie in ModelIDs. Dabei müssen der minimale Score und der maximale Störwert verschränkt in MinScore angegeben werden. Werden in ModelIDs beispielsweise zwei Modelle angegeben, wobei für Ersteres ein minimaler Score von 0.9 und für das zweite 0.8 festgelegt wird und der maximale Störwert des ersten Modells 0.1, für das zweite Modell 0.2 betragen soll, muss MinScore $= [0.9, 0.1, 0.8, 0.2]$ angegeben werden.

NumMatches: Mit NumMatches kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als NumMatches Instanzen eine Bewertung größer als MinScore erreichen, werden nur die besten NumMatches Instanzen zurückgeliefert. Falls weniger als NumMatches Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter MinScore hat Vorrang vor NumMatches. Sollen alle Modellinstanzen, deren Bewertung MinScore übersteigt, im Bild gefunden werden, muss NumMatches auf *0* gesetzt werden.

Beim Tracken der Matches durch die Bildpyramide werden auf jeder Ebene Matches verworfen, die wenig vielversprechend sind. Dadurch ist es möglich, dass Matches verworfen werden, die auf der niedrigsten Pyramidenebene einen höheren Score gehabt hätten. Beispielsweise kann das dazu führen, dass der gefundene Match für NumMatches gleich *1* nicht mit dem Match mit dem höchsten Score übereinstimmt, der von NumMatches gleich *0* oder > *1* zurückgegeben wird.

Falls erwartet wird, dass mehrere Objekte mit einem ähnlichen Score gefunden werden können, aber nur das Modell mit dem höchsten Score zurückgegeben werden soll, kann es vorteilhaft sein, NumMatches zu erhöhen, und dann den Match mit dem höchsten Score auszuwählen.

Für den Fall, dass das Formmodell mit set_shape_model_clutter um Störparameter erweitert wurde, berücksichtigt NumMatches auch den zweiten Wert, der in MinScore übergeben wird. Gefundene Instanzen weisen in diesem Fall einen Score auf, der größer als der erste Eintrag ist, sowie einen Störwert der niedriger als der zweite Eintrag ist. Ist die Anzahl gefundener Instanzen größer als NumMatches, werden die besten NumMatches Instanzen in Bezug auf ihren Störwert zurückgegeben. MinScore hat weiterhin Vorrang vor NumMatches und NumMatches muss weiterhin auf *0* gesetzt werden um alle Modellinstanzen zu erhalten, die die Bedingungen von MinScore erfüllen. Es ist zu beachten, dass beim Tracken der Matches durch die Bildpyramide keine Matches verworfen werden, wenn Störparameter verwendet werden. Die Laufzeit mit Nutzung von Störparametern ist daher mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter NumMatches.

Enthält NumMatches einen Wert, liefert find_scaled_shape_models die NumMatches besten Instanzen des Modells unabhängig von der Art des Modells zurück. Falls z.B. in ModelIDs zwei Modelle übergeben werden und NumMatches $= 2$ gewählt wird, kann es sein, dass zwei Instanzen des ersten Modells und keine des zweiten Modells, eine Instanz des ersten Modells und eine des zweiten Modells oder keine Instanz des ersten Modells und zwei des zweiten Modells zurückgeliefert werden. Falls hingegen NumMatches mehrere Werte enthält, werden so viele Instanzen des jeweiligen Modells in ModelIDs zurückgeliefert, wie durch das entsprechende Element von NumMatches angegeben. Falls z.B. NumMatches $= [1, 1]$ gewählt wird, wird eine Instanz des ersten Modells und eine des zweiten Modells zurückgeliefert.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter MaxOverlap kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als MaxOverlap überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe smallest_rectangle2). Bei MaxOverlap $= 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei MaxOverlap $= 1$ werden alle gefundenen Instanzen zurückgeliefert.

Falls MaxOverlap einen Wert enthält, wird die Überlappung für alle gefundenen Instanzen des Modells unabhängig von der Art des Modells berechnet. D.h., dass sich zu viel überlappende Instanzen verschiedener und gleicher Modelle eliminiert werden. Falls hingegen mehrere Werte in MaxOverlap übergeben werden, wird die Überlappung nur innerhalb der gefundenen Instanzen des jeweiligen Typs der Modelle in ModelIDs berechnet. D.h., es werden nur sich zu viel überlappende Instanzen gleicher Modelle eliminiert. Modelle verschiedenen Typs können sich in diesem Modus vollständig überlappen.

SubPixel: Der Parameter SubPixel gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls SubPixel auf *'none'* (oder *'false'* für Rückwärtskompatibilität) gesetzt wird, wird die Lage des Modells nur pixelgenau bzw. mit der bei create_scaled_shape_model angegebenen Winkel- und Skalierungsauflösung bestimmt. Falls SubPixel auf *'interpolation'* (oder *'true'*) gesetzt wird, werden sowohl die Position als auch die Rotation und Skalierung subpixelgenau bestimmt. Dabei wird die Lage des Modells anhand der Score-Funktion interpoliert. Dieser Modus kostet fast keine Rechenzeit und liefert in den meisten Anwendungen eine ausreichende Genauigkeit. In manchen Anwendungen ist eine möglichst hohe Genauigkeit wichtig. In diesen Fällen kann die Lage des Modells durch Ausgleichsrechnung, d.h., durch Minimierung der Abstände der Modellpunkte und der zugehörigen Bildpunkte (Least-Squares Adjustment), bestimmt werden. Im Gegensatz zu *'interpolation'* kostet dieser Modus zusätzliche Rechenzeit. Mit den verschiedenen Modi (*'least_squares'*, *'least_squares_high'* und *'least_squares_very_high'*) kann dabei die Genauigkeit festgelegt werden, mit der der minimale Abstand gesucht wird. Je höher die Genauigkeit gewählt wird, desto länger dauert allerdings auch die Subpixel-Extraktion. Im Normalfall sollte für SubPixel *'interpolation'* gewählt werden. Falls die Ausgleichsrechnung gewünscht wird, sollte *'least_squares'* gewählt werden, da sich so der optimale Kompromiss zwischen Laufzeit und Genauigkeit ergibt.

Objekte, die im Vergleich zum Modell leicht deformiert im Bild erscheinen, können in manchen Fällen entweder nicht oder nur mit einer geringen Genauigkeit gefunden werden. Für solche Objekte besteht die Möglichkeit, im Parameter SubPixel zusätzlich die maximal zulässige Objektdeformation in Pixeln anzugeben. Dies kann mit Hilfe des optionalen Parameterwertes *'max_deformation '* und einer nachfolgenden Ganzzahl zwischen *0* und *32* (im selben String), die die maximale Deformation beschreibt, erfolgen. Kann die Form des Objektes z.B. bis zu 2 Pixel von der im Modell gespeicherten Form abweichen, muss im Parameter SubPixel der Wert *'max_deformation 2'* zusätzlich zum oben beschriebenen Modus für die subpixelgenaue Extraktion übergeben werden, also z.B. *['least_squares', 'max_deformation 2']*. Mit *'max_deformation 0'* wird das Objekt ohne Deformationen gesucht, also so als ob man kein *'max_deformation '* übergeben hätte. Zu beachten ist, dass größere Werte für die maximale Deformation oft zu erhöhten Laufzeiten führen. Außerdem besteht bei einem zu großen Wert die Gefahr, dass falsche Kandidaten gefunden werden. Beide Probleme können hauptsächlich bei kleinen Objekten oder Objekten mit feinen Strukturen auftreten. Der Grund hierfür ist, dass solche Objekte durch große erlaubte Deformationen ihre für die Suche wichtige charakteristische Form verlieren. Zu beachten ist auch, dass die Genauigkeit von teilweise verdeckten Objekten für höhere Deformationen abnehmen kann wenn Störobjekte in der Nähe des Objektes vorhanden sind. Die maximale Deformation sollte demnach immer nur so groß wie nötig und so gering wie möglich gewählt werden. Näherungsweise rotationssymmetrische Objekte könnten nicht gefunden werden, falls *'max_deformation'* und AngleExtent beide auf einen Wert größer 0 gesetzt sind. In dem Fall kann es zu Mehrdeutigkeiten kommen, die nicht aufgelöst werden können, sodass der Match als falsch verworfen wird. Wenn das passiert, versuchen Sie, entweder *'max_deformation'* oder AngleExtent auf 0 zu setzen oder das Modell so zu verändern, dass weniger Symmetrien auftreten. Die Score-Berechnung bei Deformationswerten größer *0* hängt von der Art der Subpixel-Extraktion ab. In den meisten Fällen wird sich der Score eines Matches verändern, wenn man *'least_squares'*, *'least_squares_high'* oder *'least_squares_very_high'* (siehe oben) wählt (im Vergleich zu *'none'* oder *'interpolation'*). Außerdem wird sich der Score in der Regel erhöhen, wenn man bei einer der Least-Squares-Adjustment-Methoden die maximale Deformation erhöht, da dann zu den Modellpunkten mehr zugehörige Bildpunkte gefunden werden können. Für einen aussagekräftigen Score und zum Vermeiden falscher Matches wird empfohlen, das Erlauben einer Deformation mit einer Subpixel-Extraktion mit einer der Least-Squares-Adjustment-Methoden zu kombinieren.

Falls die Subpixel-Extraktion und/oder die maximale Objektdeformation für jedes Modell einzeln spezifiziert werden sollen, muss in SubPixel die Subpixel-Extraktion genauso oft angegeben werden, wie die Anzahl von Elementen in ModelIDs. Nach jedem übergebenen Wert für die Subpixel-Extraktion kann optional ein zweiter Wert übergeben werden, der die maximale Objektdeformation für das jeweilige Modell beschreibt. Wird nach dem Wert für die Subpixel-Extraktion kein Wert für die Objektdeformation übergeben, wird das entsprechende Modell ohne Deformation gesucht. Falls z.B. zwei Modelle in ModelIDs angegeben sind und für das erste Modell die Subpixel-Extraktion *'interpolation'* ohne Objektdeformation gewählt werden soll und für das zweite Modell die Subpixel-Extraktion *'least_squares'* gewünscht ist sowie eine maximale Objektdeformation von *3* Pixeln, dann kann für den Parameter SubPixel das Tupel *['interpolation', 'least_squares', 'max_deformation 3']* übergeben werden. Alternativ kann auch das äquivalente Tupel *['interpolation', 'max_deformation 0', 'least_squares', 'max_deformation 3']* übergeben werden.

NumLevels: Mit NumLevels wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit create_scaled_shape_model angegebenen Bereich beschnitten. Falls NumLevels als *0* angegeben wird, wird die mit create_scaled_shape_model angegebene Anzahl verwendet.

In manchen Fällen kann es sein, dass die Anzahl der Pyramidenebenen, die beispielsweise automatisch mit

create_scaled_shape_model ermittelt wurde, zu hoch ist. Dann werden eventuell Instanzen, die eine sehr hohe finale Bewertung gehabt hätten, bereits auf der höchsten Pyramidenebene ausgeschlossen und damit nicht gefunden. Anstatt nun MinScore auf einen sehr geringen Wert zu setzen, um alle Matches zu finden, kann der Wert von NumLevels mit get_shape_model_params ermittelt werden und dann ein etwas geringerer Wert in find_scaled_shape_models genutzt werden. Diese Herangehensweise führt oft zu besseren Ergebnissen hinsichtlich Geschwindigkeit und Robustheit.

Optional kann NumLevels einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, Laufzeit einzusparen. Allerdings ist in diesem Modus im Allgemeinen die Genauigkeit der gefundenen Lageparameter geringer als im Normalfall, in dem die Matches bis auf die unterste Pyramidenebene verfolgt werden. Falls eine hohe Genauigkeit erzielt werden soll, empfiehlt es sich also, mindestens *'least_squares'* für SubPixel zu verwenden. Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden.

Falls die unterste Pyramidenebene für jedes Modell einzeln spezifiziert werden soll, müssen in NumLevels zweimal die Anzahl von Elementen in ModelIDs angegeben werden. Dabei sind die Anzahl der Pyramidenebenen und der untersten Pyramidenebenen verschränkt anzugeben. Falls z.B. zwei Modelle in ModelIDs angegeben sind, die oberste Pyramidenebene für das erste Modell 5 und für das zweite Modell 4 sein soll und die unterste Pyramidenebene für das erste Modell 2 und für das zweite 1 sein soll, so ist NumLevels $= [5, 2, 4, 1]$ zu wählen. Falls genau zwei Modelle in ModelIDs angegeben sind, ergibt sich als Spezialfall, dass, falls die unterste Pyramidenebene spezifiziert werden soll, die oberste und unterste Pyramidenebene für beide Modelle explizit spezifiziert werden muss, selbst wenn sie gleich sind. Ein Tupel der Länge zwei in NumLevels wird in diesem Fall nämlich als Spezifikation der obersten Pyramidenebene für die zwei Modelle interpretiert.

In mangelhaften Eingabebildern, d.h., in Bildern die z.B. unscharf, deformiert oder verrauscht sind, kann auf der untersten Pyramidenebene oft keine Instanz des Formmodells gefunden werden, da aufgrund der mangelhaften Bildqualität nicht genügend Kanteninformation vorhanden ist, oder die Kanten zu stark deformiert sind. Auf höheren Pyramidenebenen dagegen kann die Kanteninformation noch ausreichend sein um Instanzen des Modells zu finden, wobei auch hierbei die oben beschriebenen Einschränkungen bezüglich Genauigkeit und Robustheit gelten. Die Wahl der passenden Pyramidenebene, d.h., der untersten Pyramidenebene auf der noch eine Instanz des Modells gefunden wird, hängt sowohl vom Modell als auch vom Eingabebild ab. Diese Pyramidenebene kann also von Bild zu Bild variieren. Um das Matching auf mangelhaften Bildern zu erleichtern, kann die unterste Pyramidenebene, auf der noch eine Instanz des Modells gefunden wird, während des Matchings automatisch bestimmt werden. Um diesen als 'increased tolerance mode' bezeichneten Mechanismus zu aktivieren, muss die unterste Pyramidenebene in NumLevels negativ angegeben werden. Wird z.B. NumLevels $= [5, 2, 4, -1]$ gesetzt, so ist die unterste Pyramidenebene für das erste Modell 2. Kann auf der Pyramidenebene 2 keine Instanz des ersten Formmodells gefunden werden, so wird für dieses Modell kein Ergebnis zurückgegeben. Für das zweite Modell ist in diesem Beispiel die unterste Pyramidenebene auf *-1* gesetzt. Daher wird versucht, eine Instanz des Formmodells auf der Pyramidenebene 1 zu finden. Kann auf dieser Pyramidenebene keine Instanz gefunden werden, so wird die unterste Pyramidenebene ermittelt, auf der noch Instanzen des Modells gefunden wurden. Die Instanzen dieser Pyramidenebene werden dann als Ergebnis zurückgegeben.

Wenn ein Modell mit adapt_shape_model_high_noise angepasst wurde wird die geschätzte untere Pyramidenstufe per Standardeinstellung verwendet. Der Benutzer kann den geschätzten Wert dennoch überschreiben, indem er jeweils zwei Werte an NumLevels übergibt und damit explizit die untere Pyramidenstufe setzt.

Greediness: Der Parameter Greediness bestimmt, wie „gierig" die Suche durchgeführt werden soll. Für Greediness $= 0$ wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden, immer findet, wenn die anderen Parameter passend gesetzt sind. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für Greediness $= 1$ wird eine unsicherere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, dass das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für Greediness $= 1$ wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten Fällen wird das Formmodell für Greediness $= 0.9$ immer sicher gefunden.

Ausgabeparameter im Detail

Row, Column, Angle und Scale: Die Position, Rotation und Skalierung der gefundenen Instanzen des Modells werden in `Row`, `Column`, `Angle` und `Scale` zurückgeliefert. Die Koordinaten `Row` und `Column` sind die Koordinaten des Ursprungs des Modells im Suchbild. Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit `create_scaled_shape_model` verwendet wurde. Ein anderer Ursprung kann mit `set_shape_model_origin` festgelegt werden.

Beachten Sie, dass die Koordinaten `Row` und `Column` nicht exakt mit dem Ursprung des Modells übereinstimmen und daher nicht direkt verwendet werden sollten. Die Werte sind dafür optimiert, die Transformationsmatrix zu erzeugen, mit der die Matching-Ergebnisse für verschiedene Aufgaben verwendet werden können, z.B. um ROIs für andere Bildverarbeitungsschritte einem Objekt nachzuführen. Das bei `find_scaled_shape_model` angegebene Beispiel zeigt, wie diese Matrix erzeugt wird und wie man damit das Modell an der gefundenen Position im Suchbild visualisiert.

Beachten Sie, dass die Visualisierung aber auch mit der Prozedur dev_display_shape_matching_results realisiert werden kann.

Score: In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Falls das Formmodell mit `set_shape_model_clutter` um Störparameter erweitert wurde, gibt `Score`, nach den oben genannten Werten, auch den Störwert jeder gefundenen Instanz zurück. Falls beispielsweise die Hälfte der Störregion Störkanten aufweist entspricht der Störwert 0.5. Werden z.B. zwei Instanzen gefunden, wobei für Erstere eine Bewertung von 0.9 und für die zweite 0.8 erreicht wird und der Störwert der ersten Instanz 0.2, der zweiten Instanz 0.1 ist, wird `Score` $= [0.9, 0.8, 0.2, 0.1]$ zurückgegeben. Es ist zu beachten, dass von sämtlichen Ergebnissen des formbasierten Matchings die Störwerte am stärksten von Beleuchtungsänderungen beeinflusst werden.

Model: Der gefundene Modelltyp wird in `Model` zurückgeliefert. Die Elemente von `Model` sind ein Index in das Tupel `ModelIDs`, d.h. sie können Werte von 0 bis $|$`ModelIDs`$| - 1$ enthalten. Ein Wert von 0 in einem Element von `Model` entspricht also einer Instanz des ersten in `ModelIDs` übergebenen Modells.

Setzen eines Timeout

Mittels des Operators `set_shape_model_param` können Sie einen *'timeout'* für `find_scaled_shape_models` angeben. Falls die durch `ModelIDs` referenzierten Modelle unterschiedliche Werte für *'timeout'* besitzen, wählt `find_scaled_shape_models` den niedrigsten aus. Wenn `find_scaled_shape_models` diesen *'timeout'* erreicht endet er ohne Ergebnis und gibt den Fehlercode 9400 (H_ERR_TIMEOUT) zurück. Abhängig von der Größe des Skalierungsintervalls, das durch `ScaleMin` und `ScaleMax` festgelegt wird, benötigt `find_scaled_shape_models` eine signifikante Zeitspanne um den Speicher für gecachte Transformationen freizugeben, falls das Modell nicht vorberechnet wurde. Da diese Transformationen auch freigegeben werden müssen nachdem ein Timeout auftritt, überschreitet `find_scaled_shape_models` den in *'timeout'* angegebenen Wert um diese Zeitspanne.

MinContrast bei mehreren Modellen

Wenn über den Parameter `ModelIDs` mehrere Modelle übergeben werden, sollten diese mit dem gleichen Wert für `MinContrast` erzeugt worden sein. Falls sie mit unterschiedlichen Werten für `MinContrast` erzeugt worden sind, wird in `find_scaled_shape_models` der kleinste dieser Werte verwendet.

Darstellung der Resultate

Die Verwendung der Prozedur `dev_display_shape_matching_results` wird zur Darstellung der Resultate formbasierten Matchings stark empfohlen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Eingabebild, in dem die Modelle gefunden werden sollen.

▷ **`ModelIDs`** (input_control) . . . . . shape_model(-array) ⇝ *handle*
Handle der Modelle.

▷ **AngleStart** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Kleinste auftretende Rotation der Modelle.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.78
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
Restriktion: AngleExtent >= 0

▷ **ScaleMin** (input_control) . . . . . number(-array) ⇝ *real*
Kleinste auftretende Skalierung der Modelle.
Defaultwert: 0.9
Wertevorschläge: `ScaleMin` ∈ {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: ScaleMin > 0

▷ **ScaleMax** (input_control) . . . . . number(-array) ⇝ *real*
Größte auftretende Skalierung der Modelle.
Defaultwert: 1.1
Wertevorschläge: `ScaleMax` ∈ {1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Restriktion: ScaleMax >= ScaleMin

▷ **MinScore** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen der Modelle.
Defaultwert: 0.5
Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumMatches** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der zu findenden Instanzen der Modelle (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real(-array) ⇝ *real*
Maximale Überlappung der zu findenden Instanzen der Modelle.
Defaultwert: 0.5
Wertevorschläge: `MaxOverlap` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ `MaxOverlap` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **SubPixel** (input_control) . . . . . string(-array) ⇝ *string*
Subpixelgenauigkeit falls ungleich *'none'*.
Defaultwert: 'least_squares'
Wertevorschläge: `SubPixel` ∈ {'none', 'interpolation', 'least_squares', 'least_squares_high', 'least_squares_very_high', 'max_deformation 1', 'max_deformation 2', 'max_deformation 3', 'max_deformation 4', 'max_deformation 5', 'max_deformation 6'}

▷ **NumLevels** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls $|$`NumLevels`$| = 2$).
Defaultwert: 0
Werteliste: `NumLevels` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real(-array) ⇝ *real*
„Gierigkeit“ der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen“ werden).
Defaultwert: 0.9
Wertevorschläge: `Greediness` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ `Greediness` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate der gefundenen Instanzen der Modelle.

▷ **Column** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinate der gefundenen Instanzen der Modelle.

▷ **Angle** (output_control) ..... angle.rad-array ⇝ *real*
Rotationswinkel der gefundenen Instanzen der Modelle.

▷ **Scale** (output_control) ..... number-array ⇝ *real*
Skalierung der gefundenen Instanzen der Modelle.

▷ **Score** (output_control) ..... real-array ⇝ *real*
Bewertung der gefundenen Instanzen der Modelle.

▷ **Model** (output_control) ..... integer-array ⇝ *integer*
Index der gefundenen Instanzen der Modelle.

Beispiel

```
read_image (Image, 'pcb_focus/pcb_focus_telecentric_061')
gen_rectangle1 (ROI_0, 236, 241, 313, 321)
gen_circle (ROI_1, 281, 653, 41)
reduce_domain (Image, ROI_0, ImageReduced1)
reduce_domain (Image, ROI_1, ImageReduced2)
create_scaled_shape_model (ImageReduced1,'auto', -0.39, 0.79, 'auto', 0.9, \
                           1.1, 'auto', 'auto', 'use_polarity', 'auto', \
                           'auto', ModelID1)
create_scaled_shape_model (ImageReduced2, 'auto', -0.39, 0.79, 'auto', 0.9,\
                           1.1, 'auto', 'auto', 'use_polarity', 'auto', \
                           'auto', ModelID2)
ModelIDs:=[ModelID1, ModelID2]
find_scaled_shape_models (Image, ModelIDs, -0.39, 0.78, 0.9, 1.1, 0.5, 1, \
                          0.5, 'least_squares', 0, 0.9, Row, Column, Angle,\
                          Scale, Score, Model)
* Display results
dev_display_shape_matching_results (ModelIDs, 'red', Row, Column, Angle, \
                                    Scale, 1, Model)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_scaled_shape_models` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`add_channels`, `create_scaled_shape_model`, `read_shape_model`, `set_shape_model_origin`, `set_shape_model_clutter`

Nachfolger

`clear_shape_model`

Alternativen

`find_shape_models`, `find_aniso_shape_models`, `find_shape_model`, `find_scaled_shape_model`, `find_aniso_shape_model`, `find_ncc_model`, `find_ncc_models`

Siehe auch

`set_system`, `get_system`, `set_shape_model_param`

Modul

Matching

find_shape_model (Image : : ModelID, AngleStart, AngleExtent, MinScore, NumMatches, MaxOverlap, SubPixel, NumLevels, Greediness : Row, Column, Angle, Score)

Suche der besten Matches eines Formmodells in einem Bild.

find_shape_model findet die besten NumMatches Instanzen des Formmodells ModelID im Eingabebild Image. Das Modell muss zuvor mit create_shape_model erzeugt oder mit read_shape_model eingelesen worden sein.

Die Position und Rotation der gefundenen Instanzen des Modells wird in Row, Column und Angle zurückgeliefert. In Score wird die Bewertung der gefundenen Instanzen zurückgegeben. Nähere Informationen finden Sie in den parameterspezifischen Abschnitten.

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes Image gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_shape_model verwendet wurde. Ein eventuell mit set_shape_model_origin anders gesetzter Ursprung wird nicht berücksichtigt. Im Normalfall wird das Modell innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als MinScore erreichen würde (siehe unten). Außerdem kann es sein, dass das Modell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit set_system('border_shape_models','true') für alle Modelle oder mit set_shape_model_param(ModelID,'border_shape_models','true') speziell für ein Modell umgestellt werden, so dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als MinScore erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit dilation_circle zu vergrößern.

AngleStart und AngleExtent: Die Parameter AngleStart und AngleExtent legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Der Winkelbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit create_shape_model angegeben worden ist. Dies bedeutet insbesondere, dass die Winkelbereiche des Modells und der Suche sich überlappen müssen.

Beachten Sie, dass in einigen Fällen Instanzen mit einer Rotation gefunden werden, die geringfügig außerhalb des angegebenen Bereichs liegen. Dies kann vorkommen, wenn der angegebene Bereich kleiner ist als der bei der Erstellung des Modells angegebene Bereich. AngleStart und AngleExtent werden nur auf der höchsten Pyramidenstufe überprüft. Matches, die auf der höchsten Pyramidenstufe gefunden werden, werden bis zur untersten Pyramidenstufe verfeinert. Aus Leistungsgründen wird bei der Verfeinerung jedoch nicht mehr überprüft, ob die Matches immer noch innerhalb des angegebenen Bereichs liegen.

MinScore: Der Parameter MinScore legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von MinScore gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, dass das Modell niemals verdeckt wird, kann MinScore auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden. Werden die Matches nicht bis zur untersten Pyramidenebene verfolgt (siehe unten), kann es in manchen Fällen vorkommen, dass Instanzen gefunden werden, deren Score geringfügig unter dem Wert von MinScore liegen.

Falls das Formmodell mit set_shape_model_clutter um Störparameter erweitert wurde und dadurch *'use_clutter'* aktiviert wurde, erwartet MinScore einen zweiten Wert, welcher den maximalen Störwert eines potentiellen Matches angibt, um als Instanz des Modells im Bild betrachtet werden zu können. Die Laufzeit mit Nutzung von Störparametern ist mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter NumMatches.

NumMatches: Mit NumMatches kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als NumMatches Instanzen eine Bewertung größer als MinScore erreichen, werden nur die besten NumMatches Instanzen zurückgeliefert. Falls weniger als NumMatches Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter MinScore hat

Vorrang vor NumMatches. Sollen alle Modellinstanzen, deren Bewertung MinScore übersteigt, im Bild gefunden werden, muss NumMatches auf *0* gesetzt werden.

Beim Tracken der Matches durch die Bildpyramide werden auf jeder Ebene Matches verworfen, die wenig vielversprechend sind. Dadurch ist es möglich, dass Matches verworfen werden, die auf der niedrigsten Pyramidenebene einen höheren Score gehabt hätten. Beispielsweise kann das dazu führen, dass der gefundene Match für NumMatches gleich *1* nicht mit dem Match mit dem höchsten Score übereinstimmt, der von NumMatches gleich *0* oder > *1* zurückgegeben wird.

Falls erwartet wird, dass mehrere Objekte mit einem ähnlichen Score gefunden werden können, aber nur das Modell mit dem höchsten Score zurückgegeben werden soll, kann es vorteilhaft sein, NumMatches zu erhöhen, und dann den Match mit dem höchsten Score auszuwählen.

Für den Fall, dass das Formmodell mit set_shape_model_clutter um Störparameter erweitert wurde, berücksichtigt NumMatches auch den zweiten Wert, der in MinScore übergeben wird. Gefundene Instanzen weisen in diesem Fall einen Score auf, der größer als der erste Eintrag ist, sowie einen Störwert der niedriger als der zweite Eintrag ist. Ist die Anzahl gefundener Instanzen größer als NumMatches, werden die besten NumMatches Instanzen in Bezug auf ihren Störwert zurückgegeben. MinScore hat weiterhin Vorrang vor NumMatches und NumMatches muss weiterhin auf *0* gesetzt werden um alle Modellinstanzen zu erhalten, die die Bedingungen von MinScore erfüllen. Es ist zu beachten, dass beim Tracken der Matches durch die Bildpyramide keine Matches verworfen werden, wenn Störparameter verwendet werden. Die Laufzeit mit Nutzung von Störparametern ist daher mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter NumMatches.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter MaxOverlap kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als MaxOverlap überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe smallest_rectangle2). Bei MaxOverlap $= 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei MaxOverlap $= 1$ werden alle gefundenen Instanzen zurückgeliefert.

SubPixel: Der Parameter SubPixel gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls SubPixel auf *'none'* (oder *'false'* für Rückwärtskompatibilität) gesetzt wird, wird die Lage des Modells nur pixelgenau bzw. mit der bei create_shape_model angegebenen Winkelauflösung bestimmt. Falls SubPixel auf *'interpolation'* (oder *'true'*) gesetzt wird, werden sowohl die Position als auch die Rotation subpixelgenau bestimmt. Dabei wird die Lage des Modells anhand der Score-Funktion interpoliert. Dieser Modus kostet fast keine Rechenzeit und liefert in den meisten Anwendungen eine ausreichende Genauigkeit. In manchen Anwendungen ist eine möglichst hohe Genauigkeit wichtig. In diesen Fällen kann die Lage des Modells durch Ausgleichsrechnung, d.h., durch Minimierung der Abstände der Modellpunkte und der zugehörigen Bildpunkte (Least-Squares Adjustment), bestimmt werden. Im Gegensatz zu *'interpolation'* kostet dieser Modus zusätzliche Rechenzeit. Mit den verschiedenen Modi (*'least_squares'*, *'least_squares_high'* und *'least_squares_very_high'*) kann dabei die Genauigkeit festgelegt werden, mit der der minimale Abstand gesucht wird. Je höher die Genauigkeit gewählt wird, desto länger dauert allerdings auch die Subpixel-Extraktion. Im Normalfall sollte für SubPixel *'interpolation'* gewählt werden. Falls die Ausgleichsrechnung gewünscht wird, sollte *'least_squares'* gewählt werden, da sich so der optimale Kompromiss zwischen Laufzeit und Genauigkeit ergibt.

Objekte, die im Vergleich zum Modell leicht deformiert im Bild erscheinen, können in manchen Fällen entweder nicht oder nur mit einer geringen Genauigkeit gefunden werden. Für solche Objekte besteht die Möglichkeit, im Parameter SubPixel zusätzlich die maximal zulässige Objektdeformation in Pixeln anzugeben. Dies kann mit Hilfe des optionalen Parameterwertes *'max_deformation '* und einer nachfolgenden Ganzzahl zwischen *0* und *32* (im selben String), die die maximale Deformation beschreibt, erfolgen. Kann die Form des Objektes z.B. bis zu 2 Pixel von der im Modell gespeicherten Form abweichen, muss im Parameter SubPixel der Wert *'max_deformation 2'* zusätzlich zum oben beschriebenen Modus für die subpixelgenaue Extraktion übergeben werden, also z.B. *['least_squares', 'max_deformation 2']*. Mit *'max_deformation 0'* wird das Objekt ohne Deformationen gesucht, also so als ob man kein *'max_deformation '* übergeben hätte. Zu beachten ist, dass größere Werte für die maximale Deformation oft zu erhöhten Laufzeiten führen. Außerdem besteht bei einem zu großen Wert die Gefahr, dass falsche Kandidaten gefunden werden. Beide Probleme können hauptsächlich bei kleinen Objekten oder Objekten mit feinen Strukturen auftreten. Der Grund hierfür ist, dass solche Objekte durch große erlaubte Deformationen ihre für die Suche wichtige charakteristische Form verlieren. Zu beachten ist auch, dass die Genauigkeit von teilweise verdeckten Objekten für höhere

Deformationen abnehmen kann wenn Störobjekte in der Nähe des Objektes vorhanden sind. Die maximale Deformation sollte demnach immer nur so groß wie nötig und so gering wie möglich gewählt werden. Näherungsweise rotationssymmetrische Objekt könnten nicht gefunden werden, falls *'max_deformation'* und `AngleExtent` beide auf einen Wert größer 0 gesetzt sind. In dem Fall kann es zu Mehrdeutigkeiten kommen, die nicht aufgelöst werden können, sodass der Match als falsch verworfen wird. Wenn das passiert, versuchen Sie, entweder *'max_deformation'* oder `AngleExtent` auf 0 zu setzen oder das Modell so zu verändern, dass weniger Symmetrien auftreten. Die Score-Berechnung bei Deformationswerten größer *0* hängt von der Art der Subpixel-Extraktion ab. In den meisten Fällen wird sich der Score eines Matches verändern, wenn man *'least_squares'*, *'least_squares_high'* oder *'least_squares_very_high'* (siehe oben) wählt (im Vergleich zu *'none'* oder *'interpolation'*). Außerdem wird sich der Score in der Regel erhöhen wenn man bei einer der Least-Squares-Adjustment-Methoden die maximale Deformation erhöht, da dann zu den Modellpunkten mehr zugehörige Bildpunkte gefunden werden können. Für einen aussagekräftigen Score und zum Vermeiden falscher Matches wird empfohlen, das Erlauben einer Deformation mit einer Subpixel-Extraktion mit einer der Least-Squares-Adjustment-Methoden zu kombinieren.

NumLevels: Mit `NumLevels` wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit `create_shape_model` angegebenen Bereich beschnitten. Falls `NumLevels` als *0* angegeben wird, wird die mit `create_shape_model` angegebene Anzahl verwendet.

In manchen Fällen kann es sein, dass die Anzahl der Pyramidenebenen, die beispielsweise automatisch mit `create_shape_model` ermittelt wurde, zu hoch ist. Dann werden eventuell Instanzen, die eine sehr hohe finale Bewertung gehabt hätten, bereits auf der höchsten Pyramidenebene ausgeschlossen und damit nicht gefunden. Anstatt nun `MinScore` auf einen sehr geringen Wert zu setzen, um alle Matches zu finden, kann der Wert von `NumLevels` mit `get_shape_model_params` ermittelt werden und dann ein etwas geringerer Wert in `find_shape_model` genutzt werden. Diese Herangehensweise führt oft zu besseren Ergebnissen hinsichtlich Geschwindigkeit und Robustheit.

Optional kann `NumLevels` einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, Laufzeit einzusparen. Allerdings ist in diesem Modus im Allgemeinen die Genauigkeit der gefundenen Lageparameter geringer als im Normalfall, in dem die Matches bis auf die unterste Pyramidenebene verfolgt werden. Falls eine hohe Genauigkeit erzielt werden soll, empfiehlt es sich also, mindestens *'least_squares'* für `SubPixel` zu verwenden. Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden.

In mangelhaften Eingabebildern, d.h., in Bildern die z.B. unscharf, deformiert oder verrauscht sind, kann auf der untersten Pyramidenebene oft keine Instanz des Formmodells gefunden werden, da aufgrund der mangelhaften Bildqualität nicht genügend Kanteninformation vorhanden ist, oder die Kanten zu stark deformiert sind. Auf höheren Pyramidenebenen dagegen kann die Kanteninformation noch ausreichend sein um Instanzen des Modells zu finden, wobei auch hierbei die oben beschriebenen Einschränkungen bezüglich Genauigkeit und Robustheit gelten. Die Wahl der passenden Pyramidenebene, d.h., der untersten Pyramidenebene auf der noch eine Instanz des Modells gefunden wird, hängt sowohl vom Modell als auch vom Eingabebild ab. Diese Pyramidenebene kann also von Bild zu Bild variieren. Um das Matching auf mangelhaften Bildern zu erleichtern, kann die unterste Pyramidenebene, auf der noch eine Instanz des Modells gefunden wird, während des Matchings automatisch bestimmt werden. Um diesen als 'increased tolerance mode' bezeichneten Mechanismus zu aktivieren, muss die unterste Pyramidenebene in `NumLevels` negativ angegeben werden. Wird z.B. `NumLevels` $= [4, -2]$ gesetzt, so wird das Matching auf der vierten Pyramidenebene begonnen und auf der zweituntersten Pyramidenebene beendet. Es wird also versucht, eine Instanz des Formmodells auf der Pyramidenebene 2 zu finden. Kann auf dieser Pyramidenebene keine Instanz gefunden werden, so wird die unterste Pyramidenebene ermittelt, auf der noch Instanzen des Modells gefunden wurden. Die Instanzen dieser Pyramidenebene werden dann als Ergebnis zurückgegeben.

Wenn `ModelID` mit `adapt_shape_model_high_noise` angepasst wurde, wird die geschätzte untere Pyramidenstufe per Standardeinstellung verwendet. Der Benutzer kann den geschätzten Wert dennoch überschreiben, indem er zwei Werte an `NumLevels` übergibt und damit explizit die untere Pyramidenstufe setzt.

Greediness: Der Parameter `Greediness` bestimmt, wie „gierig“ die Suche durchgeführt werden soll. Für `Greediness` $= 0$ wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden,

immer findet, wenn die anderen Parameter passend gesetzt sind. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für Greediness = 1 wird eine unsicherere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, dass das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für Greediness = 1 wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten Fällen wird das Formmodell für Greediness = 0.9 immer sicher gefunden.

Ausgabeparameter im Detail

Row, Column und Angle: Die Position und Rotation der gefundenen Instanzen des Modells wird in Row, Column und Angle zurückgeliefert. Die Koordinaten Row und Column beschreiben die Position des Ursprungs des Modells im Suchbild. Die Werte sind allerdings bereits geringfügig angepasst, so dass sie direkt zur Erstellung einer Transformationsmatrix verwendet werden können, die für ein Alignment oder die Visualisierung der Modellkanten benötigt wird. Diese Anpassung der Koordinaten hat mit der Art und Weise zu tun, in der HALCON ikonische Objekte transformiert, siehe hierzu affine_trans_pixel. Das Beispiel weiter unten zeigt, wie die Transformationsmatrix erzeugt wird und wie man damit das Modell an der gefundenen Position im Suchbild visualisiert.

Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_shape_model verwendet wurde. Ein anderer Ursprung kann mit set_shape_model_origin festgelegt werden.

Score: In Score wird die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Falls das Formmodell mit set_shape_model_clutter um Störparameter erweitert wurde, gibt Score, nach den oben genannten Werten, auch den Störwert jeder gefundenen Instanz zurück. Falls beispielsweise die Hälfte der Störregion Störkanten aufweist entspricht der Störwert 0.5. Werden z.B. zwei Instanzen gefunden, wobei für Erstere eine Bewertung von 0.9 und für die zweite 0.8 erreicht wird und der Störwert der ersten Instanz 0.2, der zweiten Instanz 0.1 ist, wird Score = $[0.9, 0.8, 0.2, 0.1]$ zurückgegeben. Es ist zu beachten, dass von sämtlichen Ergebnissen des formbasierten Matchings die Störwerte am stärksten von Beleuchtungsänderungen beeinflusst werden.

Setzen eines Timeout

Mittels des Operators set_shape_model_param können Sie einen *'timeout'* für find_shape_model angeben. Wenn find_shape_model diesen *'timeout'* erreicht endet er ohne Ergebnis und gibt den Fehlercode 9400 (H_ERR_TIMEOUT) zurück.

Darstellung der Resultate

Die Verwendung der Prozedur dev_display_shape_matching_results wird zur Darstellung der Resultate formbasierten Matchings stark empfohlen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
Eingabebild, in dem das Modell gefunden werden soll.

▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
Handle des Modells.

▷ **AngleStart** (input_control) ..... angle.rad ⇝ *real*
Kleinste auftretende Rotation des Modells.
Defaultwert: -0.39
Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) ..... angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: AngleExtent ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
Restriktion: AngleExtent >= 0

▷ **MinScore** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: MinScore ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MinScore ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **NumMatches** (input_control) . . . . . integer ⇝ *integer*
Anzahl der zu findenden Instanzen des Modells (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: NumMatches ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Überlappung der zu findenden Instanzen des Modells.
Defaultwert: 0.5
Wertevorschläge: MaxOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MaxOverlap ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **SubPixel** (input_control) . . . . . string(-array) ⇝ *string*
Subpixelgenauigkeit falls ungleich *'none'*.
Defaultwert: 'least_squares'
Wertevorschläge: SubPixel ∈ {'none', 'interpolation', 'least_squares', 'least_squares_high', 'least_squares_very_high', 'max_deformation 1', 'max_deformation 2', 'max_deformation 3', 'max_deformation 4', 'max_deformation 5', 'max_deformation 6'}

▷ **NumLevels** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls |NumLevels| = 2).
Defaultwert: 0
Werteliste: NumLevels ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real ⇝ *real*
„Gierigkeit" der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen" werden).
Defaultwert: 0.9
Wertevorschläge: Greediness ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ Greediness ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate der gefundenen Instanzen des Modells.

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate der gefundenen Instanzen des Modells.

▷ **Angle** (output_control) . . . . . angle.rad-array ⇝ *real*
Rotationswinkel der gefundenen Instanzen des Modells.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Instanzen des Modells.

Beispiel

```
create_shape_model (ImageReduced, 0, rad(-45), rad(180), 0, \
                    'none', 'use_polarity', 30, 10, ModelID)
get_shape_model_contours (ModelXLD, ModelID, 1)
find_shape_model (SearchImage, ModelID, rad(-45), rad(180), \
                  0.5, 1, 0.5, 'interpolation', \
                  0, 0, Row, Column, Angle, Score)
* Create transformation matrix
vector_angle_to_rigid (0, 0, 0, Row, Column, Angle, HomMat2DObject)
* Display results
dev_display_shape_matching_results (ModelID, 'red', Row, Column, Angle, \
                                    1, 1, 0)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_shape_model` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_shape_model`, `read_shape_model`, `set_shape_model_origin`, `set_shape_model_clutter`

Nachfolger

`clear_shape_model`

Alternativen

`find_scaled_shape_model`, `find_aniso_shape_model`, `find_scaled_shape_models`, `find_shape_models`, `find_aniso_shape_models`, `find_ncc_model`, `find_ncc_models`

Siehe auch

`set_system`, `get_system`, `set_shape_model_param`

Modul

Matching

find_shape_models (Image : : ModelIDs, AngleStart, AngleExtent, MinScore, NumMatches, MaxOverlap, SubPixel, NumLevels, Greediness : Row, Column, Angle, Score, Model)

Suche der besten Matches mehrerer Formmodelle.

`find_shape_models` findet die besten `NumMatches` Instanzen der Formmodelle, die in dem Tupel `ModelIDs` übergeben werden, im Eingabebild `Image`. Die Modelle müssen zuvor mit `create_shape_model` erzeugt oder mit `read_shape_model` eingelesen worden sein. Im Gegensatz zu `find_shape_model` kann also eine Mehrzahl von Modellen gleichzeitig im selben Bild gesucht werden.

Die Position und Rotation der gefundenen Instanzen des Modells wird in `Row`, `Column` und `Angle` zurückgeliefert. In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Der gefundene Modelltyp wird in `Model` zurückgeliefert. Nähere Informationen finden Sie in den parameterspezifischen Abschnitten.

Besonderheiten der Parametersemantik

Gegenüber `find_shape_model` unterscheidet sich die Semantik aller Eingabeparameter etwas. Alle Eingabeparameter müssen entweder genau ein Element enthalten, oder dieselbe Anzahl von Elementen wie `ModelIDs`. (`NumLevels` kann auch zwei oder zweimal die Anzahl von Elementen wie `ModelIDs` enthalten; siehe unten.) Im ersten Fall wird der Wert des Eingabeparameters für alle Modelle gleich verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für das entsprechende Modell in `ModelIDs` verwendet. Zusätzliche Details können in den parameterspezifischen Abschnitten weiter unten gefunden werden. Ein Aufruf von `find_shape_models` mit mehreren Werten für `ModelIDs`, `NumMatches` und `MaxOverlap` entspricht von der Wirkung her also mehreren unabhängigen Aufrufen von `find_shape_model` mit den jeweiligen Parametern, ist allerdings wesentlich effizienter.

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes `Image` gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit `create_shape_model` verwendet wurde. Ein eventuell mit `set_shape_model_origin` anders gesetzter Ursprung wird nicht berücksichtigt. Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das

bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als MinScore erreichen würde (siehe unten). Außerdem kann es sein, dass das Modell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit set_system('border_shape_models','true') für alle Modelle oder mit set_shape_model_param(ModelID,'border_shape_models','true') speziell für ein Modell umgestellt werden, so dass auch Modelle gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als MinScore erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht. Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit dilation_circle zu vergrößern.

Wie üblich wird die Region des Eingabebildes Image als Suchbereich für den Referenzpunkt der Modelle ModelIDs verwendet. Konsistent mit der obigen Semantik kann Image also ein einzelnes Bildobjekt beinhalten oder ein Bildobjekttupel. Falls Image ein einzelnes Bildobjekt enthält, wird dessen Region als Suchbereich für alle Modelle in ModelIDs verwendet. Falls Image mehrere Bildobjekte enthält, wird die jeweilige Region als Suchbereich für das entsprechende Modell aus ModelIDs verwendet. In diesem Fall müssen die Bilder, abgesehen von ihrer Region, identisch sein. D.h. Image kann nicht in beliebiger Weise mit concat_obj aufgebaut werden, sondern muss aus demselben Bild mit add_channels oder äquivalenten Aufrufen erzeugt werden. Falls das nicht der Fall ist, wird eine Fehlermeldung zurückgeliefert.

AngleStart und AngleExtent: Die Parameter AngleStart und AngleExtent legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Der Winkelbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit create_shape_model angegeben worden ist. Dies bedeutet insbesondere, dass die Winkelbereiche des Modells und der Suche sich überlappen müssen.

Beachten Sie, dass in einigen Fällen Instanzen mit einer Rotation gefunden werden, die geringfügig außerhalb des angegebenen Bereichs liegt. Dies kann vorkommen, wenn der angegebene Bereich kleiner ist als der bei der Erstellung des Modells angegebene Bereich. AngleStart und AngleExtent werden nur auf der höchsten Pyramidenstufe überprüft. Matches, die auf der höchsten Pyramidenstufe gefunden werden, werden bis zur untersten Pyramidenstufe verfeinert. Aus Leistungsgründen wird bei der Verfeinerung jedoch nicht mehr überprüft, ob die Matches immer noch innerhalb des angegebenen Bereichs liegen.

MinScore: Der Parameter MinScore legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von MinScore gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, dass das Modell niemals verdeckt wird, kann MinScore auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden. Werden die Matches nicht bis zur untersten Pyramidenebene verfolgt (siehe unten) kann es in manchen Fällen vorkommen, dass Instanzen gefunden werden, deren Score geringfügig unter dem Wert von MinScore liegen.

Falls ein einzelner Wert in MinScore übergeben wird, wird dieser Wert auf alle Instanzen angewandt, die mit allen Modellen gefunden werden. Falls MinScore mehrere Einträge enthält, werden die Werte einzeln für die entsprechenden Modelle verwendet.

Falls die Formmodelle mit set_shape_model_clutter um Störparameter erweitert wurden und dadurch *'use_clutter'* aktiviert wurde, wird in MinScore zu jedem der vorhandenen Werte für den minimalen Score ein weiterer Wert erwartet, welcher den maximalen Störwert eines potentiellen Matches angibt, um als Instanz des Modells im Bild betrachtet werden zu können. Die Laufzeit mit Nutzung von Störparametern ist mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter NumMatches. Die verschiedenen Formmodelle müssen den gleichen Wert für *'use_clutter'* haben.

Wird der maximale Störwert separat für jedes Modell spezifiziert, was immer dann nötig ist, wenn auch der minimale Score für jedes Modell gesetzt wird, muss in MinScore die doppelte Anzahl an Einträgen enthalten sein wie in ModelIDs. Dabei müssen der minimale Score und der maximale Störwert verschränkt in MinScore angegeben werden. Werden in ModelIDs beispielsweise zwei Modelle angegeben, wobei für Ersteres ein minimaler Score von 0.9 und für das zweite 0.8 festgelegt wird und der maximale Störwert des ersten Modells 0.1, für das zweite Modell 0.2 betragen soll, muss MinScore $= [0.9, 0.1, 0.8, 0.2]$ angegeben werden.

NumMatches: Mit NumMatches kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als NumMatches Instanzen eine Bewertung größer als MinScore erreichen, werden nur die besten NumMatches Instanzen zurückgeliefert. Falls weniger als NumMatches Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter MinScore hat

Vorrang vor NumMatches. Sollen alle Modellinstanzen, deren Bewertung MinScore übersteigt, im Bild gefunden werden, muss NumMatches auf *0* gesetzt werden.

Beim Tracken der Matches durch die Bildpyramide werden auf jeder Ebene Matches verworfen, die wenig vielversprechend sind. Dadurch ist es möglich, dass Matches verworfen werden, die auf der niedrigsten Pyramidenebene einen höheren Score gehabt hätten. Beispielsweise kann das dazu führen, dass der gefundene Match für NumMatches gleich *1* nicht mit dem Match mit dem höchsten Score übereinstimmt, der von NumMatches gleich *0* oder > *1* zurückgegeben wird.

Falls erwartet wird, dass mehrere Objekte mit einem ähnlichen Score gefunden werden können, aber nur das Modell mit dem höchsten Score zurückgegeben werden soll, kann es vorteilhaft sein, NumMatches zu erhöhen, und dann den Match mit dem höchsten Score auszuwählen.

Für den Fall, dass das Formmodell mit set_shape_model_clutter um Störparameter erweitert wurde, berücksichtigt NumMatches auch den zweiten Wert, der in MinScore übergeben wird. Gefundene Instanzen weisen in diesem Fall einen Score auf, der größer als der erste Eintrag ist, sowie einen Störwert der niedriger als der zweite Eintrag ist. Ist die Anzahl gefundener Instanzen größer als NumMatches, werden die besten NumMatches Instanzen in Bezug auf ihren Störwert zurückgegeben. MinScore hat weiterhin Vorrang vor NumMatches und NumMatches muss weiterhin auf *0* gesetzt werden um alle Modellinstanzen zu erhalten, die die Bedingungen von MinScore erfüllen. Es ist zu beachten, dass beim Tracken der Matches durch die Bildpyramide keine Matches verworfen werden, wenn Störparameter verwendet werden. Die Laufzeit mit Nutzung von Störparametern ist daher mindestens so hoch wie die Laufzeit ohne Störparameter mit auf *0* gesetztem Parameter NumMatches.

Falls NumMatches einen Wert enthält, liefert find_shape_models die NumMatches besten Instanzen des Modells unabhängig von der Art des Modells zurück. Falls z.B. in ModelIDs zwei Modelle übergeben werden und NumMatches $= 2$ gewählt wird, kann es sein, dass zwei Instanzen des ersten Modells und keine des zweiten Modells, eine Instanz des ersten Modells und eine des zweiten Modells oder keine Instanz des ersten Modells und zwei des zweiten Modells zurückgeliefert werden. Falls hingegen NumMatches mehrere Werte enthält, werden so viele Instanzen des jeweiligen Modells in ModelIDs zurückgeliefert, wie durch das entsprechende Element von NumMatches angegeben. Falls z.B. NumMatches $= [1, 1]$ gewählt wird, wird eine Instanz des ersten Modells und eine des zweiten Modells zurückgeliefert.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter MaxOverlap kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als MaxOverlap überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe smallest_rectangle2). Bei MaxOverlap $= 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei MaxOverlap $= 1$ werden alle gefundenen Instanzen zurückgeliefert.

Falls MaxOverlap einen Wert enthält, wird die Überlappung für alle gefundenen Instanzen des Modells unabhängig von der Art des Modells berechnet. D.h., dass sich zu viel überlappende Instanzen verschiedener und gleicher Modelle eliminiert werden. Falls hingegen mehrere Werte in MaxOverlap übergeben werden, wird die Überlappung nur innerhalb der gefundenen Instanzen des jeweiligen Typs der Modelle in ModelIDs berechnet. D.h., es werden nur sich zu viel überlappende Instanzen gleicher Modelle eliminiert. Modelle verschiedenen Typs können sich in diesem Modus vollständig überlappen.

SubPixel: Der Parameter SubPixel gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls SubPixel auf *'none'* (oder *'false'* für Rückwärtskompatibilität) gesetzt wird, wird die Lage des Modells nur pixelgenau bzw. mit der bei create_shape_model angegebenen Winkelauflösung bestimmt. Falls SubPixel auf *'interpolation'* (oder *'true'*) gesetzt wird, werden sowohl die Position als auch die Rotation subpixelgenau bestimmt. Dabei wird die Lage des Modells anhand der Score-Funktion interpoliert. Dieser Modus kostet fast keine Rechenzeit und liefert in den meisten Anwendungen eine ausreichende Genauigkeit. In manchen Anwendungen ist eine möglichst hohe Genauigkeit wichtig. In diesen Fällen kann die Lage des Modells durch Ausgleichsrechnung, d.h., durch Minimierung der Abstände der Modellpunkte und der zugehörigen Bildpunkte (Least-Squares Adjustment), bestimmt werden. Im Gegensatz zu *'interpolation'* kostet dieser Modus zusätzliche Rechenzeit. Mit den verschiedenen Modi (*'least_squares'*, *'least_squares_high'* und *'least_squares_very_high'*) kann dabei die Genauigkeit festgelegt werden, mit der der minimale Abstand gesucht wird. Je höher die Genauigkeit gewählt wird, desto länger dauert allerdings auch die Subpixel-Extraktion. Im Normalfall sollte für SubPixel *'interpolation'* gewählt werden. Falls die Ausgleichsrechnung gewünscht wird, sollte *'least_squares'* gewählt werden, da sich so der optimale Kompromiss zwischen Laufzeit und Genauigkeit ergibt.

Objekte, die im Vergleich zum Modell leicht deformiert im Bild erscheinen, können in manchen Fällen entweder nicht oder nur mit einer geringen Genauigkeit gefunden werden. Für solche Objekte besteht die Möglichkeit, im Parameter SubPixel zusätzlich die maximal zulässige Objektdeformation in Pixeln anzugeben. Dies kann mit Hilfe des optionalen Parameterwertes *'max_deformation '* und einer nachfolgenden Ganzzahl zwischen *0* und *32* (im selben String), die die maximale Deformation beschreibt, erfolgen. Kann die Form des Objektes z.B. bis zu 2 Pixel von der im Modell gespeicherten Form abweichen, muss im Parameter SubPixel der Wert *'max_deformation 2'* zusätzlich zum oben beschriebenen Modus für die subpixelgenaue Extraktion übergeben werden, also z.B. *['least_squares', 'max_deformation 2']*. Mit *'max_deformation 0'* wird das Objekt ohne Deformationen gesucht, also so als ob man kein *'max_deformation '* übergeben hätte. Zu beachten ist, dass größere Werte für die maximale Deformation oft zu erhöhten Laufzeiten führen. Außerdem besteht bei einem zu großen Wert die Gefahr, dass falsche Kandidaten gefunden werden. Beide Probleme können hauptsächlich bei kleinen Objekten oder Objekten mit feinen Strukturen auftreten. Der Grund hierfür ist, dass solche Objekte durch große erlaubte Deformationen ihre für die Suche wichtige charakteristische Form verlieren. Zu beachten ist auch, dass die Genauigkeit von teilweise verdeckten Objekten für höhere Deformationen abnehmen kann wenn Störobjekte in der Nähe des Objektes vorhanden sind. Die maximale Deformation sollte demnach immer nur so groß wie nötig und so gering wie möglich gewählt werden. Näherungsweise rotationssymmetrische Objekt könnten nicht gefunden werden, falls *'max_deformation'* und AngleExtent beide auf einen Wert größer 0 gesetzt sind. In dem Fall kann es zu Mehrdeutigkeiten kommen, die nicht aufgelöst werden können, sodass der Match als falsch verworfen wird. Wenn das passiert, versuchen Sie, entweder *'max_deformation'* oder AngleExtent auf 0 zu setzen oder das Modell so zu verändern, dass weniger Symmetrien auftreten. Die Score-Berechnung bei Deformationswerten größer *0* hängt von der Art der Subpixel-Extraktion ab. In den meisten Fällen wird sich der Score eines Matches verändern, wenn man *'least_squares'*, *'least_squares_high'* oder *'least_squares_very_high'* (siehe oben) wählt (im Vergleich zu *'none'* oder *'interpolation'*). Außerdem wird sich der Score in der Regel erhöhen, wenn man bei einer der Least-Squares-Adjustment-Methoden die maximale Deformation erhöht, da dann zu den Modellpunkten mehr zugehörige Bildpunkte gefunden werden können. Für einen aussagekräftigen Score und zum Vermeiden falscher Matches wird empfohlen, das Erlauben einer Deformation mit einer Subpixel-Extraktion mit einer der Least-Squares-Adjustment-Methoden zu kombinieren.

Falls die Subpixel-Extraktion und/oder die maximale Objektdeformation für jedes Modell einzeln spezifiziert werden sollen, muss in SubPixel die Subpixel-Extraktion genauso oft angegeben werden, wie die Anzahl von Elementen in ModelIDs. Nach jedem übergebenen Wert für die Subpixel-Extraktion kann optional ein zweiter Wert übergeben werden, der die maximale Objektdeformation für das jeweilige Modell beschreibt. Wird nach dem Wert für die Subpixel-Extraktion kein Wert für die Objektdeformation übergeben, wird das entsprechende Modell ohne Deformation gesucht. Falls z.B. zwei Modelle in ModelIDs angegeben sind und für das erste Modell die Subpixel-Extraktion *'interpolation'* ohne zulässige Objektdeformation gewählt werden soll und für das zweite Modell die Subpixel-Extraktion *'least_squares'* gewünscht ist sowie eine maximale Objektdeformation von *3* Pixeln, dann kann für den Parameter SubPixel das Tupel *['interpolation', 'least_squares', 'max_deformation 3']* übergeben werden. Alternativ kann auch das äquivalente Tupel *['interpolation', 'max_deformation 0', 'least_squares', 'max_deformation 3']* übergeben werden.

NumLevels: Mit NumLevels wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit create_shape_model angegebenen Bereich beschnitten. Falls NumLevels als *0* angegeben wird, wird die mit create_shape_model angegebene Anzahl verwendet.

In manchen Fällen kann es sein, dass die Anzahl der Pyramidenebenen, die beispielsweise automatisch mit create_shape_model ermittelt wurde, zu hoch ist. Dann werden eventuell Instanzen, die eine sehr hohe finale Bewertung gehabt hätten, bereits auf der höchsten Pyramidenebene ausgeschlossen und damit nicht gefunden. Anstatt nun MinScore auf einen sehr geringen Wert zu setzen, um alle Matches zu finden, kann der Wert von NumLevels mit get_shape_model_params ermittelt werden und dann ein etwas geringerer Wert in find_shape_models genutzt werden. Diese Herangehensweise führt oft zu besseren Ergebnissen hinsichtlich Geschwindigkeit und Robustheit.

Optional kann NumLevels einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, Laufzeit einzusparen. Allerdings ist in diesem Modus im Allgemeinen die Genauigkeit der gefundenen Lageparameter geringer als im Normalfall, in dem die Matches bis auf die unterste Pyramidenebene verfolgt werden. Falls eine hohe Genauigkeit erzielt werden soll, empfiehlt es sich also, mindestens *'least_squares'* für SubPixel zu verwenden. Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells

gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden.

Falls die unterste Pyramidenebene für jedes Modell einzeln spezifiziert werden soll, müssen in NumLevels zweimal die Anzahl von Elementen in ModelIDs angegeben werden. Dabei sind die Anzahl der Pyramidenebenen und der untersten Pyramidenebenen verschränkt anzugeben. Falls z.B. zwei Modelle in ModelIDs angegeben sind, die oberste Pyramidenebene für das erste Modell 5 und für das zweite Modell 4 sein soll und die unterste Pyramidenebene für das erste Modell 2 und für das zweite 1 sein soll, so ist NumLevels $= [5, 2, 4, 1]$ zu wählen. Falls genau zwei in ModelIDs angegeben sind, ergibt sich als Spezialfall, dass, falls die unterste Pyramidenebene spezifiziert werden soll, die oberste und unterste Pyramidenebene für beide Modelle explizit spezifiziert werden muss, selbst wenn sie gleich sind. Ein Tupel der Länge zwei in NumLevels wird in diesem Fall nämlich als Spezifikation der obersten Pyramidenebene für die zwei Modelle interpretiert.

In mangelhaften Eingabebildern, d.h., in Bildern die z.B. unscharf, deformiert oder verrauscht sind, kann auf der untersten Pyramidenebene oft keine Instanz des Formmodells gefunden werden, da aufgrund der mangelhaften Bildqualität nicht genügend Kanteninformation vorhanden ist, oder die Kanten zu stark deformiert sind. Auf höheren Pyramidenebenen dagegen kann die Kanteninformation noch ausreichend sein um Instanzen des Modells zu finden, wobei auch hierbei die oben beschriebenen Einschränkungen bezüglich Genauigkeit und Robustheit gelten. Die Wahl der passenden Pyramidenebene, d.h., der untersten Pyramidenebene auf der noch eine Instanz des Modells gefunden wird, hängt sowohl vom Modell als auch vom Eingabebild ab. Diese Pyramidenebene kann also von Bild zu Bild variieren. Um das Matching auf mangelhaften Bildern zu erleichtern, kann die unterste Pyramidenebene, auf der noch eine Instanz des Modells gefunden wird, während des Matchings automatisch bestimmt werden. Um diesen als 'increased tolerance mode' bezeichneten Mechanismus zu aktivieren, muss die unterste Pyramidenebene in NumLevels negativ angegeben werden. Wird z.B. NumLevels $= [5, 2, 4, -1]$ gesetzt, so ist die unterste Pyramidenebene für das erste Modell 2. Kann auf der Pyramidenebene 2 keine Instanz des ersten Formmodells gefunden werden, so wird für dieses Modell kein Ergebnis zurückgegeben. Für das zweite Modell ist in diesem Beispiel die unterste Pyramidenebene auf *-1* gesetzt. Daher wird versucht, eine Instanz des Formmodells auf der Pyramidenebene 1 zu finden. Kann auf dieser Pyramidenebene keine Instanz gefunden werden, so wird die unterste Pyramidenebene ermittelt, auf der noch Instanzen des Modells gefunden wurden. Die Instanzen dieser Pyramidenebene werden dann als Ergebnis zurückgegeben.

Wenn ein Modell mit adapt_shape_model_high_noise angepasst wurde wird die geschätzte untere Pyramidenstufe per Standardeinstellung verwendet. Der Benutzer kann den geschätzten Wert dennoch überschreiben, indem er jeweils zwei Werte an NumLevels übergibt und damit explizit die untere Pyramidenstufe setzt.

Greediness: Der Parameter Greediness bestimmt, wie „gierig" die Suche durchgeführt werden soll. Für Greediness $= 0$ wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden, immer findet, wenn die anderen Parameter passend gesetzt sind. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für Greediness $= 1$ wird eine unsicherere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, dass das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für Greediness $= 1$ wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten Fällen wird das Formmodell für Greediness $= 0.9$ immer sicher gefunden.

Ausgabeparameter im Detail

Row, Column und Angle: Die Position und Rotation der gefundenen Instanzen der Modelle wird in Row, Column und Angle zurückgeliefert. Die Koordinaten Row und Column sind die Koordinaten des Ursprungs des Modells im Suchbild. Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_shape_model verwendet wurde. Ein anderer Ursprung kann mit set_shape_model_origin festgelegt werden.

Beachten Sie, dass die Koordinaten Row und Column nicht exakt mit dem Ursprung des Modells übereinstimmen und daher nicht direkt verwendet werden sollten. Die Werte sind dafür optimiert, die Transformationsmatrix zu erzeugen, mit der die Matching-Ergebnisse für verschiedene Aufgaben verwendet werden können, z.B. um ROIs für andere Bildverarbeitungsschritte einem Objekt nachzuführen. Das bei find_shape_model angegebene Beispiel zeigt, wie diese Matrix erzeugt wird und wie man damit das Modell an der gefundenen Position im Suchbild visualisiert.

Beachten Sie, dass die Visualisierung aber auch mit der Prozedur dev_display_shape_matching_results realisiert werden kann.

Score: In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Falls das Formmodell mit `set_shape_model_clutter` um Störparameter erweitert wurde, gibt `Score`, nach den oben genannten Werten, auch den Störwert jeder gefundenen Instanz zurück. Falls beispielsweise die Hälfte der Störregion Störkanten aufweist entspricht der Störwert 0.5. Werden z.B. zwei Instanzen gefunden, wobei für Erstere eine Bewertung von 0.9 und für die zweite 0.8 erreicht wird und der Störwert der ersten Instanz 0.2, der zweiten Instanz 0.1 ist, wird $\texttt{Score} = [0.9, 0.8, 0.2, 0.1]$ zurückgegeben. Es ist zu beachten, dass von sämtlichen Ergebnissen des formbasierten Matchings die Störwerte am stärksten von Beleuchtungsänderungen beeinflusst werden.

Model: Der gefundene Modelltyp wird in `Model` zurückgeliefert. Die Elemente von `Model` sind ein Index in das Tupel `ModelIDs`, d.h. sie können Werte von 0 bis $|\texttt{ModelIDs}| - 1$ enthalten. Ein Wert von 0 in einem Element von `Model` entspricht also einer Instanz des ersten in `ModelIDs` übergebenen Modells.

Setzen eines Timeout

Mittels des Operators `set_shape_model_param` können Sie einen *'timeout'* für `find_shape_models` angeben. Falls die durch `ModelIDs` referenzierten Modelle unterschiedliche Werte für *'timeout'* besitzen, wählt `find_shape_models` den niedrigsten aus. Wenn `find_shape_models` diesen *'timeout'* erreicht endet er ohne Ergebnis und gibt den Fehlercode 9400 (H_ERR_TIMEOUT) zurück.

MinContrast bei mehreren Modellen

Wenn über den Parameter `ModelIDs` mehrere Modelle übergeben werden, sollten diese mit dem gleichen Wert für `MinContrast` erzeugt worden sein. Falls sie mit unterschiedlichen Werten für `MinContrast` erzeugt worden sind, wird in `find_shape_models` der kleinste dieser Werte verwendet.

Darstellung der Resultate

Die Verwendung der Prozedur `dev_display_shape_matching_results` wird zur Darstellung der Resultate formbasierten Matchings stark empfohlen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Eingabebild, in dem die Modelle gefunden werden sollen.

▷ **`ModelIDs`** (input_control) . . . . . shape_model(-array) ⇝ *handle*
Handle der Modelle.

▷ **`AngleStart`** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Kleinste auftretende Rotation der Modelle.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **`AngleExtent`** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
Restriktion: AngleExtent >= 0

▷ **`MinScore`** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen der Modelle.
Defaultwert: 0.5
Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **`NumMatches`** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der zu findenden Instanzen der Modelle (oder 0 für alle Treffer).
Defaultwert: 1
Wertevorschläge: `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **MaxOverlap** (input_control) . . . . . real(-array) ⇝ *real*
Maximale Überlappung der zu findenden Instanzen der Modelle.
Defaultwert: 0.5
Wertevorschläge: MaxOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ MaxOverlap ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **SubPixel** (input_control) . . . . . string(-array) ⇝ *string*
Subpixelgenauigkeit falls ungleich *'none'*.
Defaultwert: 'least_squares'
Wertevorschläge: SubPixel ∈ {'none', 'interpolation', 'least_squares', 'least_squares_high', 'least_squares_very_high', 'max_deformation 1', 'max_deformation 2', 'max_deformation 3', 'max_deformation 4', 'max_deformation 5', 'max_deformation 6'}

▷ **NumLevels** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls |NumLevels| = 2).
Defaultwert: 0
Werteliste: NumLevels ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) . . . . . real(-array) ⇝ *real*
„Gierigkeit" der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen" werden).
Defaultwert: 0.9
Wertevorschläge: Greediness ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich: 0 ≤ Greediness ≤ 1
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinate der gefundenen Instanzen der Modelle.

▷ **Column** (output_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinate der gefundenen Instanzen der Modelle.

▷ **Angle** (output_control) . . . . . angle.rad-array ⇝ *real*
Rotationswinkel der gefundenen Instanzen der Modelle.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Bewertung der gefundenen Instanzen der Modelle.

▷ **Model** (output_control) . . . . . integer-array ⇝ *integer*
Index der gefundenen Instanzen der Modelle.

Beispiel

```
read_image (Image, 'pcb_focus/pcb_focus_telecentric_061')
gen_rectangle1 (ROI_0, 236, 241, 313, 321)
gen_circle (ROI_1, 281, 653, 41)
reduce_domain (Image, ROI_0, ImageReduced1)
reduce_domain (Image, ROI_1, ImageReduced2)

create_shape_model (ImageReduced1, 0, rad(-45), rad(180), 0, \
                    'none', 'use_polarity', 30, 10, ModelID1)
create_shape_model (ImageReduced2, 0, rad(-45), rad(180), 0, \
                    'none', 'use_polarity', 30, 10, ModelID2)
ModelIDs:=[ModelID1, ModelID2]
find_shape_models (Image, ModelIDs, rad(-45), rad(90), 0.7, [1,1], 0.5, \
                   'least_squares', 0, 1, Row, Column, Angle, Score, Model)
* Display results
dev_display_shape_matching_results (ModelIDs, 'red', Row, Column, Angle, \
                                    1, 1, Model)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert find_shape_models den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system

('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

add_channels, create_shape_model, read_shape_model, set_shape_model_origin, set_shape_model_param, set_shape_model_clutter

Nachfolger

clear_shape_model

Alternativen

find_scaled_shape_models, find_aniso_shape_models, find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_ncc_model, find_ncc_models

Siehe auch

set_system, get_system

Modul

Matching

get_generic_shape_model_param (: : ModelID, GenParamName : GenParamValue)

Auslesen der Parameter des Formmodells.

Der Operator get_generic_shape_model_param gibt die Parameterwerte GenParamValue für GenParamName des Formmodells ModelID zurück.

Für eine Beschreibung der setzbaren Werte GenParamValue, siehe set_generic_shape_model_param. Zusätzliche Parameter, deren Werte lediglich ausgelesen werden können, sind:

- *'needs_training'*:
  Gibt an, ob das Modell mit train_generic_shape_model trainiert werden muss bevor es von find_generic_shape_model verwendet werden kann (*'true'*) oder nicht (*'false'*).
- *'model_identifier'*:
  String, der als eindeutiger Identifikator des Formmodells dient.

Der Operator get_generic_shape_model_param gibt Werte so zurück wie sie vom Formmodell verwendet werden. Da bestimmte Parameter durch den Aufruf von train_generic_shape_model modifiziert werden können, kann sich deren Werte von den gesetzten Werten unterscheiden. Dies gilt insbesondere für Parameter, deren Werte automatisch bestimmt werden, beispielsweise Werte, die auf *'auto'* gesetzt wurden. Der (unmodifizierte) gesetzte Wert von GenParamName kann durch das Hinzufügen des Suffix *'_param'* an den Parameternamen abgefragt werden.

Beispiel: Nach dem Aufruf von train_generic_shape_model gibt *'num_levels'* den geschätzten und *'num_levels_param'* den gesetzten Wert (z.B. der Standardwert oder der vom Nutzer gesetzte Wert).

Parameter

▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
Handle des Formmodells.

▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
Parameternamen.
Defaultwert: 'min_score'
Werteliste: GenParamName ∈ {'angle_end', 'angle_extent', 'angle_start', 'angle_step', 'border_shape_models', 'contrast_high', 'contrast_low', 'greediness', 'iso_scale_max', 'iso_scale_min',

'iso_scale_step', 'max_clutter', 'max_deformation', 'max_overlap', 'metric', 'min_contrast', 'min_score', 'min_size', 'model_cache', 'model_identifier', 'needs_training', 'num_levels', 'num_matches', 'optimization', 'origin_column', 'origin_row', 'prepare_contours_for_visualization', 'prepare_clutter_region_for_visualization', 'pyramid_level_highest', 'pyramid_level_lowest', 'pyramid_level_robust_tracking', 'restrict_iso_scale_max', 'restrict_iso_scale_min', 'restrict_scale_column_max', 'restrict_scale_column_min', 'restrict_scale_row_max', 'restrict_scale_row_min', 'scale_column_max', 'scale_column_min', 'scale_column_step', 'scale_row_max', 'scale_row_min', 'scale_row_step', 'strict_boundaries', 'subpixel', 'timeout', 'use_clutter'}

▷ **GenParamValue** (output_control) ..... attribute.value-array ⇝ *real* / integer / string
Parameterwerte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_generic_shape_model_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_generic_shape_model`, `set_generic_shape_model_param`, `train_generic_shape_model`

Modul

Matching

get_generic_shape_model_result (: : MatchResultID, MatchSelector, GenParamName : GenParamValue)

Auslesen alphanumerischer Werte des Matching-Ergebnisses.

`get_generic_shape_model_result` gibt alphanumerische Ergebnisse des formbasierten Matchings in `GenParamValue` zurück. Die in `MatchResultID` enthaltenen Matches sind absteigend nach ihrem Score sortiert.

Der Parameter `MatchSelector` dient zur Auswahl von welchen Matches Werte ausgelesen werden sollen. Die Reihenfolge der Matches wird dabei ihren Scores entsprechend beibehalten, daher ist die Auswahl unabhängig von der Reihenfolge der Auswahlkriterien die in `MatchSelector` übergeben werden. Zudem wird ein Matching-Ergebnis nur einmal zurückgegeben, auch wenn es mehr als ein Auswahlkriterium erfüllt.

Für `MatchSelector` können folgende Auswahlkriterien verschiedener Arten übergeben und kombiniert werden:

- *'all'*:
  Alle Matching-Ergebnisse werden ausgewählt.
- *'best'*:
  Das Matching-Ergebnis mit dem höchsten Score wird ausgewählt.
- Ergebnisindex:
  Wird eine (ganze) Zahl für `MatchSelector` gesetzt, wird damit der Index des Matches in `MatchResultID` festgelegt.
  Wertebereich: 0 bis n-1 (wobei n für die Anzahl Ergebnisse steht).
  Ist der angegebene Index außerhalb des zulässigen Wertebereichs wird eine Fehlermeldung zurückgegeben.

- Identifikator des Formmodells:
  Alle Matches die mit einem bestimmten Formmodell gefunden wurden werden ausgewählt. Dazu kann der Modell-Identifikator *'model_identifier'* als String an `MatchSelector` übergeben werden.
- Handle des Formmodells:
  Alle Matches die mit einem bestimmten Formmodell gefunden wurden werden ausgewählt. Dazu kann das Handle des Formmodells an `MatchSelector` übergeben werden.

Falls `MatchSelector` kein gültiges Auswahlkriterium ist, wird eine Fehlermeldung zurückgegeben.

`GenParamName` gibt an welcher alphanumerische Ergebniswert zurückgegeben werden soll. Folgende Werte sind setzbar:

- *'num_match_result'*:
  Anzahl Matches. Für mehrere Auswahlkriterien in `MatchSelector` wird die kumulierte Anzahl zurückgegeben.
- *'model_identifier'*:
  String der das Modell identifiziert, mit welchem der Match gefunden wurde.
- *'score'*:
  Score des gefundenen Matches. Der Score ist eine Zahl zwischen 0 und 1 und gibt näherungsweise an wie gut das Modell im Bild gefundenen wurde. Für weitere Informationen zur Bedeutung des Scores, siehe „`Solution Guide II-B - Matching`", Kapitel 'General Topics', Abschnitt 'About the Score'.
- *'angle'*:
- *'angle'*:
  Rotationswinkel des gefundenen Matches.
  Wertebereich: -3.14 $\leq$ *'angle'* $\leq$ *3.14* (=π).
- *'row'*:
  Zeilenkoordinate des gefundenen Matches im Suchbild. Die Koordinate bezieht sich auf den Ursprung des Formmodells.
- *'column'*:
  Spaltenkoordinate des gefundenen Matches im Suchbild. Die Koordinate bezieht sich auf den Ursprung des Formmodells.
- *'scale_row'*:
  Skalierung des gefundenen Matches in Zeilenrichtung.
- *'scale_column'*:
  Skalierung des gefundenen Matches in Spaltenrichtung.
- *'hom_mat_2d'*:
  Homographische Transformationsmatrix zum Transformieren der Formmodellkonturen von ihrem Ursprung an den Ort im Suchbild an dem der Match gefunden wurde. Dabei werden Position, Rotation und Skalierung berücksichtigt. Für weitere Informationen zur Transformation mit homographischen Matrizen siehe `affine_trans_point_2d`.
- *'clutter_score'*:
  Störwert des gefundenen Matches. Der Wert ist eine Zahl zwischen 0 und 1 und gibt näherungsweise an wie stark Störkanten in der Störregion vorhanden sind.
- *'clutter_hom_mat_2d'*:
  Homographische Transformationsmatrix zum Transformieren der Störregion des Formmodells von ihrem Ursprung an den Ort im Suchbild an dem der Match gefunden wurde. Dabei werden Position, Rotation und Skalierung berücksichtigt. Für weitere Informationen zur Transformation mit homographischen Matrizen siehe `affine_trans_point_2d`.

Es ist zu beachten, dass für Werte, die sich auf die Lage der Matches beziehen Koordinaten als Ecken-zentrierte Koordinaten gegeben werden. Für weitere Informationen über Ecken-zentrierte Koordinaten siehe das Kapitel Transformationen / 2D-Transformationen.

Parameter

- ▷ **MatchResultID** (input_control) . . . . . generic_shape_model_result ⇝ *handle*
  Handle der Matches.
- ▷ **MatchSelector** (input_control) . . . . . string-array ⇝ *string* / integer
  Selektor für die auszulesenden Matching-Ergebnisse.
  Defaultwert: 'all'
  Werteliste: `MatchSelector` ∈ {'all', 'best'}
- ▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
  Name des auszulesenden Ergebnisparameters.
  Defaultwert: 'score'
  Werteliste: `GenParamName` ∈ {'num_match_result', 'model_identifier', 'score', 'angle', 'row', 'column', 'scale_row', 'scale_column', 'hom_mat_2d', 'clutter_score', 'clutter_hom_mat_2d'}
- ▷ **GenParamValue** (output_control) . . . . . attribute.value-array ⇝ *real* / integer / string
  Wert des auszulesenden Ergebnisparameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_generic_shape_model_result` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_generic_shape_model`

Modul

Matching

get_generic_shape_model_result_object (: Objects : MatchResultID, MatchSelector, GenParamName :)

Auslesen von Objekten des Matching-Ergebnisses.

`get_generic_shape_model_result_object` gibt in `Objects` die ikonischen Ergebnisse eines formbasierten Matchings zurück. Die Matching-Ergebnisse sind in `MatchResultID` enthalten.

Der Parameter `MatchSelector` dient zur Auswahl von welchen Matches Werte ausgelesen werden sollen. Für weitere Informationen und eine Liste von unterstützten Werten für `MatchSelector` siehe `get_generic_shape_model_result`.

`GenParamName` gibt an welches Objekt des Matching-Ergebnisses zurückgegeben werden soll. Folgende Werte sind setzbar:

'contours': Die Modellkonturen, dem Matching-Ergebnis entsprechend transformiert.

'clutter_region': Die Region in der keine Störungen auftreten sollen.

Parameter

- ▷ **Objects** (output_object) . . . . . (multichannel-)object(-array) ⇝ *object*
  Auszulesende Objekte des Matching-Ergebnisses.
- ▷ **MatchResultID** (input_control) . . . . . generic_shape_model_result ⇝ *handle*
  Handle der Matches.
- ▷ **MatchSelector** (input_control) . . . . . string-array ⇝ *string* / integer
  Selektor für die auszulesenden Matching-Ergebnisse.
  Defaultwert: 'all'
  Werteliste: `MatchSelector` ∈ {'all', 'best'}

▷ **GenParamName** (input_control) . . . . . string ⇝ *string*
Name des auszulesenden Ergebnisparameters.
Defaultwert: 'contours'
Werteliste: GenParamName ∈ {'contours', 'clutter_region'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_generic_shape_model_result_object den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

find_generic_shape_model

Modul

Matching

get_shape_model_clutter (: ClutterRegion : ModelID,
GenParamName : GenParamValue, HomMat2D, ClutterContrast)

Auslesen der Störparameter eines Formmodells.

Der Operator get_shape_model_clutter kann dazu genutzt werden, die Störparameter des Formmodells ModelID zu untersuchen, die zuvor mit set_shape_model_clutter gesetzt wurden. Ebenfalls kann der Wert von *'use_clutter'*, der in set_shape_model_param gesetzt werden kann, abgefragt werden, indem GenParamName auf *'use_clutter'* gesetzt wird. Für eine Beschreibung der Parameter siehe set_shape_model_clutter sowie set_shape_model_param.

Parameter

▷ **ClutterRegion** (output_object) . . . . . region ⇝ *object*
Region, in der keine Störungen auftreten sollen.

▷ **ModelID** (input_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Parameternamen.
Defaultwert: 'use_clutter'
Werteliste: GenParamName ∈ {'use_clutter', 'clutter_border_mode'}

▷ **GenParamValue** (output_control) . . . . . attribute.name(-array) ⇝ *string* / real / integer
Parameterwerte.

▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
Transformationsmatrix.

▷ **ClutterContrast** (output_control) . . . . . number ⇝ *integer*
Minimaler Kontrast einer Störung.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_shape_model_clutter den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_shape_model_clutter, read_shape_model, deserialize_shape_model

Nachfolger

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models

Siehe auch

create_shape_model, create_scaled_shape_model, create_aniso_shape_model, set_shape_model_param

Modul

Matching

get_shape_model_contours (: ModelContours : ModelID, Level :)

Auslesen der Konturrepräsentation eines Formmodells.

Der Operator get_shape_model_contours gibt in ModelContours eine Repräsentation des Formmodells ModelID als XLD-Konturen zurück. Der Parameter Level gibt an, für welche Pyramidenstufe des Modells die Konturrepräsentation zurückgegeben werden soll. Die Konturen können z.B. zur Visualisierung der gefundenen Instanzen des Modells verwendet werden. Dabei ist darauf zu achten, dass die Lage von ModelContours so normalisiert worden ist, dass der Referenzpunkt des Modells (siehe set_shape_model_origin) an der Pixelposition $(0, 0)$ liegt. D.h., die Konturen müssen lediglich an die gefundene Stelle verschoben werden (und eventuell um diesen Punkt rotiert und skaliert werden).

Parameter

- ▷ **ModelContours** (output_object) ..... xld_cont-array ⇝ *object*
  Konturrepräsentation des Formmodells.
- ▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
  Handle des Modells.
- ▷ **Level** (input_control) ..... integer ⇝ *integer*
  Pyramidenebene, für die die Konturrepräsentation zurückgegeben werden soll.
  Defaultwert: 1
  Wertevorschläge: Level ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Restriktion: Level >= 1

Ergebnis

Ist der Handle des Modells gültig, dann liefert get_shape_model_contours den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_shape_model, create_scaled_shape_model, create_aniso_shape_model, read_shape_model

Siehe auch

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models

Modul

Matching

get_shape_model_origin (: : ModelID : Row, Column)

Auslesen des Ursprungs (Referenzpunktes) eines Formmodells.

Der Operator get_shape_model_origin gibt den Ursprung (Referenzpunkt) des Formmodells ModelID zurück. Der Ursprung wird relativ zum Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit create_shape_model, create_scaled_shape_model oder create_aniso_shape_model verwendet wurde, angegeben. Ein Ursprung von (0,0) heißt also, dass der Schwerpunkt der Region des Modellbildes verwendet wird. Ein Ursprung von (-20,-40) heißt, dass der Ursprung links oberhalb des Schwerpunktes liegt.

Parameter

▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
Handle des Modells.

▷ **Row** (output_control) ..... point.y ⇝ *real*
Zeilenkoordinate des Ursprungs des Formmodells.

▷ **Column** (output_control) ..... point.x ⇝ *real*
Spaltenkoordinate des Ursprungs des Formmodells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert get_shape_model_origin den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_shape_model, create_scaled_shape_model, create_aniso_shape_model, read_shape_model, set_shape_model_origin

Nachfolger

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models

Siehe auch

area_center

Modul

Matching

get_shape_model_params (: : ModelID : NumLevels, AngleStart, AngleExtent, AngleStep, ScaleMin, ScaleMax, ScaleStep, Metric, MinContrast)

Auslesen der Parameter eines Formmodells.

Der Operator get_shape_model_params gibt die zur Erzeugung des Formmodells ModelID mit create_shape_model, create_scaled_shape_model oder create_aniso_shape_model verwendeten Parameter zurück. Dies ist insbesondere hilfreich, um die Parameter NumLevels, AngleStep, ScaleStep oder MinContrast zu überprüfen, falls diese durch create_shape_model, create_scaled_shape_model oder create_aniso_shape_model automatisch bestimmt wurden.

Wenn ModelID mit adapt_shape_model_high_noise angepasst wurde, wird NumLevels zwei Werte enthalten: die oberste Pyramidenstufe und die geschätzte unterste Pyramidenstufe.

Falls das Formmodell mit create_shape_model oder create_scaled_shape_model erzeugt worden ist, wird in ScaleMin, ScaleMax und ScaleStep jeweils ein Wert für den jeweiligen isotropen Skalierungsparameter zurückgegeben. Falls das Formmodell mit create_aniso_shape_model erzeugt worden ist, werden in den obigen drei Parametern jeweils zwei Werte zurückgegeben. Dabei bezieht sich der jeweils erste Parameter auf die Skalierung in Zeilenrichtung, der jeweils zweite Parameter auf die Skalierung in Spaltenrichtung.

Zu beachten ist, dass die Parameter Optimization und Contrast, die während der Modellgenerierung ebenfalls automatisch bestimmt werden können, nicht mit get_shape_model_params abgefragt werden können.

In diesem Fall sollte determine_shape_model_params statt get_shape_model_params verwendet werden.

Parameter

- ▷ **ModelID** (input_control) . . . . . shape_model ⇝ *handle*
  Handle des Modells.
- ▷ **NumLevels** (output_control) . . . . . integer(-array) ⇝ *integer*
  Anzahl von Pyramidenebenen.
- ▷ **AngleStart** (output_control) . . . . . angle.rad ⇝ *real*
  Kleinste auftretende Rotation des Musters.
- ▷ **AngleExtent** (output_control) . . . . . angle.rad ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Zusicherung: AngleExtent >= 0
- ▷ **AngleStep** (output_control) . . . . . angle.rad ⇝ *real*
  Schrittweite der Winkel (Auflösung).
  Zusicherung: AngleStep >= 0 && AngleStep <= pi / 16
- ▷ **ScaleMin** (output_control) . . . . . number(-array) ⇝ *real*
  Kleinste auftretende Skalierung des Musters.
  Zusicherung: ScaleMin > 0
- ▷ **ScaleMax** (output_control) . . . . . number(-array) ⇝ *real*
  Größte auftretende Skalierung des Musters.
  Zusicherung: ScaleMax >= ScaleMin
- ▷ **ScaleStep** (output_control) . . . . . number(-array) ⇝ *real*
  Schrittweite der Skalierungen (Auflösung).
  Zusicherung: ScaleStep >= 0
- ▷ **Metric** (output_control) . . . . . string ⇝ *string*
  Art der zum Matchen verwendeten Metrik.
- ▷ **MinContrast** (output_control) . . . . . number ⇝ *integer*
  Minimaler Kontrast des Objektes in den Suchbildern.

Ergebnis

Ist der Handle des Modells gültig, dann liefert get_shape_model_params den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_shape_model, create_scaled_shape_model, create_aniso_shape_model, read_shape_model

Siehe auch

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models

Modul

Matching

inspect_shape_model (Image : ModelImages, ModelRegions : NumLevels, Contrast :)

Repräsentation eines Formmodells.

inspect_shape_model erzeugt eine Repräsentation eines Formmodells. Der Operator ist insbesondere dazu nützlich, um die Parameter NumLevels und Contrast, die bei create_shape_model, create_scaled_shape_model oder create_aniso_shape_model verwendet werden, einfach und

schnell zu bestimmen. Die Repräsentation des Modells wird auf mehreren Pyramidenebenen generiert, wobei die Anzahl der Pyramidenebenen durch NumLevels bestimmt wird. Im Gegensatz zu create_shape_model, create_scaled_shape_model und create_aniso_shape_model wird das Modell nur in der im Eingabebild Image vorliegenden Rotation und Skalierung, also 0° und 1, erzeugt. Als Ausgabe erzeugt inspect_shape_model ein Bildobjekt ModelImages mit den Bildern der einzelnen Pyramidenstufen, sowie eine Region in ModelRegions für jede Pyramidenstufe, die das Modell auf dieser Pyramidenstufe repräsentiert. Auf die einzelnen Objekte der jeweiligen Pyramidenstufe kann mit select_obj zugegriffen werden. Falls das Eingabebild Image einen Kanal besitzt, wird zur Berechnung der Repräsentation des Modells das Verfahren verwendet, das auch bei create_shape_model, create_scaled_shape_model oder create_aniso_shape_model für die Metriken *'use_polarity'*, *'ignore_global_polarity'* und *'ignore_local_polarity'* verwendet wird. Falls das Eingabebild mehr als einen Kanal besitzt, wird das Verfahren verwendet, das für die Metrik *'ignore_color_polarity'* verwendet wird. Wie bei create_shape_model, create_scaled_shape_model oder create_aniso_shape_model beschrieben, sollten die Anzahl der Pyramidenebenen möglichst groß gewählt werden, wobei darauf zu achten ist, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt. Der Parameter Contrast sollte so gewählt werden, dass nur die signifikanten Merkmale des Musters für das Modell verwendet werden. Contrast kann auch ein Tupel mit zwei Werten enthalten. In diesem Fall wird bei der Segmentierung des Modells ein Verfahren, das ähnlich zum Hysterese-Schwellenwert-Verfahren in edges_image ist, eingesetzt. Dabei ist der erste Wert des Tupels der untere Schwellenwert und der zweite Wert der obere Schwellenwert. Für weitere Informationen über das Hysterese-Schwellenwert-Verfahren siehe hysteresis_threshold. Optional kann in Contrast auch ein dritter Wert als letztes Element des Tupels übergeben werden. Dieser Wert dient zur Selektion von signifikanten Komponenten des Modells basierend auf der Größe der Komponenten, d.h. Komponenten, deren Anzahl von Punkten kleiner ist als diese Mindestgröße, werden unterdrückt. Da sich die Mindestgröße auf die Größe der Komponenten bezieht kann es auch vorkommen, dass die daraus abgeleiteten Modellkonturen trotzdem kleiner als die angegebene Mindestgröße sind. Dieser Schwellenwert wird pro Pyramidenstufe halbiert. Falls eine Unterdrückung von kleinen Modellkomponenten gewünscht ist, aber keine Hysterese-Schwellenwert-Operation, müssen dennoch drei Werte in Contrast übergeben werden, von denen die ersten zwei auf denselben Wert gesetzt werden. In der normalen Verwendung wird inspect_shape_model mehrmals interaktiv mit unterschiedlichen Werten für NumLevels und Contrast aufgerufen, bis ein zufriedenstellendes Modell entsteht. Hierauf werden create_shape_model, create_scaled_shape_model oder create_aniso_shape_model mit den so bestimmten Parametern aufgerufen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **ModelImages** (output_object) ..... (multichannel-)image-array ⇝ *object :* byte / uint2
  Bildpyramide des Eingabebildes.
- ▷ **ModelRegions** (output_object) ..... region-array ⇝ *object*
  Model-Regionen-Pyramide.
- ▷ **NumLevels** (input_control) ..... integer ⇝ *integer*
  Anzahl von Pyramidenebenen.
  Defaultwert: 4
  Werteliste: NumLevels ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **Contrast** (input_control) ..... number(-array) ⇝ *integer*
  Schwellenwert bzw. Hystereseschwellenwerte für den Kontrast des Objektes im Bild und optional Mindestgröße der Objektteile.
  Defaultwert: 30
  Wertevorschläge: Contrast ∈ {10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert inspect_shape_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

reduce_domain

Nachfolger

select_obj

Siehe auch

create_shape_model, create_scaled_shape_model, create_aniso_shape_model

Modul

Foundation

read_shape_model (: : FileName : ModelID)

Einlesen eines Formmodells von Datei.

read_shape_model liest ein Formmodell, das mit write_shape_model geschrieben wurde, aus der Datei FileName ein. Die Dateiendung für das Formmodell ist in HALCON 'shm'.

Parameter

- ▷ **FileName** (input_control) ..... filename.read ⇝ *string*
  Name der Datei.
  Dateiendung: .shm
- ▷ **ModelID** (output_control) ..... shape_model ⇝ *handle*
  Handle des Modells.

Ergebnis

Ist der Dateiname korrekt, dann liefert read_shape_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models, get_shape_model_clutter

Siehe auch

create_shape_model, create_scaled_shape_model, create_aniso_shape_model, clear_shape_model

Modul

Matching

serialize_shape_model (: : ModelID : SerializedItemHandle)

Serialisieren eines Formmodells.

serialize_shape_model serialisiert die Daten eines Formmodells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_shape_model in eine Datei geschrieben werden. Das Formmodell wird in dem Handle ModelID übergeben. Das serialisierte Formmodell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_shape_model wieder deserialisiert werden.

Parameter

▷ **ModelID** (input_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_shape_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_shape_model`, `create_scaled_shape_model`, `create_aniso_shape_model`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_shape_model`

Modul

Matching

set_generic_shape_model_object (Object : : ModelID, GenParamName :)

Setzen von Objektparametern eines Formmodells.

Der Operator `set_generic_shape_model_object` setzt Parameter des Formmodells `ModelID` basierend auf dem übergebenen `Object`.

Der Parameter `GenParamName` bestimmt, für welche Art von Einstellung das übergebene Objekt verwendet wird. Aktuell wird nur der folgende Wert unterstützt:

- *'high_noise_sample'*: Schätzt und passt die Parameter des Formmodells an um starkes Bildrauschen besser zu bewältigen. Die Parameterschätzung wird auf Basis des beispielhaften Suchbild `Object` durchgeführt. Daher sollte die Domäne von `Object` sowohl das Modell als auch Hintergrundumgebung, welche für die Suchbilder typisches Rauschen aufweist, beinhalten.
  Der Operator ändert folgende Parameter:
  - *'pyramid_level_lowest'*

Parameter

▷ **Object** (input_object) . . . . . (multichannel-)object(-array) ⇝ *object*
Objekt, auf dem das Setzen der Parameter basiert.

▷ **ModelID** (input_control) . . . . . shape_model ⇝ *handle*
Handle des Formmodells.

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Parametername.
Defaultwert: 'high_noise_sample'
Werteliste: `GenParamName` ∈ {'high_noise_sample'}

Ergebnis

Wird `set_generic_shape_model_object` erfolgreich ausgeführt, liefert der Operator den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_generic_shape_model`

Nachfolger

`train_generic_shape_model`, `find_generic_shape_model`

Modul

Matching

set_generic_shape_model_param (: : ModelID, GenParamName, GenParamValue :)

Setzen ausgewählter Parameter eines Formmodells.

Der Operator `set_generic_shape_model_param` setzt die Parameter `GenParamName` des Formmodells `ModelID` auf die in `GenParamValue` gegebenen Werte.

Verschiedene Aspekte des Matching-Vorgangs können über die in `GenParamName` auswählbaren Parameter gesteuert werden. In den Parameterbeschreibungen weiter unten sind diese nach den folgenden Kategorien gruppiert (wobei die Kategorie den Hauptverwendungszweck, aber nicht notwendigerweise den einzigen Zweck des jeweiligen Parameters angibt):

Modifizieren des Modells

Finden von Modellinstanzen
1. Modifizieren der Instanz
2. Aussortieren der gefundenen Matches
3. Grauwertbehandlung
4. Verfeinerung
5. Bildpyramide
6. Ausgabe

Rechnerressourcen für die Suche

Nachfolgend werden die für die verschiedenen Kategorien setzbaren Parameter gelistet und Fälle in denen Parameterwerte automatisch angepasst werden aufgeführt.

Modifizieren des Modells

In diesem Abschnitt werden die Parameter aufgelistet und erläutert, die das Modell `ModelID` selbst modifizieren. Wird ein zugehöriger Wert verändert, ist es notwendig das angepasste Modell vor dem Matching zu trainieren. Folgende Parameter `GenParamName` werden unterstützt:

Kontrast Der Kontrast wird als Maß für lokale Grauwertdifferenzen verwendet mit Hilfe derer Objekte vom Hintergrund oder Teile von Objekten untereinander unterschieden werden. Für das Modell werden nur die Pixel gewählt, welche den gewünschten Kontrasteigenschaften in ihrer Umgebung entsprechen. Der gewünschte Kontrast kann ein einfacher Schwellenwert oder ein Hysterese-Schwellenwert sein, je nach dem welche Werte für *'contrast_low'* und *'contrast_high'* gesetzt wurden:

- Einfacher Schwellenwert:
  - Beide Werte sind gleich, aber nicht *'auto'*.
  - Einer der Werte ist auf *'auto'* gesetzt, der andere auf einen anderen Wert.
- Hysterese-Schwellenwert:
  - Die beiden Werte sind nicht gleich und keiner der beiden ist auf *'auto'* gesetzt
  - Beide Werte sind auf *'auto'* gesetzt

In diesen Fällen wird das Modell mit Hilfe einer Methode segmentiert, welche ähnlich zum Hysterese-Schwellenwert-Verfahren in `edges_image` ist. Für weitere Informationen zum Hysterese-Schwellenwert-Verfahren siehe `hysteresis_threshold`.

Für weitere Informationen zu Kontrast und dessen Einfluss auf das Matching mit Formmodellen wird auf den „`Solution Guide II-B - Matching`“ verwiesen.

In bestimmten Fällen kann es vorkommen, dass die automatische Bestimmung des Kontrastschwellenwerts nicht ausreichend ist. Daher sollte in diesen Fällen ein manuelles Setzen der Kontrastparameter bevorzugt werden, beispielsweise falls bestimmte Modellkomponenten aus anwendungsspezifischen Gründen mit inbegriffen oder unterdrückt werden sollen, oder wenn das Objekt verschiedene Kontrasteigenschaften aufweist. Die Kontrast-Schwellenwerte sollten mit `inspect_shape_model` überprüft werden bevor das Modell an `train_generic_shape_model` übergeben wird.

- *'contrast_high'*:
  Bestimmt zusammen mit *'constrast_low'* den Grauwertbereich in dem ein Pixel liegen muss um als Teil des Modells berücksichtigt zu werden (siehe Beschreibung oben). Wird ein Hysterese-Schwellenwert-Verfahren genutzt bestimmt dieser Parameter den oberen Schwellenwert.
  Mögliche Werte: 'auto', *90*, *80*.
  Restriktion: Wird beim Training mit XLDs nicht genutzt.
  Default: 'auto'.
- *'contrast_low'*:
  Bestimmt zusammen mit *'constrast_high'* den Grauwertbereich in dem ein Pixel liegen muss um als Teil des Modells berücksichtigt zu werden (siehe Beschreibung oben). Wird ein Hysterese-Schwellenwert-Verfahren genutzt bestimmt dieser Parameter den unteren Schwellenwert.
  Mögliche Werte: 'auto', *10*, *20*.
  Restriktion: Wird beim Training mit XLDs nicht genutzt.
  Default: 'auto'.

Rotation Die Spanne möglicher Rotationen von Formmodellen liegt zwischen *0* und *6.28* (=2π), kann aber für die Suche eingeschränkt werden. Der Suchraum und damit die Diskretisierungsschritte für die Winkel kann modifiziert werden mit:

- *'angle_step'*:
  Bestimmt die Schrittweite innerhalb des gewählten Winkelbereichs. Der übergebene Wert sollte basierend auf der Objektgröße gewählt werden. Kleinere Modelle haben nicht viele verschiedene diskrete Rotationen im Bild, weshalb empfohlen wird für kleinere Modelle größere Werte zu wählen. Wird der Wert *'auto'* gesetzt, so wird die Schrittweite beim Aufruf von `train_generic_shape_model` geschätzt. Gesetzte Werte werden auf ein Maximum von *0.20* (=$\frac{\pi}{16}$) begrenzt.
  Mögliche Werte: 'auto', *0.1*, *0.2*.
  Zu beachten: Nach Modifikation ist ein Training notwendig.
  Default: 'auto'.

Skalierung Der Suchraum und somit der Suchbereich sowie die Diskretisierungsschritte für die Skalierung kann mit Skalierungsparametern modifiziert werden. Die folgenden Tabelle gibt einen Überblick darüber welche Parameter für welche Art von Skalierung gesetzt werden können. Ein Skalierungswert von *1.0* entspricht dabei der Originalgröße des Modells in der entsprechenden Richtung.

GenParamName	Isotrope Skalierung	Anisotrope Skalierung
'iso_scale_max'	x	
'iso_scale_min'	x	
'iso_scale_step'	x	
'scale_row_max'		x
'scale_row_min'		x
'scale_row_step'		x
'scale_column_max'		x
'scale_column_min'		x
'scale_column_step'		x

- *'iso_scale_max'*:
  Beschränkt das Maximum der isotropischen Skalierung bei der Modellsuche.
  Mögliche Werte: 1.0, *1.1*, *1.2*.
  Default: 1.0.

- *'iso_scale_min'*:
  Beschränkt das Minimum der isotropischen Skalierung bei der Modellsuche.
  Mögliche Werte: 1.0, *0.9*, *0.8*.
  Default: 1.0.
- *'iso_scale_step'*:
  Beschränkt die Schrittweite innerhalb des gewählten Bereichs der isotropischen Skalierung bei der Modellsuche. Wird der Wert auf *'auto'* gesetzt, dann wird die Schrittweite beim Aufruf von `train_generic_shape_model` geschätzt.
- *'scale_row_max'*:
  Beschränkt das Maximum der isotropischen Skalierung in Zeilenrichtung bei der Modellsuche.
  Mögliche Werte: 1.0, *1.1*, *1.2*.
  Default: 1.0.
- *'scale_row_min'*:
  Beschränkt das Minimum der isotropischen Skalierung in Zeilenrichtung bei der Modellsuche.
  Mögliche Werte: 1.0, *0.9*, *0.8*.
  Default: 1.0.
- *'scale_row_step'*:
  Beschränkt die Schrittweite innerhalb des gewählten Bereichs der isotropischen Skalierung in Zeilenrichtung bei der Modellsuche. Wird der Wert auf *'auto'* gesetzt, dann wird die Schrittweite beim Aufruf von `train_generic_shape_model` geschätzt.
- *'scale_column_max'*:
  Beschränkt das Maximum der isotropischen Skalierung in Spaltenrichtung bei der Modellsuche.
  Mögliche Werte: 1.0, *1.1*, *1.2*.
  Default: 1.0.
- *'scale_column_min'*:
  Beschränkt das Minimum der isotropischen Skalierung in Spaltenrichtung bei der Modellsuche.
  Mögliche Werte: 1.0, *0.9*, *0.8*.
  Default: 1.0.
- *'scale_column_step'*:
  Beschränkt die Schrittweite innerhalb des gewählten Bereichs der isotropischen Skalierung in Spaltenrichtung bei der Modellsuche. Wird der Wert auf *'auto'* gesetzt, dann wird die Schrittweite beim Aufruf von `train_generic_shape_model` geschätzt.

Metrik

- *'metric'*:
  Bestimmt die Bedingungen unter denen das Modell im Bild gefunden wird. Folgende Werte werden für *'metric'* unterstützt:
  - *'auto'*:
    Der Wert wird beim Aufruf von `train_generic_shape_model` in Abhängigkeit davon bestimmt, ob Bilder (*'use_polarity'*) oder XLDs (*'ignore_local_polarity'*) für das Training verwendet werden.
  - *'use_polarity'*:
    Die Instanz im Bild und das Modell müssen die selben Kontrasteigenschaften aufweisen.
    Restriktion: Wird nur auf Einkanalbilder angewandt. Für Mehrkanalbilder wird nur der erste Kanal genutzt (ohne Fehlermeldung).
    Beispiel: Das Modell wurde mit einem hellen Objekt auf dunklem Hintergrund erstellt. Nur Instanzen die ebenfalls heller als der Hintergrund sind werden gefunden.
  - *'ignore_global_polarity'*:
    Eine Instanz wird auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Die Laufzeit von `find_generic_shape_model` erhöht sich in diesem Modus geringfügig im Vergleich zu *'use_polarity'*.
    Restriktion: Wird nur auf Einkanalbilder angewandt. Für Mehrkanalbilder wird nur der erste Kanal genutzt (ohne Fehlermeldung).
    Beispiel: Das Modell wurde mit einem hellen Objekt auf dunklem Hintergrund erstellt. Instanzen die heller als oder dunkler als der Hintergrund sind werden gefunden.
  - *'ignore_local_polarity'*:

Eine Instanz wird auch dann im Bild gefunden, wenn sich der Kontrast lokal verändert. Die Laufzeit von `find_generic_shape_model` erhöht sich in diesem Modus signifikant im Vergleich zu *'use_polarity'*.
Es ist üblicherweise zu empfehlen mehrere Modelle zu verwenden, die die verschiedenen möglichen Kontrastverhältnisse des Objekts abdecken, und diese gleichzeitig für das Matching zu verwenden.
Restriktion: Wird nur auf Einkanalbilder angewandt. Für Mehrkanalbilder wird nur der erste Kanal genutzt (ohne Fehlermeldung).
Beispiel: Ein Teil des Models oder der Instanz liegt im mittleren Grauwertbereich und schließt Unterobjekte ein die entweder heller oder dunkler sein können.

- *'ignore_color_polarity'*:
  Eine Instanz wird auch dann im Bild gefunden, wenn sich der Farbkontrast lokal verändert. Die Laufzeit von `find_generic_shape_model` kann sich in diesem Modus signifikant im Vergleich zu *'use_polarity'* erhöhen.
  Diese Metrik kann für Bilder mit einer beliebigen Anzahl an Kanälen angewandt werden, welche keiner spektralen Unterteilung des Lichtes (wie z.B. bei einem RGB-Bild) entsprechen müssen. Für Einkanalbilder hat der Parameter denselben Effekt wie *'ignore_local_polarity'*. Die Anzahl Kanäle kann zwischen dem Training mit `train_generic_shape_model` und der Suche mit `find_generic_shape_model` variieren.
  Beispiel: Instanzen werden auch dann gefunden, wenn Teile des Objekts in verschiedenen Farben auftreten können, beispielsweise in rot und grün.

Restriction: Werden XLDs verwendet, dann is es für das Zurücksetzen der Metrik notwendig, dass die Informationen zur Kantenrichtung als XLD-Kontur-Attribut verfügbar ist. Für weitere Informationen siehe `set_shape_model_metric`.
Default: 'auto'.

Größe

- *'min_size'*:
  Bestimmt die minimale Anzahl an Punkten, die ein Modell aufweisen muss um berücksichtigt zu werden. Dementsprechend werden Komponenten, die weniger als *'min_size'* Punkte haben unterdrückt. Da *'min_size'* auf die Größe der Komponenten angewandt wird können die abgeleiteten Modellkonturen trotzdem kleiner als die festgelegte Minimalgröße sein. Dieser Schwellenwert für die minimale Größe wird für jede weitere Pyramidenstufe halbiert. Die Auswirkungen dieses Parameters können vorab mit `inspect_shape_model` überprüft werden.
  Wird der Wert auf *'auto'* gesetzt, dann wird der Wert beim Aufruf von `train_generic_shape_model` geschätzt.
  Restriktion: Wird beim Training mit XLDs nicht berücksichtigt.
  Default: 'auto'.
- *'optimization'*:
  Bestimmt die Anzahl an Punkten um die das Modell reduziert wird. Diese Reduktion kann besonders für sehr große Modelle nützlich sein. Mögliche Werte sind:
  - *'auto'*: Die Anzahl Punkte wird automatisch durch `train_generic_shape_model` bestimmt.
  - *'none'*: Alle Modellpunkte werden behalten.
  - Zahl: Anzahl an Punkten um die das Modell reduziert wird. In diesem Fall könnte es notwendig sein den Parameter *'greediness'* auf einen kleineren Wert zu setzen, z.B. *0.7* oder *0.8*.

  Es ist zu beachten, dass für kleine Modelle eine Reduktion der Punkteanzahl üblicherweise keine Beschleunigung ergibt, da dadurch die Anzahl potentieller Instanzen, die geprüft werden, steigt.
  Default: 'auto'.

Bildpyramide

- *'num_levels'*:
  Anzahl berücksichtigter Bildpyramidenstufen. Der Wert muss so gewählt werden, dass das Modell weiterhin erkennbar ist und ausreichend Punkte (mindestens vier) auf der höchsten Pyramidenstufe aufweist. Mit Hilfe der Ausgabe von `inspect_shape_model` kann dies überprüft werden. Es ist ratsam den Wert so hoch wie möglich zu setzen, da es die Laufzeit der Suche nach Instanzen signifikant verkürzt.
  Wird der Wert auf *'auto'* gesetzt, dann wird der Wert beim Aufruf von `train_generic_shape_model` geschätzt.

Es kann vorkommen, dass der automatisch bestimmte Wert nicht passend ist. Falls die Anzahl Pyramidenstufen zu hoch geschätzt wird könnte das Modell in den Bildern nicht erkannt werden, außer es werden sehr niedrige Werte für die Parameter *'min_score'* oder *'greediness'* gesetzt. Wird die Anzahl Pyramidenstufen zu niedrig geschätzt, steigt die Laufzeit der Suche mit `find_generic_shape_model`. In beiden Fällen sollte die Anzahl Pyramidenstufen mit Hilfe der Ausgabe von `inspect_shape_model` ausgewählt werden.
Default: 'auto'.

Störungen

- *'use_clutter'*:
  Bestimmt ob das Modell `ModelID` Störparameter berücksichtigt (*'true'*) oder nicht (*'false'*). Die Laufzeit mit Nutzung von Störparametern ist mindestens so hoch wie die Laufzeit ohne Störparameter aber Rückgabe aller Matches, d.h., mit auf *'all'* gesetztem Parameter *'num_matches'*.
  Werteliste: *'false'*, *'true'*.
  Default: 'false'.

Finden von Modellinstanzen

In diesem Abschnitt werden die Parameter aufgelistet und erläutert, die die Suche mit dem Modell `ModelID` modifizieren. Wird ein zugehöriger Wert verändert ist es nicht notwendig das Modell zu trainieren, außer ein zuvor nicht durch automatische Schätzung ermittelter Wert soll nun von `train_generic_shape_model` automatisch geschätzt werden. Die Beschreibungen dieser Parameter enthalten eine entsprechende Bemerkung. Folgende Parameter `GenParamName` werden unterstützt:

1. **Modifizieren der Instanz**

   Rotation

   - *'angle_start'*:
     Bestimmt den Startwert des Winkelbereichs möglicher Rotationen, für die das Modell gesucht wird.
     Mögliche Werte: 0.0, *-3.14*, *3.14*.
     Default: 0.0.
   - *'angle_end'*
     Bestimmt den Endwert des Winkelbereichs möglicher Rotationen, für die das Modell gesucht wird.
     Mögliche Werte: 0.0, *3.14*, *6.28*.
     Default: 6.28 (= 2π).

   Die Werte werden auf den Suchbereich abgebildet. *Beispiel: 'angle_start' = '-rad(10)'*, *'angle_end' = 'rad(10)'* sucht nach Matches in den Winkelbereichen [rad(350), rad(360)] und [0, rad(10)].

   Skalierung Der Skalierungsbereich, der für das Modell bestimmt wird kann weiter eingeschränkt werden. Wird für diese Parameter der Wert auf *'auto'* gesetzt, dann wird der Wert beim Aufruf von `train_generic_shape_model` geschätzt.

`GenParamName`	Isotrope Skalierung	Anisotrope Skalierung
'restrict_iso_scale_max'	x	
'restrict_iso_scale_min'	x	
'restrict_scale_row_max'		x
'restrict_scale_row_min'		x
'restrict_scale_column_max'		x
'restrict_scale_column_min'		x

   - *'restrict_iso_scale_max'*:
     Beschränkt den möglichen Skalierungsbereich für den das Modell gesucht wird auf ein Maximum.
     Mögliche Werte: 'auto', *1.1*, *1.2*.
     Default: 'auto'.
   - *'restrict_iso_scale_min'*:
     Beschränkt den möglichen Skalierungsbereich für den das Modell gesucht wird auf ein Minimum.
     Mögliche Werte: 'auto', *0.9*, *0.8*.
     Default: 'auto'.

- *'restrict_scale_row_max'*:
  Beschränkt den möglichen Skalierungsbereich in Zeilenrichtung für den das Modell gesucht wird auf ein Maximum.
  Mögliche Werte: 'auto', *1.1*, *1.2*.
  Default: 'auto'.
- *'restrict_scale_row_min'*:
  Beschränkt den möglichen Skalierungsbereich in Zeilenrichtung für den das Modell gesucht wird auf ein Minimum.
  Mögliche Werte: 'auto', *0.9*, *0.8*.
  Default: 'auto'.
- *'restrict_scale_column_max'*:
  Beschränkt den möglichen Skalierungsbereich in Spaltenrichtung für den das Modell gesucht wird auf ein Maximum.
  Mögliche Werte: 'auto', *1.1*, *1.2*.
  Default: 'auto'.
- *'restrict_scale_column_min'*:
  Beschränkt den möglichen Skalierungsbereich in Spaltenrichtung für den das Modell gesucht wird auf ein Maximum.
  Mögliche Werte: 'auto', *0.9*, *0.8*.
  Default: 'auto'.

Deformation

- *'max_deformation'*:
  Bestimmt um wie viel ein Objekt vom Modell abweichen darf um als Match berücksichtigt zu werden. Die maximal zulässige Objektdeformation wird in Pixeln festgelegt. Für einen Wert von *0* wäre daher keine Abweichung für das Objekt zulässig.
  Beispiel: Ein Objekt, dessen Form im Vergleich zum Formmodell um 2 Pixel deformiert ist wird mit *'max_deformation' = 2* noch gefunden.
  Höhere Werte in *'max_deformation'* erhöhen häufig auch die Laufzeit.
  Höhere Werte in *'max_deformation'* erhöhen außerdem häufig auch das Risiko ungewollte Matches zu finden. Dies ist insbesondere der Fall für kleine Objekte oder Objekte mit feinen Strukturen, da diese somit ihre charakteristische Form, welche wichtig für eine robuste Suche ist, verlieren.
  Zusammenspiel mit *'subpixel'*: Wenn eine Deformation erlaubt ist (also ein Wert größer *0* gesetzt wurde) dann hängt die Berechnung des Scores davon ab ob möglicherweise eine Subpixel-Verfeinerung durchgeführt wurde. In den meisten Fällen ändert sich der Score wenn eine Least-Squares-Adjustment-Methode gesetzt wurde. Mit dieser Methode kann ein höherer Score erreicht werden wenn *'max_deformation'* erhöht wird, da für die Modellpunkte mehr übereinstimmende Bildpunkte gefunden werden können.
  Default: 0.

2. **Aussortieren der gefundenen Matches**

- *'min_score'*:
  Bestimmt den minimalen Score den ein möglicher Match aufweisen muss um als Instanz im Bild zu gelten. Umso höher *'min_score'* gesetzt wird, desto schneller ist die Suche. Falls zu erwarten ist, dass das Modell in den Bildern nie verdeckt auftritt, kann *'min_score'* auf *0.8* oder sogar *0.9* gesetzt werden. Falls die Matches nicht bis zur untersten Pyramidenstufe verfolgt werden kann es sein, dass Instanzen mit Scores knapp unter *'min_score'* zurückgegeben werden.
  Das Zusammenspiel mit *'num_matches'* wird in der Beschreibung zu Letzterem erklärt.
  Default: 0.5.
- *'num_matches'*:
  Bestimmt die maximale Anzahl zurückgegebener Matches.
  Der Wert *'all'* (oder *0*) bedeutet, dass alle Instanzen zurückgegeben werden, die einen Score größer als *'min_score'* erreichen. Ansonsten ist das Zusammenspiel zwischen *'num_matches'* und *'min_score'* sowie *'max_clutter'* (falls gesetzt) wie folgt:
  - Keine Störparameter gesetzt:
    Für den Fall dass mehr als *'num_matches'* Instanzen gefunden werden, die den erforderlichen minimalen Score erreichen, werden nur die *'num_matches'* Matches mit den höchsten Scores zurückgegeben.

Falls weniger als *'num_matches'* Instanzen mit dem erforderlichen Score gefunden werden, werden nur die Instanzen zurückgegeben, die die Anforderungen an den Score erfüllen.

– Störparameter gesetzt:
Für den Fall dass mehr als *'num_matches'* Instanzen gefunden werden, die den erforderlichen minimalen Score und einen akzeptablen Störwert erreichen, werden nur die *'num_matches'* Matches mit den besten Störwerten zurückgegeben.
Falls weniger als *'num_matches'* Instanzen mit dem erforderlichen Score und / oder akzeptablen Störwert gefunden werden, werden nur die Instanzen zurückgegeben, die die Anforderungen an den Score und den Störwert erfüllen.

Das Verfolgen von Matches durch die Bildpyramide hängt davon ab ob Störparameter gesetzt wurden:

– Keine Störparameter gesetzt:
Auf jeder Stufe werden weniger versprechende Matches basierend auf *'num_matches'* verworfen. Dabei können Matches verworfen werden die auf der niedrigsten Pyramidenstufe einen höheren Score erreicht hätten. Daher kann sich das Ergebnis für beispielsweise *'num_matches'* = *1* von der Instanz mit dem höchsten Score unterscheiden, wenn *'num_matches'* auf *'all'* oder einen Wert größer *1* gesetzt wird. Folglich kann im Fall, dass mehrere Objekte mit ähnlichem Score zu erwarten sind, aber nur der beste Match gewählt werden soll, ratsam sein mehrere Instanzen zurückzugeben und dann den Match mit dem höchsten Score auszuwählen.

– Störparameter gesetzt:
Beim Verfolgen der Matches werden keine Matches verworfen. Die Laufzeit mit Nutzung von Störparametern ist mindestens so hoch wie die Laufzeit ohne Störparameter aber Rückgabe aller Matches, d.h., mit auf *'all'* gesetztem Parameter *'num_matches'*.

Default: 'all'.

- *'strict_boundaries'*:
Bestimmt ob gefundene Instanzen nur dann zurückgegeben werden wenn sie strikt innerhalb des gegebenen Suchbereichs liegen (*'true'*) oder nicht (*'false'*). Es ist zu beachten, dass dieses Auswahlkriterium nach der Reduktion der Matches auf die besten *'num_matches'* Matches angewandt wird und daher in Extremfällen dazu führen kann, dass kein Ergebnis zurückgegeben wird.
Kompatibilität: Wird dieser Wert für ein Modell gesetzt, das nicht mit `create_generic_shape_model` erstellt wurde, werden interne Modelleinstellungen verändert. Folglich kann unterschiedliches Verhalten bezüglich Matches außerhalb des eingestellten Suchbereichs auftreten wenn für solche Modelle *'strict_boundaries'* auf *'true'* und wieder zurück *'false'* gesetzt wird.
Default: 'false'.
- *'max_overlap'*:
Bestimmt zu welchem Anteil zwei Instanzen höchstens überlappen dürfen um als unterschiedliche Instanzen zu gelten und daher separat zurückgegeben zu werden. Falls sich zwei Instanzen zu einem Anteil von mehr als *'max_overlap'* überlappen wird nur diejenige mit dem höheren Score zurückgegeben. Die Berechnung der Überlappung basiert auf dem kleinsten umschließenden Rechteck beliebiger Orientierung (siehe `smallest_rectangle2`) der gefundenen Instanzen. Wird *'max_overlap'* auf *0* gesetzt, so dürfen gefundene Instanzen überhaupt nicht überlappen, während für *'max_overlap'* = *1* alle Instanzen zurückgegeben werden, unabhängig von möglicher Überlappung.
Restriktion: $0 \leq$ *'max_overlap'* ≤ 1.
Default: 0.5.
- *'max_clutter'*:
Bestimmt den maximalen Störwert den ein möglicher Match aufweisen darf um als Modellinstanz im Bild in Frage zu kommen. Dies gilt natürlich nur für Modelle die Störungen berücksichtigen, also wo *'use_clutter'* auf *'true'* gesetzt wurde.
Default: 0.0.

3. **Grauwertbehandlung**

- *'min_contrast'*:
Bestimmt den minimalen Kontrast (Grauwertdifferenz zu Nachbarpixeln) den ein Punkt im Suchbild mindestens aufweisen muss um mit dem Modell verglichen zu werden. Der Hauptverwendungszweck dieses Parameters ist es Rauschen im Bild für den Matching-Vorgang zu unterdrücken.
Beispiel: Das Rauschen im Bild weist Grauwertschwankungen im Bereich von 10 Grauwerten auf. In diesem Fall ist es ratsam *'min_contrast'* auf *10* zu setzen.

Mehrkanalige Bilder: Für eine Suche in mehrkanaligen Bildern mit Hilfe von *'metric'* = *'ignore_color_polarity'*, muss das Rauschen in einem Channel multipliziert werden und zwar mit der Quadratwurzel aus der Anzahl an Kanälen für die *'min_contrast'* bestimmt werden soll.
Beispiel: Das Rauschen in einem Kanal eines dreikanaligen Bildes weist Grauwertschwankungen im Bereich von 10 Grauwerten auf. In diesem Fall ist es ratsam *'min_contrast'* auf *17* zu setzen.
Es ist zu beachten, dass *'min_contrast'* kleiner sein muss als der Kontrast-Schwellenwert der mit *'contrast_low'* und *'contrast_high'* gesetzt wird, siehe oben.
Der Wert *'auto'* bedeutet, dass der Wert beim Aufruf von `train_generic_shape_model` basierend auf dem Rauschen im Modellbild geschätzt wird. Für eine sinnvolle automatische Schätzung muss daher das Rauschen im Modellbild ähnlich zum Rauschen in den Suchbildern sein.
Training: Wird dieser Wert auf *'auto'* gesetzt, dann ist ein Training des Formmodells notwendig.
Mögliche Werte: *'auto'*, *10*, *20*.
Default: 'auto'.

- *'border_shape_models'*:
  Bestimmt ob das Formmodell `ModelID` nach dem beispielsweise mit `find_generic_shape_model` gesucht wird teilweise außerhalb des Bildes liegen darf (d.h. ob der Bildbereich, unabhängig von der Domäne, überschritten werden darf). Der Wert *'system'* bedeutet hierbei, dass das Modell den systemweiten Wert, der über `set_system` für *'border_shape_models'* gesetzt wurde, verwendet.
  Werteliste: *'system'*, *'true'*, *'false'*.
  Default: 'system'.

4. **Verfeinerung**

   - *'subpixel'*:
     Bestimmt welche Art der Subpixel-Verfeinerung ausgeführt werden soll. Mögliche Werte:
     - *'none'*: Die Pose des Modells wird lediglich mit Pixelgenauigkeit und der in *'angle_step'* angegebenen Winkelauflösung bestimmt.
     - *'interpolation'*: Die Pose des Modells wird aus der Score-Funktion interpoliert. Dieser Modus zeichnet sich durch sehr niedrigen Berechnungsaufwand aus.
     - *'least_squares'*: Die Pose des Modells wird mit der Least-Squares-Adjustment-Methode verfeinert, d.h. die Distanzen zwischen den Modellpunkten und den entsprechenden Bildpunkten werden minimiert.
     - *'least_squares_high'*: Die Pose des Modells wird mit der Least-Squares-Adjustment-Methode verfeinert, d.h. die Distanzen zwischen den Modellpunkten und den entsprechenden Bildpunkten werden minimiert. Hierbei sind mehr Iterationsschritte als für *'least_squares'* zugelassen, wodurch sich die Laufzeit erhöhen kann.
     - *'least_squares_very_high'*: Die Pose des Modells wird mit der Least-Squares-Adjustment-Methode verfeinert, d.h. die Distanzen zwischen den Modellpunkten und den entsprechenden Bildpunkten werden minimiert. Hierbei sind mehr Iterationsschritte als für *'least_squares_high'* zugelassen, wodurch sich die Laufzeit erhöhen kann.

     Das Zusammenspiel mit *'max_deformation'* wird in der Beschreibung zu Letzterem erklärt.
     Default: 'least_squares'.
   - *'greediness'*:
     Bestimmt wie „gierig“ die Suche ausgeführt werden soll. Ein höherer Wert resultiert in einer schnelleren Suche, welche allerdings eine weniger sichere Suchheuristik verwendet.
     Restriktion: $0 \leq$ *'greediness'* ≤ 1.
     Default: 0.9.

5. **Bildpyramide**

   - *'pyramid_level_highest'*:
     Bestimmt die höchste Stufe der Bildpyramide die für die Suche verwendet wird. Damit kann die im Modell gesetzte höchste Pyramidenstufe für die Suche heruntergesetzt werden. Der Wert kann nicht höher sein als die höchste Pyramidenstufe des Modells, welche mit *'num_levels'* gesetzt wird. Höhere Werte für *'pyramid_level_highest'* werden entsprechend reduziert. Der Wert *0* bedeutet, dass die Stufe für die Suche nicht heruntergesetzt wird und das Modell dementsprechend die in *'num_levels'* gesetzte höchste Pyramidenstufe nutzt.
     Default: 0.
   - *'pyramid_level_lowest'*:

Bestimmt die niedrigste Stufe der Bildpyramide zu der die gefundenen Matches verfolgt werden. Werden die Matches nicht bis auf die unterste Stufe der Bildpyramide verfolgt verringert sich die Laufzeit. Im Allgemeinen ist die Genauigkeit der erhaltenen Posen allerdings um so niedriger je höher die niedrigste Stufe gesetzt wird. In Extremfällen kann dies auch zum Finden falscher Instanzen führen, für die die gewünschte Genauigkeit nicht erreicht werden kann.
Default: 1.

- *'pyramid_level_robust_tracking'*:
  Bestimmt ob der sogenannte „increased tolerance"-Modus aktiviert ist (*'true'*) oder nicht (*'false'*). In diesem Modus wird die niedrigste Pyramidenstufe auf der mindestens ein Match gefunden wird während der Suche automatisch bestimmt. Für den Fall dass Eingabebilder in schlechter Qualität vorliegen, beispielsweise unscharf, verzerrt oder verrauscht, kann dieser Modus hilfreich sein. In diesen Fällen können die Kanteninformationen auf der niedrigsten Pyramidenstufe fehlen oder verzerrt sein und dazu führen, dass keine Instanzen des Formmodells gefunden werden. Nichtsdestotrotz kann die Kanteninformation auf höheren Pyramidenstufen ausreichend sein. Die Wahl der passenden Pyramidenstufe, d.h. der niedrigsten Pyramidenstufe auf der mindestens ein Match gefunden wird, hängt vom Modell und dem Eingabebild ab. Natürlich gelten auch hier die für *'pyramid_level_lowest'* genannten Einschränkungen bezüglich Genauigkeit und Robustheit.
  Beispiel: *'pyramid_level_highest'* wird auf *4*, *'pyramid_level_lowest'* auf *2* und *'pyramid_level_robust_tracking'* auf *'true'* gesetzt. Das Matching startet nun auf der vierten Pyramidenstufe und verfolgt die Matches bis zur zweiten Stufe. Falls keine Instanz des Modells auf der zweiten Pyramidenstufe gefunden wird, aber Matches auf höheren Pyramidenstufen gefunden wurden, dann wird die niedrigste Pyramidenstufe auf die Stufe gesetzt auf der mindestens eine Instanz des Modells gefunden wurde. Die Instanzen dieser Pyramidenstufe werden dann zurückgegeben.
  Default: 'false'.

6. **Ausgabe**

   - Ursprung:
     Der Ursprung (Referenzpunkt) ist relativ zum Schwerpunkt der Domäne (Region) des Bildes definiert, das zum Trainieren des Formmodells mit `train_generic_shape_model` benutzt wurde. Daher bedeutet ein Ursprung von (0,0), dass der Schwerpunkt der Domäne des Template-Bildes als Ursprung verwendet wird. Der Ursprung kann mit folgenden Parametern neu gesetzt werden:
     - *'origin_row'* Bestimmt die Zeilenkoordinate des neuen, verschobenen Ursprungs.
       Default: 0.
     - *'origin_column'*: Bestimmt die Spaltenkoordinate des neuen, verschobenen Ursprungs.
       Default: 0.

     Beispiel: Setzt man *'origin_column'* = *-20* und *'origin_row'* = *-40*, dann liegt der neue Ursprung links oberhalb des Schwerpunkts des Modells.
   - *'prepare_contours_for_visualization'*
     Bestimmt, ob die Kontur eines gefundenen Matches mit `find_generic_shape_model` für die Visualisierung vorbereitet und in *'MatchResultID'* ausgegeben werden soll (*'true'*) oder nicht (*'false'*). Durch die Deaktivierung von *'prepare_contours_for_visualization'* verkürzt sich die Laufzeit. In der Dokumentation von `get_generic_shape_model_result_object` wird beschrieben wie die generierte Kontur abgefragt werden kann.
     Werteliste: *'true'*, *'false'*.
     Default: 'true'.
   - *'prepare_clutter_region_for_visualization'*
     Bestimmt, ob die Störregion eines gefundenen Matches mit `find_generic_shape_model` für die Visualisierung vorbereitet und in *'MatchResultID'* ausgegeben werden soll (*'true'*) oder nicht (*'false'*). Durch die Deaktivierung von *'prepare_clutter_region_for_visualization'* verkürzt sich die Laufzeit. In der Dokumentation von `get_generic_shape_model_result_object` wird beschrieben wie die generierte Störregion abgefragt werden kann.
     Werteliste: *'true'*, *'false'*.
     Default: 'true'.

Rechnerressourcen für die Suche

In diesem Abschnitt werden die Parameter aufgelistet und erläutert, welche technische Einflüsse beim Suchen des Modells `ModelID` betreffen. Folgende Parameter `GenParamName` werden unterstützt:

- *'model_cache'*:

Bestimmt, ob z.B. `find_generic_shape_model` einen internen Cache, der auf temporärem Speicher basiert, verwendet (*'true'*) oder nicht (*'false'*). Wird *'model_cache'* deaktiviert (durch Setzen von *'false'*), kann die Laufzeit leicht verlängert werden, führt aber zu einem konstant kleineren Speicherverbrauch.

Die Größe des Cache hängt von der Größe des Suchraums ab, also ob der Suchbereich viele Diskretisierungsschritte aufweist. Das bedeutet, dass aus einem engmaschiges Suchgitter mit kleinen Schrittweiten (`step`), bei dem große Bereiche zwischen Start- (`start`) und Endwerten (`end`) abgedeckt werden, großer Speicherverbrauch resultiert.

Default: 'true'.

- *'timeout'*:

  Bestimmt die maximale Laufzeit der Operatoren, die zum Suchen des Formmodells `ModelID` verwendet werden (z.B. `find_generic_shape_model`). Die zeitliche Genauigkeit hängt von einigen Faktoren ab, darunter die Größe des Modells, die Geschwindigkeit des verwendeten Rechners, und der Modus der in *'timer_mode'* über `set_system` gesetzt wurde. Es ist zu beachten, dass die Laufzeit der Suche bis zu 10 Prozent ansteigt wenn *'timeout'* aktiviert ist.

  Der Wert von *'timeout'* muss in Millisekunden angegeben werden. Um *'timeout'* zu deaktivieren kann entweder eine negative Zahl oder *'false'* gesetzt werden.

  Mögliche Werte: *'false'*, *-1*, *100*.

  Default: 'false'.

Automatische Modifikationen

In diesem Abschnitt werden die Parameter aufgelistet, die ohne weitere Benachrichtigung automatisch modifiziert werden:

- *'num_levels'*:

  Falls *'num_levels'* so gesetzt wird, dass auf der gewünschten Pyramidenstufe nicht mehr ausreichend Modellpunkte generiert werden, dann wird die Anzahl Stufen intern so reduziert, dass auf der obersten Pyramidenstufe ausreichend Modellpunkte gefunden werden können.

  Falls dieses Vorgehen zu einem Modell ohne Pyramidenstufen führen würde, die Anzahl Modellpunkte also von vornherein auf der untersten Pyramidenstufe zu klein war, dann gibt `train_generic_shape_model` eine Fehlermeldung aus.

- Parameter, die eine Schrittweite (*step_size*) bestimmen:

  Falls der Suchbereich kein Vielfaches der Schrittweite ist wird letztere entsprechend angepasst.

  Beispiele: *'angle_step'*, *'iso_scale_step'*.

- Start- und Endwerte (*start* und *end*) eines Suchbereichs:

  Um sicher zu gehen, dass Modellinstanzen in ihrer exakten Modellposition von `find_generic_shape_model` zurückgegeben werden, können Suchbereiche wie folgt angepasst werden: Falls keine positive Ganzzahl n existiert, mit welcher der Startwert plus n mal die Schrittweite (*step_size*) 0.0 (für Winkel) beziehungsweise 1.0 (für Skalierungen) ergibt, dann wird der Startwert um bis zu eine Schrittweite verringert und der Endwert wird erhöht, so dass der Suchbereich um genau einmal die Schrittweite erweitert wurde.

  Beispiele: *'angle_start'*, *'restrict_iso_scale_max'*.

Es ist zu beachten das Transformationen intern so berechnet werden, dass die Skalierung und dann die Rotation angewandt wird. Daher sollte das Modell üblicherweise so ausgerichtet werden, dass es horizontal oder vertikal im Modellbild auftritt.

Parameter

▷ **`ModelID`** (input_control) . . . . . shape_model ⇝ *handle*
Handle des Formmodells.

▷ **`GenParamName`** (input_control) . . . . . attribute.name-array ⇝ *string*
Parameternamen.
Defaultwert: 'min_score'
Werteliste: `GenParamName` ∈ {'angle_end', 'angle_start', 'angle_step', 'border_shape_models', 'contrast_high', 'contrast_low', 'greediness', 'iso_scale_max', 'iso_scale_min', 'iso_scale_step', 'max_clutter', 'max_deformation', 'max_overlap', 'metric', 'min_contrast', 'min_score', 'min_size', 'model_cache', 'num_levels', 'num_matches', 'optimization', 'origin_column', 'origin_row',

'prepare_contours_for_visualization', 'prepare_clutter_region_for_visualization', 'pyramid_level_highest', 'pyramid_level_lowest', 'pyramid_level_robust_tracking', 'restrict_iso_scale_max', 'restrict_iso_scale_min', 'restrict_scale_column_max', 'restrict_scale_column_min', 'restrict_scale_row_max', 'restrict_scale_row_min', 'scale_column_max', 'scale_column_min', 'scale_column_step', 'scale_row_max', 'scale_row_min', 'scale_row_step', 'strict_boundaries', 'subpixel', 'timeout', 'use_clutter'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Parameterwerte.
Defaultwert: 0.5
Wertevorschläge: GenParamValue ∈ {-3.14, -1.57, -0.79, -0.39, 0.0, 0.39, 0.79, 1.57, 3.14}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_generic_shape_model_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_generic_shape_model

Nachfolger

train_generic_shape_model, find_generic_shape_model

Modul

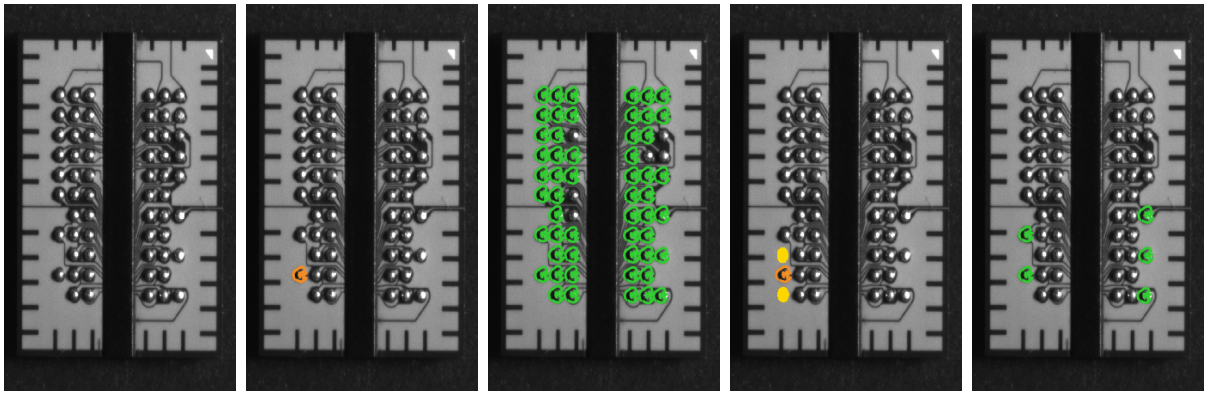
Matching

set_shape_model_clutter (ClutterRegion : : ModelID, HomMat2D, ClutterContrast, GenParamName, GenParamValue :)

Setzen der Störparameter eines Formmodells.

set_shape_model_clutter setzt die Störparameter eines Formmodells ModelID, d.h. es wird eine Region relativ zum Formmodell definiert, in der im Suchbild keine (oder zu schwache) Störungen auftreten sollen. Hilfreich ist dies beispielsweise, wenn in einem Suchbereich mit sich wiederholendem Muster nach einer Instanz des Modells gesucht wird, in deren Umgebung ein Teil des Musters nicht auftritt.

Das HDevelop Beispielprogramm find_shape_model_clutter.hdev zeigt z.B. wie mit Hilfe von Störparametern spezifische Matches in Bildern von Kugelgitteranordnungen (BGA) gefunden werden können.



(1) (2) (3) (4) (5)

(1) Bild des BGA, (2) Formmodell einer einzelnen Kugel (orange), (3) Matchingergebnis (grün) ohne Hilfe von Störparametern, (4) Definieren einer Störregion (gelb) relativ zu der Modelregion (orange). (5) Matchingergebnis nach Setzen von Störparametern (grün).

Für Matches im Suchbild erhöhen Kanten, die in der Störregion gefunden werden, den Störwert. Dieser wird von den Operatoren `find_shape_model`, `find_scaled_shape_model`, `find_aniso_shape_model`, `find_shape_models`, `find_scaled_shape_models` und `find_aniso_shape_models` zusätzlich im Parameter `Score` ausgegeben, nach den üblichen Werten aus der Berechnung der Sichtbarkeit des Modells. Es ist zu beachten, dass im Eingabeparameter `MinScore` dieser Operatoren zusätzlich Angaben zum maximalen Störwert (für welchen Matches zurückgegeben werden sollen) gemacht werden. Für weitere Informationen zum Setzen von `MinScore`, siehe die entsprechende Operatorreferenz.

Um die Störregion relativ zu den Modellkonturen zu definieren, braucht man zum einen die Region `ClutterRegion` und zum anderen die Transformationsmatrix `HomMat2D`, welche die Position der Modellkonturen im Suchbild abbildet. Ein typisches Vorgehen um eine geeignete Transformationsmatrix zu erhalten wäre, eine Instanz im Bild zu finden (beispielsweise mit `find_shape_model`) und aus den Ausgabeparametern mit Hilfe der Prozedur `get_hom_mat2d_from_matching_result` die Transformationsmatrix `HomMat2D` zu berechnen. Im Anschluss kann `ClutterRegion` für die gefundene Instanz im gleichen Bild spezifiziert werden. Mit `ClutterRegion` sollten dabei sämtliche Störungen, welche typischerweise im Umfeld der gefundenen Instanzen auftreten, abgedeckt sein. Es wird empfohlen die Störregion größer als notwendig zu spezifizieren wenn die Skalierungen des Formmodells stark variieren. Die Robustheit der Suche kann verbessert werden indem die Störregion nicht zu nah an den erwarteten Modellkonturen gewählt wird.

Der Parameter `ClutterContrast` bestimmt den Kontrast, den eine Kante in der Störregion aufweisen muss, um als Störung gewertet zu werden. In vielen Anwendungsfällen ist der Parameter `Contrast`, welcher zur Erstellung des Formmodells genutzt wurde, auch hierfür ein sinnvoller Wert. `ClutterContrast` darf dabei nicht kleiner sein als der Parameter `MinContrast` des Formmodells, ansonsten wird eine Fehlermeldung zurückgegeben. Die Polarität der gefundenen Störkanten wird ignoriert, d.h. helle Objekte auf dunklem Hintergrund erzielen den selben Störwert wie dunkle Objekte auf hellem Hintergrund, unabhängig vom Parameter `Metric` des Formmodells. Es ist zu beachten, dass von sämtlichen Ergebnissen des formbasierten Matchings die Störwerte am stärksten von Beleuchtungsänderungen beeinflusst werden.

Wird `GenParamName` auf *'clutter_border_mode'* gesetzt, kann das Verhalten des Störwertes für den Fall, dass die Störregion nicht komplett im Bildbereich enthalten ist, beeinflusst werden. Als entsprechende Werte für *'clutter_border_mode'* (`GenParamValue`) kommen *'clutter_border_average'* oder *'clutter_border_empty'* in Frage.

- *'clutter_border_average'*:
  Wenn `GenParamValue` auf *'clutter_border_average'* (Standardwert) gesetzt wird, wird angenommen, dass der verdeckte Teil durchschnittlich den selben Störanteil aufweist wie der sichtbare Teil. Ist die Störregion gar nicht zu sehen, wird der Störwert des gefundenen Matches auf *0.0* gesetzt.
- *'clutter_border_empty'*:
  Wird `GenParamValue` auf *'clutter_border_empty'* gesetzt, wird angenommen, dass der verdeckte Teil der Störregionen keine Störungen enthält.

Es ist zu beachten, dass mit *'clutter_border_average'* für `GenParamValue` höhere Störwerte erreicht werden.

Die Nutzung von Störparametern im Formmodell kann mit `set_shape_model_param` aktiviert oder deaktiviert werden. Für neu generierte Formmodelle ist die Nutzung von Störparametern deaktiviert. Nach dem Aufruf von `set_shape_model_clutter` ist die Nutzung von Störparametern aktiviert. Je nach Aktivierungsstatus erwarten die jeweiligen find-Operatoren eine unterschiedliche Anzahl an Eingabeparametern für `MinScore` (siehe auch die entsprechenden Operatorreferenzen). Die gesetzten Störparameter sowie der Wert von *'use_clutter'* können mit dem Operator `get_shape_model_clutter` abgefragt werden.

Parameter

▷ **`ClutterRegion`** (input_object) ..... region ⇝ *object*
Region, in der keine Störungen auftreten sollen.

▷ **`ModelID`** (input_control) ..... shape_model ⇝ *handle*
Handle des Modells.

▷ **`HomMat2D`** (input_control) ..... hom_mat2d ⇝ *real*
Transformationsmatrix.

▷ **ClutterContrast** (input_control) ..... number ⇝ *integer*
Minimaler Kontrast für Störungen im Suchbild.
Defaultwert: 128

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Parameternamen.
Werteliste: GenParamName ∈ {'clutter_border_mode'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *real* / integer / string
Parameterwerte.
Werteliste: GenParamValue ∈ {'clutter_border_empty', 'clutter_border_average'}

Beispiel

```
*
* Create a shape model.
read_image (ImageModel, '/bga_gap/bga_gap_01.png')
gen_circle (ROI, 753.869, 551.624, 28.4027)
reduce_domain (ImageModel, ROI, ImageReduced)
create_aniso_shape_model (ImageReduced, 'auto', rad(0), rad(0), 'auto', 0.95, 1.05, 'aut
*
* Specify the clutter parameters.
find_aniso_shape_model (ImageModel, ModelID, rad(0), rad(0), 0.95, 1.05, 0.95, 1.05, 0.8
get_hom_mat2d_from_matching_result (Row[0], Column[0], Angle[0], ScaleR[0], ScaleC[0], H
*
gen_circle (ROI_0, 700.655, 548.666, 21.6273)
gen_circle (ROI_1_0, 810.655, 550.611, 21.6273)
union2 (ROI_0, ROI_1_0, ClutterRegion)
*
ClutterContrast := 12
*
* Set the clutter parameters into shape model.
set_shape_model_clutter (ClutterRegion, ModelID, HomMat2D, ClutterContrast, [], [])
*
* Use the shape model to detect objects with a small amount of clutter
read_image (Image,'/bga_gap/bga_gap_02.png')
MaxClutter := 0.09
find_aniso_shape_model (Image, ModelID, rad(0), rad(0), 0.95, 1.05, 0.95, 1.05, [0.83, M
*
* Visualize the matches
dev_display_shape_matching_results (ModelID, ['green', 'red'], Row, Column, Angle, Scale
Clutter := Score[|Score|/2:|Score|-1]
Score := Score[0:|Score|/2-1]
dev_inspect_ctrl ([Score, Clutter])
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_shape_model_clutter den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_shape_model, create_scaled_shape_model, create_aniso_shape_model, create_shape_model_xld, create_scaled_shape_model_xld, create_aniso_shape_model_xld

Nachfolger

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models, get_shape_model_clutter

Siehe auch

set_shape_model_param

Modul

Matching

set_shape_model_metric (Image : : ModelID, HomMat2D, Metric :)

Setzen der Metrik eines Formmodells, das aus XLD-Konturen erzeugt wurde.

set_shape_model_metric ändert den Wert des Parameters Metric der bei der Erstellung des Formmodells ModelID gesetzt wurde. Hierfür wird die Polarität der Modellkanten unter Verwendung des Bildes Image bestimmt. Die Transformationsmatrix HomMat2D beschreibt die Transformation, die die Modellkanten an die entsprechende Stelle des Bildes transformiert, an der das Objekt erscheint.

Der Parameter Metric legt fest, unter welchen Bedingungen das Muster später im Suchbild noch erkannt wird. Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist. Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_shape_model erhöht sich in diesem Fall geringfügig.

Es muss sichergestellt werden, dass die Objektkanten im Bild Image die gleiche (oder inverse) Polarität aufweisen wie die entsprechenden Kanten im späteren Suchbild. Insbesondere darf das Objekt nicht verdeckt sein und der Hintergrund muss entweder heller oder dunkler sein als das Objekt. Ansonsten stimmt die Polarität der Modellkanten nicht mit der Polarität der Kanten im Suchbild überein und das Formmodell kann dort nicht gefunden werden. Zu beachten ist, dass nur die Polarität der Kanten bestimmt wird, nicht ihr Kontrast. Desweiteren ist zu beachten, dass die Polarität nur aus einem einkanaligen Bild bestimmt wird. Wird ein Mehrkanalbild in Image übergeben, so wird nur der erste Kanal verwendet (und keine Fehlermeldung zurückgegeben).

Die Transformation, die durch die Matrix HomMat2D definiert ist, transformiert die Modellkonturen an die Stelle an der das Objekt im Bild Image erscheint. Sie kann dadurch bestimmt werden, dass man das Objekt z.B. mittels find_shape_model im Bild Image findet und anschließend die entsprechende Transformationsmatrix mit der Prozedur get_hom_mat2d_from_matching_result berechnet. Der Operator set_shape_model_metric ignoriert mit set_shape_model_origin gesetzte Ursprünge und nimmt stets (0,0) als Ursprung. Wird das Objekt im Bild Image gesucht, müssen auch alle anderen Operatoren den Ursprung im Punkt (0,0) setzen, weshalb set_shape_model_origin mit (0,0) als Ursprung aufgerufen werden soll.

Eine typische Vorgehensweise besteht darin, zuerst die XLD-Konturen aus einer Datei einzulesen. Da diese XLD-Konturen keine Polaritätsinformation haben, muss der Parameter Metric bei der Erzeugung des Formmodells aus den XLD-Konturen auf den Wert *'ignore_local_polarity'* gesetzt werden. Dann kann das Modell in einem ersten Suchbild gefunden werden. Aus den resultierenden Matchingergebnissen lassen sich die Transformationsparameter bestimmen, die die Modellkonturen an die Stelle des Suchbildes abbilden, an der das Objekt erscheint. Um die Matchingergebnisse interaktiv zu kontrollieren, können die Modellkonturen entsprechend transformiert werden. Ist das Matchingergebnis korrekt, kann der Wert des Parameters Metric z.B. zu *'use_polarity'* verändert werden. Dies führt in den nachfolgenden Bildern zu einer schnelleren und robusteren Erkennung.

Achtung

set_shape_model_metric kann nur mit Formmodellen genutzt werden, die aus XLD-Konturen erzeugt wurden.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild, das zur Bestimmung der Polarität verwendet wird.
- ▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
  Handle des Modells.
- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Transformationsmatrix.
- ▷ **Metric** (input_control) ..... string ⇝ *string*
  Art der zum Matchen verwendeten Metrik.
  Defaultwert: 'use_polarity'
  Werteliste: `Metric` ∈ {'use_polarity', 'ignore_global_polarity'}

Beispiel

```
*
* Read the contours from file
read_contour_xld_dxf (Contours, 'metal-part-01', [], [], DxfStatus)
*
* Scale the contours such that they correspond to the image
hom_mat2d_identity (HomMat2DIdentity)
hom_mat2d_scale (HomMat2DIdentity, 3.35, 3.35, 0, 0, HomMat2DScale)
affine_trans_contour_xld (Contours, ContoursAffineTrans, HomMat2DScale)
*
* Create the shape model with Metric='ignore_local_polarity'
create_shape_model_xld (ContoursAffineTrans, 'auto', rad(0), rad(360), \
                        'auto', 'auto', 'ignore_local_polarity', 5, ModelID)
*
* Get an image of the object and try to find the object in this image
read_image (Image, 'metal-parts/metal-parts-01')
find_shape_model (Image, ModelID, rad(0), rad(360), 0.5, 1, 0.5, \
                  'least_squares', 0, 0.9, Row, Column, Angle, Score)
*
* Visualize the match
vector_angle_to_rigid (0, 0, 0, Row, Column, Angle, HomMat2D)
get_shape_model_contours (ModelContours, ModelID, 1)
affine_trans_contour_xld (ModelContours, ModelContoursAffineTrans, HomMat2D)
*
* Set the matching metric to 'use_polarity', based on the position
* of the found object
set_shape_model_metric (Image, ModelID, HomMat2D, 'use_polarity')
*
* Use the shape model to detect the object faster and more robust
read_image (Image, 'metal-parts/metal-parts-02')
find_shape_model (Image, ModelID, rad(0), rad(360), 0.5, 0, 0.5, \
                  'least_squares', 0, 0.9, Row, Column, Angle, Score)
*
* Visualize the matches
for Index := 0 to |Score|-1 by 1
    vector_angle_to_rigid (0, 0, 0, Row[Index], Column[Index], Angle[Index], \
                           HomMat2D)
    affine_trans_contour_xld (ModelContours, ModelContoursAffineTrans, \
                              HomMat2D)
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_shape_model_metric` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_shape_model_xld`, `create_scaled_shape_model_xld`, `create_aniso_shape_model_xld`

Nachfolger

`find_shape_model`, `find_scaled_shape_model`, `find_aniso_shape_model`

Siehe auch

`create_shape_model_xld`, `create_scaled_shape_model_xld`, `create_aniso_shape_model_xld`

Modul

Matching

set_shape_model_origin (: : ModelID, Row, Column :)

Setzen des Ursprungs (Referenzpunktes) eines Formmodells.

Der Operator `set_shape_model_origin` setzt den Ursprung (Referenzpunkt) des Formmodells `ModelID` auf einen neuen Wert. Der Ursprung wird relativ zum Schwerpunkt der Region des Bildes, das zur Erzeugung des Formmodells mit `create_shape_model`, `create_scaled_shape_model` oder `create_aniso_shape_model` verwendet wurde, angegeben. Ein Ursprung von (0,0) heißt also, dass der Schwerpunkt der Region des Modellbildes verwendet wird. Ein Ursprung von (-20,-40) heißt, dass der Ursprung links oberhalb des Schwerpunktes liegt.

Parameter

▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
Handle des Modells.

▷ **Row** (input_control) ..... point.y ⇝ *real*
Zeilenkoordinate des Ursprungs des Formmodells.

▷ **Column** (input_control) ..... point.x ⇝ *real*
Spaltenkoordinate des Ursprungs des Formmodells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `set_shape_model_origin` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_shape_model, create_scaled_shape_model, create_aniso_shape_model, read_shape_model

Nachfolger

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models, get_shape_model_origin

Siehe auch

area_center

Modul

Matching

set_shape_model_param (: : ModelID, GenParamName, GenParamValue :)

Setzen ausgewählter Formmodellparameter.

Der Operator set_shape_model_param setzt ausgewählte Parameter im Formmodell ModelID. Die folgenden Parameter können modifiziert werden:

'min_contrast'

Setzt im Formmodell ModelID den minimalen Kontrast des Objekts in den Suchbildern. Damit wird der Wert *'min_contrast'*, der ursprünglich z.B. mit create_shape_model gesetzt wurde, überschrieben. Falls das Formmodell ModelID aus einer Datei eingelesen wurde und diese Datei geändert werden soll, muss das Formmodell ModelID nochmals in die Datei geschrieben werden, nachdem der Operator set_shape_model_param ausgeführt wurde.

'timeout'

Setzt die maximale Laufzeit der Operatoren die verwendet werden um das Formmodell ModelID zu finden (z.B. find_shape_model). Dies ist dann besonders nützlich, wenn eine maximale Zykluszeit sichergestellt werden muss. Der *'timeout'* muss in Millisekunden angegeben werden. Die zeitliche Genauigkeit des Timeouts hängt von vielen Faktoren ab, darunter ist die Größe des Modells, die Geschwindigkeit des Rechners und der *'timer_mode'* der über set_system gesetzt wurde. Beachten Sie, dass sich die Laufzeit der Modellsuche bei aktiviertem Timeout um bis zu 10 Prozent erhöht. Um den Timeout zu deaktivieren, können Sie entweder einen negativen Wert oder *'false'* angeben.

'border_shape_models'

Legt für die Operatoren, die verwendet werden um das Formmodell ModelID zu finden (z.B. find_shape_model), fest, ob die gesuchten Modelle teilweise außerhalb des Bildes liegen dürfen (also den Bildrand überschreiten dürfen). Der Wert von *'border_shape_models'* ist entweder *'true'*, *'false'* oder *'system'*. Wenn der Wert auf den Standardwert *'system'* gesetzt ist, wird der systemweite Wert verwendet, der zuletzt mit set_system für *'border_shape_models'* gesetzt wurde.

'use_clutter'

Deaktiviert oder aktiviert die Nutzung von Störparametern für das Formmodell ModelID, welche zuvor mit set_shape_model_clutter gesetzt wurden. Der Wert von *'use_clutter'* kann *'true'* oder *'false'* sein. Es ist zu beachten, dass der Wert von *'use_clutter'* die Anzahl erwarteter Einträge für den Parameter MinScore beeinflusst. Für weitere Informationen, siehe die Referenz der entsprechenden Operatoren, wie beispielsweise find_shape_model.

Für neu generierte Formmodelle ist die Nutzung von Störparametern deaktiviert. Nach dem Aufruf von set_shape_model_clutter ist die Nutzung von Störparametern aktiviert. Störparameter sowie der Wert von *'use_clutter'* können mit get_shape_model_clutter abgefragt werden.

'model_cache'

Schaltet einen internen Cache, der auf temporärem Speicher basiert und der von dem Formmodel bei z.B. find_shape_model verwendet wird, ein oder aus. Die Größe des Caches hängt davon ab, wie groß der Suchraum ist und wie viele Diskretisierungsschritte der Suchoperator benötigt. Das bedeutet, dass der Speicherverbrauch sowohl durch kleine Schrittweiten in AngleStep und ScaleStep als auch durch die Festlegung eines großen Suchraums mittels AngleExtent sowie ScaleMin und ScaleMax erhöht wird. Wenn man 'mo-

del_cache' ausschaltet (durch Setzen auf *'false'*), führt das manchmal zu einer leicht längerer Laufzeit, aber konstant kleinem Speicherverbrauch, was besonders für Anwendungen auf Embedded-Systemen wichtig ist. Standardmäßig ist der Cache eingeschaltet (*'true'*).

Parameter

▷ **ModelID** (input_control) . . . . . shape_model ⇝ *handle*
Handle des Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Parameternamen.
Werteliste: GenParamName ∈ {'min_contrast', 'timeout', 'border_shape_models', 'use_clutter', 'model_cache'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Parameterwerte.
Werteliste: GenParamValue ∈ {'true', 'false', 'system'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_shape_model_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_shape_model_xld, create_scaled_shape_model_xld, create_aniso_shape_model_xld

Nachfolger

find_shape_model, find_scaled_shape_model, find_aniso_shape_model

Siehe auch

create_shape_model_xld, create_scaled_shape_model_xld, create_aniso_shape_model_xld, set_shape_model_clutter, get_shape_model_clutter

Modul

Matching

train_generic_shape_model (Template : : ModelID :)

Trainieren eines Formmodells für das Matching.

Der Operator train_generic_shape_model trainiert das Formmodell ModelID um das in Template übergebene Trainingsmuster finden zu können.

Für Template können folgende Typen als Trainingsmuster übergeben werden:

Bild: Die Domäne des in Template übergebenen Bildes bestimmt die ROI des Modells.
Der Ursprung des Modells (Referenzpunkt) wird durch den Schwerpunkt der Bilddomäne definiert. Der Ursprung kann mittels set_shape_model_origin verändert werden.

XLD-Kontur: Die in Template übergebene XLD-Kontur sollte den Grauwertkanten des gesuchten Musters entsprechen.
Der Ursprung des Modells (Referenzpunkt) wird durch den Schwerpunkt des kleinsten achsenparallelen Rechtecks, das die Konturen umschließt, definiert. Der Ursprung kann mittels set_shape_model_origin verändert werden.

Eigenschaften des Modells ModelID können mit get_generic_shape_model_param abgefragt und mit set_generic_shape_model_param angepasst werden. Es ist zu beachten, dass die Veränderung bestimmter Parameter dazu führt, dass das Modell (erneut) trainiert werden muss (siehe set_generic_shape_model_param).

Parameter

▷ **Template** (input_object) ..... (multichannel-)object(-array) ⇝ *object*
Trainingsbild oder -XLD.

▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
Handle des Formmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert train_generic_shape_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_generic_shape_model, set_generic_shape_model_param

Nachfolger

find_generic_shape_model

Modul

Matching

write_shape_model (: : ModelID, FileName :)

Schreiben eines Formmodells auf Datei.

write_shape_model schreibt ein Formmodell in die Datei FileName. Das Modell kann mit read_shape_model wieder eingelesen werden. Die Dateiendung für das Formmodell ist in HALCON 'shm'.

Parameter

▷ **ModelID** (input_control) ..... shape_model ⇝ *handle*
Handle des Modells.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei.
Dateiendung: .shm

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert write_shape_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_shape_model`, `create_scaled_shape_model`, `create_aniso_shape_model`

Modul

Matching

17.4 Komponentenbasiert

`clear_component_model` (: : `ComponentModelID` :)

Freigabe des Speichers eines Komponentenmodells.

`clear_component_model` gibt den Speicher eines Komponentenmodells, das mit `create_component_model` oder `create_trained_component_model` angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `ComponentModelID` ist nach dem Aufruf ungültig.

Parameter

▷ **`ComponentModelID`** (input_control) . . . . . component_model ⇝ *handle*
Handle des Komponentenmodells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `clear_component_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ComponentModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_component_model`, `create_trained_component_model`, `read_component_model`, `write_component_model`

Modul

Matching

`clear_training_components` (: : `ComponentTrainingID` :)

Freigabe des Speichers eines Komponenten-Trainingsergebnisses.

`clear_training_components` gibt den mit `train_model_components` angelegten Speicher eines Trainingsergebnisses wieder frei. Das Trainingsergebnis kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `ComponentTrainingID` ist nach dem Aufruf ungültig.

Parameter

▷ **`ComponentTrainingID`** (input_control) . . . . . component_training ⇝ *handle*
Handle des Trainingsergebnisses.

Ergebnis

Ist der Handle des Trainingsergebnisses gültig, dann liefert `clear_training_components` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ComponentTrainingID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`train_model_components`, `write_training_components`

Modul

Matching

```
cluster_model_components (
    TrainingImages : ModelComponents : ComponentTrainingID,
    AmbiguityCriterion, MaxContourOverlap, ClusterThreshold : )
```

Übernahme neuer Parameter zur Bildung der Modellkomponenten in das Trainingsergebnis.

Mit `cluster_model_components` lassen sich Parameter ändern nachdem ein erstes Training mit `train_model_components` erfolgt ist. `cluster_model_components` ändert das Kriterium `AmbiguityCriterion` zur Lösung der Mehrdeutigkeiten, die maximale Konturüberlappung `MaxContourOverlap` und den Cluster-Schwellenwert `ClusterThreshold` des Trainingsergebnisses `ComponentTrainingID` auf die angegebenen Werte ab. Eine detaillierte Beschreibung dieser Parameter ist in der Dokumentation zu `train_model_components` nachzulesen. Durch das Ändern dieser Parameter ändert sich die Art, wie die initialen Komponenten zu starren Modellkomponenten zusammengefasst werden. Je größer z.B. der Cluster-Schwellenwert gewählt wird, desto weniger Zusammenschlüsse finden statt. Die für eine bestimmte Anwendung geeigneten Parameterwerte lassen sich durch wiederholten Aufruf von `inspect_clustered_components` interaktiv finden. Die ausgewählten Parameterwerte können dann mit `get_training_components` übernommen werden.

Die starren Modellkomponenten werden in `ModelComponents` zurückgeliefert. Um sinnvolle Ergebnisse zu erhalten, ist es wichtig in `TrainingImages` dieselben Trainingsbilder zu übergeben, die auch zum Trainieren mit `train_model_components` verwendet wurden. Die Lage der neu zusammengefassten Komponenten in den Trainingsbildern wird erneut mit Hilfe des formbasierten Matchings bestimmt. Dabei kann mit `set_system ('pregenerate_shape_models',...)` wie bei `train_model_components` entschieden werden, ob die entsprechenden Formmodelle vorab generiert werden sollen. Es kann sein, dass Modelle, die in einer Pyramidenebene den Bildrand berühren, nicht gefunden werden, auch wenn sie im ursprünglichen Bild vollständig enthalten sind. Mittels `set_system('border_shape_models',...)` kann festgelegt werden, ob die Modelle vollständig in den Trainingsbildern liegen müssen, oder ob sie auch teilweise über den Bildrand hinausragen können.

Parameter

- ▷ **`TrainingImages`** (input_object) ..... (multichannel-)image(-array) ⇝ *object* : byte / uint2
  Zum Trainieren der Modellkomponenten verwendete Trainingsbilder.
- ▷ **`ModelComponents`** (output_object) ..... region(-array) ⇝ *object*
  Konturregionen der starren Modellkomponenten.
- ▷ **`ComponentTrainingID`** (input_control) ..... component_training ⇝ *handle*
  Handle des Trainingsergebnisses.
- ▷ **`AmbiguityCriterion`** (input_control) ..... string ⇝ *string*
  Kriterium zur Lösung der Mehrdeutigkeiten.
  Defaultwert: 'rigidity'
  Werteliste: `AmbiguityCriterion` ∈ {'distance', 'orientation', 'distance_orientation', 'rigidity'}

▷ **MaxContourOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Konturüberlappung gefundener initialer Komponenten.
Defaultwert: 0.2
Wertevorschläge: MaxContourOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05
Restriktion: 0 <= MaxContourOverlap && MaxContourOverlap <= 1

▷ **ClusterThreshold** (input_control) . . . . . real ⇝ *real*
Schwellenwert für das Zusammenfassen initialer Komponenten.
Defaultwert: 0.5
Wertevorschläge: ClusterThreshold ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: 0 <= ClusterThreshold && ClusterThreshold <= 1

Beispiel

```
* Get the model image.
read_image (ModelImage, 'model_image.tif')
* Define the regions for the initial components.
gen_rectangle2 (InitialComponentRegions, 212, 233, 0.62, 167, 29)
gen_rectangle2 (Rectangle2, 298, 363, 1.17, 162, 34)
gen_rectangle2 (Rectangle3, 63, 444, -0.26, 50, 27)
gen_rectangle2 (Rectangle4, 120, 473, 0, 33, 20)
concat_obj (InitialComponentRegions, Rectangle2, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle3, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle4, InitialComponentRegions)
* Get the training images
gen_empty_obj (TrainingImages)
for i := 1 to 4 by 1
    read_image (TrainingImage, 'training_image-'+i$'02'+'.tif')
    concat_obj (TrainingImages, TrainingImage, TrainingImages)
endfor
* Extract the model components and train the relations.
train_model_components (ModelImage, InitialComponentRegions, \
                        TrainingImages, ModelComponents, 22, 60, 30, 0.65, \
                        0, 0, rad(60), 'speed', 'rigidity', 0.2, 0.5, \
                        ComponentTrainingID)
* Find the best value for the parameter ClusterThreshold.
inspect_clustered_components (ModelComponents, ComponentTrainingID, \
                              'rigidity', 0.2, 0.4)
* Adopt the ClusterThreshold into the training result.
cluster_model_components (TrainingImages, ModelComponents, \
                          ComponentTrainingID, 'rigidity', 0.2, 0.4)
* Create the component model based on the training result.
create_trained_component_model (ComponentTrainingID, -rad(30), rad(60), 10, \
                                0.5, 'auto', 'auto', 'none', \
                                'use_polarity', 'false', ComponentModelID, \
                                RootRanking)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert cluster_model_components den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_model_components, inspect_clustered_components

Nachfolger

get_training_components, create_trained_component_model, modify_component_relations, write_training_components, get_component_relations, clear_training_components

Modul

Matching

```
create_component_model ( ModelImage,
    ComponentRegions : : VariationRow, VariationColumn, VariationAngle,
    AngleStart, AngleExtent, ContrastLowComp, ContrastHighComp,
    MinSizeComp, MinContrastComp, MinScoreComp, NumLevelsComp,
    AngleStepComp, OptimizationComp, MetricComp,
    PregenerationComp : ComponentModelID, RootRanking )
```

Vorbereiten eines Komponentenmodells für das Matching auf der Basis explizit vorgegebener Komponenten und Relationen.

create_component_model bereitet Muster, die in Form eines Modellbildes ModelImage und mehrerer Regionen ComponentRegions übergeben werden, als Komponentenmodell für das Matching vor. Der Ausgabeparameter ComponentModelID ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von find_component_model verwendet wird. Im Gegensatz zu create_trained_component_model ist ein mit train_model_components durchgeführtes Training zum Aufruf von create_component_model nicht erforderlich.

Jede der in ComponentRegions übergebenen Regionen beschreibt eine separate Modellkomponente. In nachfolgenden Operatoren wird der Index einer Komponenten in ComponentRegions verwendet, um die Modellkomponente zu bezeichnen. Der Referenzpunkt einer Modellkomponente ist festgelegt durch den Schwerpunkt der in ComponentRegions übergebenen Region und kann mit area_center ermittelt werden.

Die relativen Bewegungen (Relationen) zwischen den einzelnen Modellkomponenten können mit Hilfe der Parameter VariationRow, VariationColumn und VariationAngle festgelegt werden. Da eine direkte Angabe der Relationen sehr aufwendig ist, werden anstatt der Relationen die Variationen der Modellkomponenten übergeben. Die Variationen beschreiben die Bewegungen der Komponenten unabhängig voneinander relativ zu deren Lagen im Modellbild ModelImage. Dabei beschreiben die Parameter VariationRow und VariationColumn die Bewegung der Komponenten in Zeilen- und Spaltenrichtung um $\pm\frac{1}{2}$VariationRow und $\pm\frac{1}{2}$VariationColumn. Der Parameter VariationAngle beschreibt die Winkelbewegung der Komponente um $\pm\frac{1}{2}$VariationAngle. Aus diesen Angaben werden die Relationen zwischen den Komponenten automatisch berechnet. Die drei Parameter müssen entweder genau ein Element enthalten oder dieselbe Anzahl von Elementen wie ComponentRegions. Im ersten Fall wird der Wert des Eingabeparameters für alle Modellkomponenten gleich verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für die entsprechende Modellkomponente in ComponentRegions verwendet.

Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Komponentenmodells im Bild fest.

Im Wesentlichen wird intern für jede Modellkomponente ein separates Formmodell erzeugt (siehe create_shape_model). Daher entsprechen die Parameter ContrastLowComp, ContrastHighComp, MinSizeComp, MinContrastComp, MinScoreComp, NumLevelsComp, AngleStepComp, OptimizationComp, MetricComp und PregenerationComp den Parametern in create_shape_model mit folgenden Unterschieden: Im Parameter Contrast von create_shape_model kann sowohl der obere, als auch der untere Schwellenwert für das Hysterese-Schwellenwert-Verfahren übergeben werden. Zusätzlich kann ein dritter Wert angegeben werden, der die Unterdrückung kleiner zusammenhängender Konturregionen ermöglicht. Der Operator create_component_model bietet dagegen zum Einstellen dieser drei Werte drei separate Parameter ContrastHighComp, ContrastLowComp und MinScoreComp. Daraus ergibt sich für die automatische Bestimmung der Kontrastwerte folgender Unterschied: Sollen beide Hysterese-Schwellen automatisch bestimmt werden, so müssen sowohl ContrastLowComp als auch ContrastHighComp jeweils auf *'auto'* gesetzt werden. Soll dagegen lediglich ein Kontrastwert automatisch bestimmt werden, so muss ContrastLowComp auf

'auto' und ContrastHighComp auf einen beliebigen Wert ungleich *'auto'* gesetzt werden. Im Parameter Optimization von create_shape_model kann neben einer möglichen Reduktion der Modellpunkte auch entschieden werden, ob das Formmodell vollständig vorab generiert wird oder nicht. Der Operator create_trained_component_model bietet dagegen für die Vorabgenerierung der Formmodelle einen separaten Parameter PregenerationComp. Ein dritter Unterschied bezieht sich auf den Parameter MinScoreComp, der beim formbasierten Matching nicht bereits beim Vorbereiten des Modells mit create_shape_model sondern erst bei der Suche mit find_shape_model übergeben wird. Beim Vorbereiten des Komponentenmodells ist es jedoch vorteilhaft, auftretende Rotationssymmetrien der Modellkomponenten und Ähnlichkeiten zwischen den Modellkomponenten zu analysieren und beim Aufbau des Komponentenmodells zu berücksichtigen. Diese Analyse kann jedoch nur dann zu sinnvollen Ergebnissen führen, wenn der zur Suche verwendete Wert für MinScoreComp (siehe find_component_model) bereits annähernd bekannt ist.

Neben den Parametern ContrastLowComp, ContrastHighComp und MinSizeComp können auch die Parameter MinContrastComp, NumLevelsComp, AngleStepComp und OptimizationComp jeweils durch die Angabe von *'auto'* automatisch ermittelt werden.

Alle komponentenspezifischen Eingabeparameter (Parameternamen enden jeweils auf Comp) müssen entweder genau ein Element enthalten oder dieselbe Anzahl von Elementen wie ComponentRegions. Im ersten Fall wird der Wert des Eingabeparameters für alle Modellkomponenten gleich verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für die entsprechende Modellkomponente in ComponentRegions verwendet.

Das Komponentenmodell enthält neben den einzelnen Formmodellen auch die Information über die Art und Weise, wie die einzelnen Modellkomponenten mit find_component_model relativ zueinander gesucht werden müssen, damit die Rechenzeit während der Suche minimal ist. Dabei werden die Komponenten durch eine Baumstruktur repräsentiert. Zunächst wird die Komponente, die an der Wurzel des Baumes (Wurzelkomponente) steht, gesucht. Anschließend werden die übrigen Komponenten jeweils relativ zur Lage ihres Vorgängers im Baum gesucht.

Die Wurzelkomponente kann bei der Suche mit find_component_model als Parameter übergeben werden. Inwiefern sich eine Modellkomponente als Wurzelkomponente eignet, hängt von verschiedenen Faktoren ab. Prinzipiell sollte eine Modellkomponente gewählt werden, die mit hoher Wahrscheinlichkeit im Bild wiedergefunden werden kann. Eine im Bild stark verdeckte oder fehlende Komponente eignet sind daher nur bedingt als Wurzelkomponente. Auch die mit der Wurzelkomponente verbundene Rechenzeit während der Suche kann als Auswahlkriterium herangezogen werden. Eine nach diesem Kriterium sortierte Rangfolge der Modellkomponenten wird im Parameter RootRanking zurückgeliefert. In diesem Parameter sind die Indizes der Modellkomponenten nach der mit ihnen verbundenen Suchzeit aufsteigend sortiert, d.h. RootRanking[0] enthält den Index der Modellkomponente, die - als Wurzelkomponente gewählt - die schnellste Suche ermöglicht. Die in RootRanking ermittelte Reihenfolge stellt jedoch nur eine grobe Abschätzung dar. Die Abschätzung setzt außerdem voraus, dass sowohl die Bildgröße als auch der Wert des Systemparameters *'border_shape_models'* beim Aufruf von create_component_model und find_component_model identisch sind.

Parameter

- ▷ **ModelImage** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild, aus dem die Formmodelle für die Modellkomponenten generiert werden sollen.
- ▷ **ComponentRegions** (input_object) . . . . . region-array ⇝ *object*
  Eingaberegionen, aus denen die Formmodelle für die Modellkomponenten generiert werden sollen.
- ▷ **VariationRow** (input_control) . . . . . integer(-array) ⇝ *integer*
  Variation der Modellkomponenten in Zeilenrichtung.
  Wertevorschläge: VariationRow ∈ {0, 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 100, 150}
  Restriktion: VariationRow >= 0
- ▷ **VariationColumn** (input_control) . . . . . integer(-array) ⇝ *integer*
  Variation der Modellkomponenten in Spaltenrichtung.
  Wertevorschläge: VariationColumn ∈ {0, 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 100, 150}
  Restriktion: VariationColumn >= 0
- ▷ **VariationAngle** (input_control) . . . . . angle.rad(-array) ⇝ *real*
  Winkel-Variation der Modellkomponenten.
  Wertevorschläge: VariationAngle ∈ {0.0, 0.017, 0.035, 0.05, 0.07, 0.09, 0.17, 0.35, 0.52, 0.67, 0.87}
  Restriktion: VariationAngle >= 0

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Komponentenmodells.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs des Komponentenmodells.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **ContrastLowComp** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Unterer Hystereseschwellenwert für den Kontrast der Komponenten im Modellbild.
Defaultwert: 'auto'
Wertevorschläge: `ContrastLowComp` ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}
Restriktion: ContrastLowComp > 0

▷ **ContrastHighComp** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Oberer Hystereseschwellenwert für den Kontrast der Komponenten im Modellbild.
Defaultwert: 'auto'
Wertevorschläge: `ContrastHighComp` ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}
Restriktion: ContrastHighComp > 0 && ContrastHighComp >= ContrastLowComp

▷ **MinSizeComp** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Mindestgröße der Konturregionen im Modell.
Defaultwert: 'auto'
Wertevorschläge: `MinSizeComp` ∈ {'auto', 0, 5, 10, 20, 30, 40}
Restriktion: MinSizeComp >= 0

▷ **MinContrastComp** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Minimaler Kontrast der Komponenten in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: `MinContrastComp` ∈ {'auto', 10, 20, 20, 40}
Restriktion: MinContrastComp <= ContrastLowComp && MinContrastComp >= 0

▷ **MinScoreComp** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen der Komponenten.
Defaultwert: 0.5
Wertevorschläge: `MinScoreComp` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05
Restriktion: 0 <= MinScoreComp && MinScoreComp <= 1

▷ **NumLevelsComp** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen für die Komponenten.
Defaultwert: 'auto'
Werteliste: `NumLevelsComp` ∈ {'auto', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStepComp** (input_control) . . . . . angle.rad(-array) ⇝ *real* / string
Schrittweite der Winkel (Auflösung) für die Komponenten.
Defaultwert: 'auto'
Wertevorschläge: `AngleStepComp` ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStepComp >= 0

▷ **OptimizationComp** (input_control) . . . . . string ⇝ *string*
Art der Optimierung für die Komponenten.
Defaultwert: 'auto'
Werteliste: `OptimizationComp` ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **MetricComp** (input_control) . . . . . string(-array) ⇝ *string*
Art der zum Matchen der Komponenten verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: `MetricComp` ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **PregenerationComp** (input_control) . . . . . string(-array) ⇝ *string*
Komplette Vorabgenerierung der Formmodelle für die Komponenten falls *'true'*.
Defaultwert: 'false'
Werteliste: `PregenerationComp` ∈ {'true', 'false'}

▷ **ComponentModelID** (output_control) . . . . . component_model ⇝ *handle*
Handle des Komponentenmodells.

▷ **RootRanking** (output_control) . . . . . integer(-array) ⇝ *integer*
Reihenfolge der Modellkomponenten, die deren Eignung als Wurzelkomponente zu fungieren ausdrückt.

Beispiel

```
* Read the model image.
read_image (ModelImage, 'model_image.tif')
* Describe the model components.
gen_rectangle2 (ComponentRegions, 318, 109, -1.62, 34, 19)
gen_rectangle2 (Rectangle2, 342, 238, -1.63, 32, 17)
gen_rectangle2 (Rectangle3, 355, 505, 1.41, 25, 17)
concat_obj (ComponentRegions, Rectangle2, ComponentRegions)
concat_obj (ComponentRegions, Rectangle3, ComponentRegions)
* Create the component model by explicitly specifying the relations.
create_component_model (ModelImage, ComponentRegions, 20, 20, rad(25), 0, \
                        rad(360), 15, 40, 15, 10, 0.8, 'auto', 'auto', \
                        'none', 'use_polarity', 'false', ComponentModelID, \
                        RootRanking)
* Find the component model in a run-time image.
read_image (SearchImage, 'search_image.tif')
find_component_model (SearchImage, ComponentModelID, RootRanking, 0, \
                      rad(360), 0.5, 0, 0.5, 'stop_search', \
                      'search_from_best', 'none', 0.8, 'least_squares', 0, \
                      0.8, ModelStart, ModelEnd, Score, RowComp, \
                      ColumnComp, AngleComp, ScoreComp, ModelComp)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_component_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`draw_region`, `concat_obj`

Nachfolger

`find_component_model`

Alternativen

`create_trained_component_model`

Siehe auch

`create_shape_model`, `find_shape_model`

Modul

Matching

create_trained_component_model (: : ComponentTrainingID, AngleStart, AngleExtent, MinContrastComp, MinScoreComp, NumLevelsComp, AngleStepComp, OptimizationComp, MetricComp, PregenerationComp : ComponentModelID, RootRanking)

Vorbereiten eines Komponentenmodells für das Matching auf der Basis trainierter Komponenten.

create_trained_component_model bereitet das Trainingsergebnis, das in ComponentTrainingID übergeben wird, als Komponentenmodell für das Matching vor. Der Ausgabeparameter ComponentModelID ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von find_component_model verwendet wird. Im Gegensatz zu create_component_model müssen die Modellkomponenten mit train_model_components trainiert worden sein bevor create_trained_component_model aufgerufen werden kann.

Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Komponentenmodells im Bild fest.

Im Wesentlichen wird intern für jede Modellkomponente ein separates Formmodell erzeugt (siehe create_shape_model). Daher entsprechen die Parameter MinContrastComp, MinScoreComp, NumLevelsComp, AngleStepComp, OptimizationComp, MetricComp und PregenerationComp den Parametern in create_shape_model mit den folgenden Unterschieden: Im Parameter Optimization von create_shape_model kann neben einer möglichen Reduktion der Modellpunkte auch entschieden werden, ob das Formmodell vollständig vorab generiert wird oder nicht. Der Operator create_trained_component_model bietet dagegen für die Vorabgenerierung der Formmodelle einen separaten Parameter PregenerationComp. Ein zweiter Unterschied bezieht sich auf den Parameter MinScoreComp, der beim formbasierten Matching nicht bereits beim Vorbereiten des Modells mit create_shape_model sondern erst bei der Suche mit find_shape_model übergeben wird. Beim Vorbereiten des Komponentenmodells ist es jedoch vorteilhaft, auftretende Rotationssymmetrien der Modellkomponenten und Ähnlichkeiten zwischen den Modellkomponenten zu analysieren und beim Aufbau des Komponentenmodells zu berücksichtigen. Diese Analyse kann jedoch nur dann zu sinnvollen Ergebnissen führen, wenn der zur Suche verwendete Wert für MinScoreComp (siehe find_component_model) bereits annähernd bekannt ist. Die von find_component_model ermittelten Lagekoordinaten der Modellkomponenten in einem Suchbild beziehen sich auf deren Referenzpunkt. Der Referenzpunkt einer Modellkomponente ist festgelegt durch den Schwerpunkt der entsprechenden von train_model_components in ModelComponents zurückgelieferten Region.

Die Parameter MinContrastComp, NumLevelsComp, AngleStepComp und OptimizationComp können jeweils durch die Angabe von *'auto'* automatisch ermittelt werden.

Alle komponentenspezifischen Eingabeparameter (Parameternamen enden jeweils auf Comp) müssen entweder genau ein Element enthalten oder dieselbe Anzahl von Elementen wie in ComponentTrainingID enthaltene Modellkomponenten. Im ersten Fall wird der Wert des Eingabeparameters für alle Modellkomponenten gleich verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für die entsprechende Modellkomponente in ComponentTrainingID verwendet.

Das Komponentenmodell enthält neben den einzelnen Formmodellen auch die Information über die Art und Weise, wie die einzelnen Modellkomponenten mit find_component_model relativ zueinander gesucht werden müssen, damit die Rechenzeit während der Suche minimal ist. Dabei werden die Komponenten durch eine Baumstruktur repräsentiert. Zunächst wird die Komponente, die an der Wurzel des Baumes (Wurzelkomponente) steht, gesucht. Anschließend werden die übrigen Komponenten jeweils relativ zur Lage ihres Vorgängers im Baum gesucht.

Die Wurzelkomponente kann bei der Suche mit find_component_model als Parameter übergeben werden. Inwiefern sich eine Modellkomponente als Wurzelkomponente eignet, hängt von verschiedenen Faktoren ab. Prinzipiell sollte eine Modellkomponente gewählt werden, die mit hoher Wahrscheinlichkeit im Bild wiedergefunden werden kann. Eine im Bild stark verdeckte oder fehlende Komponente eignet sind daher nur bedingt als Wurzelkomponente. Auch die mit der Wurzelkomponente verbundene Rechenzeit während der Suche kann als Auswahlkriterium herangezogen werden. Eine nach diesem Kriterium sortierte Rangfolge der Modellkomponenten wird im Parameter RootRanking zurückgeliefert. In diesem Parameter sind die Indizes der Modellkomponenten nach der mit ihnen verbundenen Rechenzeit aufsteigend sortiert, d.h. RootRanking[0] enthält den Index der Modellkomponente, die - als Wurzelkomponente gewählt - die schnellste Suche ermöglicht. Die in RootRanking ermittelte Reihenfolge stellt jedoch nur eine grobe Abschätzung dar. Die Abschätzung setzt außerdem voraus, dass sowohl die Bildgröße als auch der Wert des Systemparameters *'border_shape_models'* beim Aufruf von create_trained_component_model und find_component_model identisch sind.

Parameter

▷ **ComponentTrainingID** (input_control) . . . . . component_training ⇝ *handle*
Handle des Trainingsergebnisses.

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Komponentenmodells.
Defaultwert: -0.39
Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: `AngleExtent` ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **MinContrastComp** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Minimaler Kontrast der Komponenten in den Suchbildern.
Defaultwert: 'auto'
Wertevorschläge: `MinContrastComp` ∈ {'auto', 10, 20, 20, 40}
Restriktion: MinContrastComp >= 0

▷ **MinScoreComp** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen der Komponenten.
Defaultwert: 0.5
Wertevorschläge: `MinScoreComp` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05
Restriktion: 0 <= MinScoreComp && MinScoreComp <= 1

▷ **NumLevelsComp** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen für die Komponenten.
Defaultwert: 'auto'
Werteliste: `NumLevelsComp` ∈ {'auto', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStepComp** (input_control) . . . . . angle.rad(-array) ⇝ *real* / string
Schrittweite der Winkel (Auflösung) für die Komponenten.
Defaultwert: 'auto'
Wertevorschläge: `AngleStepComp` ∈ {'auto', 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStepComp >= 0

▷ **OptimizationComp** (input_control) . . . . . string ⇝ *string*
Art der Optimierung für die Komponenten.
Defaultwert: 'auto'
Werteliste: `OptimizationComp` ∈ {'auto', 'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

▷ **MetricComp** (input_control) . . . . . string(-array) ⇝ *string*
Art der zum Matchen der Komponenten verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: `MetricComp` ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity', 'ignore_color_polarity'}

▷ **PregenerationComp** (input_control) . . . . . string(-array) ⇝ *string*
Komplette Vorabgenerierung der Formmodelle für die Komponenten falls *'true'*.
Defaultwert: 'false'
Werteliste: `PregenerationComp` ∈ {'true', 'false'}

▷ **ComponentModelID** (output_control) . . . . . component_model ⇝ *handle*
Handle des Komponentenmodells.

▷ **RootRanking** (output_control) . . . . . integer(-array) ⇝ *integer*
Reihenfolge der Modellkomponenten, die deren Eignung als Wurzelkomponente zu fungieren ausdrückt.

Beispiel

```
* Get the model image.
read_image (ModelImage, 'model_image.tif')
* Define the regions for the initial components.
```

```
gen_rectangle2 (InitialComponentRegions, 212, 233, 0.62, 167, 29)
gen_rectangle2 (Rectangle2, 298, 363, 1.17, 162, 34)
gen_rectangle2 (Rectangle3, 63, 444, -0.26, 50, 27)
gen_rectangle2 (Rectangle4, 120, 473, 0, 33, 20)
concat_obj (InitialComponentRegions, Rectangle2, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle3, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle4, InitialComponentRegions)
* Get the training images.
gen_empty_obj (TrainingImages)
for i := 1 to 4 by 1
    read_image (TrainingImage, 'training_image-'+i+'.tif')
    concat_obj (TrainingImages, TrainingImage, TrainingImages)
endfor
* Extract the model components and train the relations.
train_model_components (ModelImage, InitialComponentRegions, \
                        TrainingImages, ModelComponents, 22, 60, 30, 0.65, \
                        0, 0, rad(60), 'speed', 'rigidity', 0.2, 0.4, \
                        ComponentTrainingID)
* Create the component model based on the training result.
create_trained_component_model (ComponentTrainingID, -rad(30), rad(60), 10, \
                                0.5, 'auto', 'auto', 'none', \
                                'use_polarity', 'false', ComponentModelID, \
                                RootRanking)
* Find the component model in a run-time image.
read_image (SearchImage, 'search_image.tif')
find_component_model (SearchImage, ComponentModelID, RootRanking, -rad(30), \
                      rad(60), 0.5, 0, 0.5, 'stop_search', 'prune_branch', \
                      'none', 0.55, 'least_squares', 0, 0.9, ModelStart, \
                      ModelEnd, Score, RowComp, ColumnComp, AngleComp, \
                      ScoreComp, ModelComp)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_trained_component_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`train_model_components`, `read_training_components`

Nachfolger

`find_component_model`

Alternativen

`create_component_model`

Siehe auch

`create_shape_model`, `find_shape_model`

Modul

Matching

deserialize_component_model (
: : SerializedItemHandle : ComponentModelID)

Deserialisieren eines serialisierten Komponentenmodells.

deserialize_component_model deserialisiert ein Komponentenmodell, welches mit serialize_component_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Komponentenmodell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Komponentenmodell mit dem Handle ComponentModelID gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **ComponentModelID** (output_control) ..... component_model ⇝ *handle*
Handle des Komponentenmodells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_component_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_component_model

Nachfolger

find_component_model

Modul

Matching

deserialize_training_components (
: : SerializedItemHandle : ComponentTrainingID)

Deserialisieren eines Komponenten-Trainingsergebnisses.

deserialize_training_components deserialisiert ein Komponenten-Trainingsergebnis, welches mit serialize_training_components serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte Komponenten-Trainingsergebnis wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten Komponenten-Trainingsergebnis mit dem Handle ComponentTrainingID gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **ComponentTrainingID** (output_control) ..... component_training ⇝ *handle*
Handle des Trainingsergebnisses.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_training_components den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_training_components`

Nachfolger

`create_trained_component_model`

Siehe auch

`train_model_components`, `clear_training_components`

Modul

Matching

```
find_component_model ( Image : : ComponentModelID, RootComponent,
    AngleStartRoot, AngleExtentRoot, MinScore, NumMatches, MaxOverlap,
    IfRootNotFound, IfComponentNotFound, PosePrediction, MinScoreComp,
    SubPixelComp, NumLevelsComp, GreedinessComp : ModelStart, ModelEnd,
    Score, RowComp, ColumnComp, AngleComp, ScoreComp, ModelComp )
```

Suche der besten Matches eines Komponentenmodells in einem Bild.

`find_component_model` findet die besten `NumMatches` Instanzen des Komponentenmodells `ComponentModelID` im Eingabebild `Image`.

Das Ergebnis der Suche kann mit `get_found_component_model` visualisiert werden. Ebenso können damit die Matches der Komponenten einer bestimmten Instanz extrahiert werden.

Weitere Informationen zu Eingabeparametern

`ComponentModelID`: Handle des Komponentenmodells.

Das Modell muss zuvor mit `create_trained_component_model` oder `create_component_model` erzeugt oder mit `read_component_model` eingelesen worden sein.

`RootComponent`: Index der Wurzelkomponente.

Die Komponenten des Komponentenmodells `ComponentModelID` sind durch eine Baumstruktur repräsentiert. Die Wurzelkomponente steht an der Wurzel des Baumes.

Die Wurzelkomponente wird während der Suche im gesamten Suchraum gesucht, d.h. innerhalb aller erlaubten Positionen und Orientierungen. Die übrigen Komponenten werden dagegen jeweils relativ zur Lage ihres Vorgängers im Baum in einem eingeschränkten Suchraum gesucht (rekursive Suche). Die eingeschränkten Suchräume werden dabei aus den entsprechenden Relationen berechnet.

Inwiefern sich eine Modellkomponente als Wurzelkomponente eignet, hängt von verschiedenen Faktoren ab. Prinzipiell sollte eine Modellkomponente gewählt werden, die mit hoher Wahrscheinlichkeit im Bild wiedergefunden werden kann. Eine im Bild stark verdeckte oder fehlende Komponente eignet sind daher nur bedingt als Wurzelkomponente. Für Verhalten des Operators in solchen Fällen, siehe den Eintrag zu `IfRootNotFound` weiter unten.

Ein anderes mögliches Auswahlkriterium ist die mit der Wurzelkomponente verbundene Rechenzeit während der Suche. Eine entsprechende Rangfolge der Modellkomponenten wird über den Parameter `RootRanking` des Operators `create_trained_component_model` bzw. `create_component_model` zurückgegeben. Wird die komplette Rangfolge `RootRanking` in `RootComponent` übergeben, dann wird der erste Wert `RootComponent`[0] als Wurzelkomponente ausgewählt.

`AngleStartRoot` und `AngleExtentRoot`: Legen den erlaubten Winkelbereich fest (in [rad]), in dem nach der Wurzelkomponente gesucht wird.

Der Winkelbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Komponentenmodells mit `create_trained_component_model` bzw. `create_component_model` festgelegt worden ist. Der Winkelbereich kann für die einzelnen Formmodelle mit `get_shape_model_params` abgefragt werden. Das dazu notwendige Handle des entsprechenden Formmodells kann mit `get_component_model_params` erhalten werden.

`MinScore`: Minimale Bewertung, die ein potentieller Match des Komoponentenmodells erhalten muss um als Instanz zu zählen.

Falls erwartet werden kann, dass das Komponentenmodell niemals verdeckt wird, kann MinScore auf so hohe Werte wie *0.8* oder sogar *0.9* gesetzt werden. Der Wert dieses Parameters nur geringen Einfluss auf die Rechenzeit. Ausnahme ist im Fall IfRootNotFound auf *'select_new_root'*, siehe unten.

NumMatches: Bestimmt die maximale Anzahl zurückgegebener Instanzen.

Werden weniger Instanzen als NumMatches gefunden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter MinScore hat höhere Priorität.

Werden mehr Instanzen als NumMatches mit einer Bewertung besser MinScore gefunden, werden nur die besten NumMatches Instanzen zurückgeliefert. Sollen jedoch alle Modellinstanzen, deren Bewertung MinScore übersteigt, im Bild gefunden werden, muss NumMatches auf *0* gesetzt werden.

MaxOverlap: Bestimmt die maximal erlaubte Überlappung zweier Instanzen, wobei dieser Anteil als Zahl zwischen 0 und 1 ausgedrückt wird.

In manchen Fällen kann es vorkommen, dass sich verschiedene Instanzen nur durch die Lage einer oder weniger Komponenten unterscheiden. Überlappen sich zwei Instanzen um mehr als MaxOverlap, wird nur die bessere zurückgeliefert. Dies bedeutet, bei MaxOverlap = *0* werden nur Instanzen zurückgeliefert die sich überhaupt nicht überlappen, während bei MaxOverlap = *1* keine Überprüfung der Überlappung stattfindet und alle gefundenen Instanzen zurückgeliefert werden. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen der einzelnen Komponenten (siehe smallest_rectangle2).

IfRootNotFound: Steuert das Verhalten des Operators bei einem fehlenden oder stark verdeckten Wurzelobjekt.

Mögliche Werte:

- *'stop_search'*: Es wird davon ausgegangen, dass die Wurzelkomponente im Bild immer gefunden wird. Folglich wird für eine Instanz die Suche nach den restlichen Komponenten nicht fortgesetzt, wenn die Wurzelkomponente nicht gefunden wurde.
- *'select_new_root'*: Unterschiedliche Komponenten werden sukzessive als Wurzelkomponente gewählt um den gesamten Suchraum zu durchsuchen. Die Auswahl der Wurzelkomponenten erfolgt in der Reihenfolge, die in RootRanking übergeben wurde. Die Lagen der gefundenen Instanzen aller Wurzelkomponenten werden dann dazu genutzt die rekursiven Suchen nach den übrigen Komponenten zu starten. Dadurch ist es zwar möglich, Instanzen auch dann zu finden, wenn die ursprüngliche Wurzelkomponente nicht gefunden wird, die Rechenzeit der Suche steigt allerdings im Vergleich zur Suche mit *'stop_search'* deutlich an. Dies ist insbesondere bei kleinem MinScore der Fall, da dann mehr Wurzelkomponenten gesucht werden müssen. Sollten bei der Suche mehr Wurzelkomponenten benötigt werden, als in RootComponent übergeben wurden, so werden die Wurzelkomponenten in der automatisch berechneten Reihenfolge ergänzt (siehe create_trained_component_model bzw. create_component_model).

IfComponentNotFound: Bestimmt inwiefern Komponenten gesucht werden, wenn die Vorgänger-Komponente selbst nicht gefunden wurde (z.B. weil sie fehlt oder stark verdeckt ist).

Mögliche Werte:

- *'prune_branch'*: Solche Komponenten werden nicht gesucht und ebenfalls als 'nicht gefunden' eingestuft.
- *'search_from_upper'*: Solche Komponenten werden relativ zur Lage der Vorgänger-Komponente der Vorgänger-Komponente gesucht.
- *'search_from_best'*: Solche Komponenten werden relativ zur Lage der bereits gefundenen Komponente gesucht, von der aus die relative Suche mit dem geringsten Rechenaufwand durchgeführt werden kann.

PosePrediction: Gibt an, ob die Lage von nicht gefundenen Komponenten nach der Suche geschätzt werden soll.

Mögliche Werte:

- *'none'*: Nur für die gefundenen Komponenten werden die Lagen zurückgeliefert.
- *'from_neighbors'*: Die Lagen der nicht gefundenen Komponenten geschätzt und mit einer Bewertung von ScoreComp = *0.0* zurückgeliefert. Die Schätzung der Lage erfolgt auf Grundlage der Lagen gefundener Nachbarkomponenten im Suchbaum.
- *'from_all'*: Die Lagen der nicht gefundenen Komponenten geschätzt und mit einer Bewertung von ScoreComp = *0.0* zurückgeliefert. Die Schätzung der Lage erfolgt auf Grundlage der Lagen aller gefundenen Komponenten

MinScoreComp: Minimale notwendige Bewertung der Komponenten einer zu findenden Instanz.

Dieser Parameter verhält sich analog zu MinScore in find_shape_model.

Setzbar sind genau ein Wert oder dieselbe Anzahl Werte wie Modellkomponenten in ComponentModelID. Im ersten Fall wird der Wert des Eingabeparameters für alle Modellkomponenten verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für die entsprechende Komponente in ComponentModelID verwendet.

SubPixelComp: Bestimmt, ob die Extraktion subpixelgenau erfolgen soll und in gegebenem Fall die maximal zulässige Objektdeformation in Pixeln.

Dieser Parameter verhält sich analog zu SubPixel in find_shape_model. Somit ist die maximale Objektdeformation im gleichen String als Ganzzahl anzugeben.

Setzbar sind genau ein Wert oder dieselbe Anzahl Werte wie Modellkomponenten in ComponentModelID. Im ersten Fall wird der Wert des Eingabeparameters für alle Modellkomponenten verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für die entsprechende Komponente in ComponentModelID verwendet.

Beispiel: ['least_squares', 'max_deformation 2'].

NumLevelsComp: Bestimmt die verwendeten Pyramidenebenen für die Modellkomponenten.

Dieser Parameter verhält sich analog zu NumLevels in find_shape_model.

Er bestimmt die Anzahl der verwendeten Pyramidenebenen, die während der Suche benutzt werden. Setzbar sind genau ein Wert oder dieselbe Anzahl Werte wie Modellkomponenten in ComponentModelID. Im ersten Fall wird der Wert des Eingabeparameters für alle Modellkomponenten verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für die entsprechende Komponente in ComponentModelID verwendet.

Optional kann man für diesen Parameter auch Wertepaare setzen: Dann spezifiziert der erste Wert weiterhin die Anzahl an zu verwendende Pyramidenebenen. Der zweite Wert spezifiziert die unterste Pyramidenebene, bis auf welche die Matches verfolgt werden. Dabei kann man ein einzelnes Wertepaar für alle Komponenten angeben oder für jede Komponente in ComponentModelID ein eigenes. Sollen für unterschiedliche Komponenten unterschiedliche Wertepaare berücksichtigt werden, so sind diese im selben Tupel anzugeben.

Enthält ComponentModelID genau zwei Komponenten und NumLevelsComp zwei gesetzte Werte, so werden diese Werte als jeweilige Anzahl zu verwendende Pyramidenebenen aufgefasst und nicht als Wertepaar.

Beispiel: In ComponentModelID sind zwei Komponenten enthalten, für welche unterschiedliche Pyramidenebenen verwendet werden sollen. Für die erste Komponente sollen 5 Pyramidenebenen verwendet werden, mit Ebene 2 als der untersten. Für die zweite Komponente sollen 4 Pyramidenebenen verwendet werden, mit Ebene 1 als der untersten. Dann ist NumLevelsComp = *[5,2,4,1]*.

GreedinessComp: „Gierigkeit“ der Suchheuristik für die Komponenten: Wert von 0 bis 1. Dabei bedeutet 0: sicher aber langsam, 1: schnell aber Matches können „übersehen“ werden.

Dieser Parameter verhält sich analog zu Greediness in find_shape_model.

Setzbar sind genau ein Wert oder dieselbe Anzahl Werte wie Modellkomponenten in ComponentModelID. Im ersten Fall wird der Wert des Eingabeparameters für alle Modellkomponenten verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für die entsprechende Komponente in ComponentModelID verwendet.

Weitere Informationen zu Ausgabeparametern

ModelStart und ModelEnd: Geben den ersten, respektive letzten Index und damit den Indexbereich all jener Komponentenmatches die zur selben gefundenen Instanz des Komponentenmodells gehören

Die Komponentenmatches, die zur ersten gefundenen Instanz des Komponentenmodells gehören, werden somit durch das Intervall an Indizes [ModelStart[0],ModelEnd[0]] beschrieben. Dabei beziehen sich die Indizes auf die Werte der Parameter RowComp, ColumnComp, AngleComp, ScoreComp und ModelComp.

Beispiel: Das Komponentenmodell besteht aus drei Komponenten. Auf dem Bild werden zwei Instanzen gefunden, wobei für die zweite Instanz lediglich zwei Komponenten (Komponente 0 und Komponente 2) gefunden werden konnten. Dann könnten die zurückgegebenen Parameter wie folgt aussehen:

RowComp = *[100,200,300,150,250]*	ModelStart = *[0,3]*
ColumnComp = *[200,210,220,400,425]*	ModelEnd = *[2,4]*
AngleComp = *[0,0.1,-0.2,0.1,0.2]*	ModelComp = *[0,1,2,0,2]*
ScoreComp = *[1,1,1,1,1]*	Score = *[1,1]*

Aus der rechten Spalte ersichtlich, dass in der linken Spalte:

- Werte mit Index 0 bis 2 gehören zu den Komponenten 0 bis 2 von Instanz 1.
- Werte mit Index 3 bis 4 gehören zu den Komponenten 0 und 2 von Instanz 2.

Score: Bewertung der Instanzen des gesamten Komponentenmodells.

Score ist das gewichtete Mittel der einzelnen Komponentenbewertungen, den Werten in ScoreComp. Dabei erfolgt die Gewichtung proportional zur Anzahl an Modellpunkten in der jeweiligen Komponente.

RowComp, ColumnComp und AngleComp: Positionen (RowComp, ColumnComp) und Rotationen (AngleComp) der Modellkomponenten aller gefundenen Instanzen des Komponentenmodells.

Die Koordinaten RowComp und ColumnComp sind die Koordinaten des Komponenten-Ursprungs (Referenzpunkts) im Suchbild. Der Komponenten-Ursprung hängt davon ab, wie das Modell erzeugt wurde:

- Mit create_trained_component_model durch Trainieren: Der Ursprung einer Komponente ist der Schwerpunkt der Konturregion, die der Operator train_model_components in ModelComponents zurück lieferte.
- Mit create_component_model manuell: Der Ursprung einer Komponente ist der Schwerpunkt Komponentenregion, die über ComponentRegion dem Operator create_component_model übergeben wurde.

Da sich die Relationen zwischen den Komponenten in ComponentModelID auf diesen Schwerpunkt beziehen, darf der Ursprung einer Komponente nicht mit set_shape_model_origin verändert werden.

ScoreComp: Bewertung der gefundenen Komponenteninstanzen.

Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil der Komponente im Bild zu sehen ist. Falls z.B. die Hälfte der Komponente im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

ModelComp: Index der gefundenen Komponenten.

Das Tupel enthält die Indizes der jeweiligen gefundenen Modellkomponenten (siehe create_component_model bzw. train_model_components). Damit können die Werte in RowComp, ColumnComp, AngleComp und ScoreComp den verschieden Modellkomponenten zugeordnet werden. Siehe auch das Beispiel bei ModelStart und ModelEnd.

Informationen zum Suchverlauf

Im Wesentlichen wird intern für die Suche der einzelnen Komponenten das formbasierte Matching verwendet (siehe find_shape_model).

Der Definitionsbereich des Bildes Image gibt den Suchbereich für den Referenzpunkt, d.h. die erlaubte Position, der Wurzelkomponente an.

Im Normalfall wird das Komponentenmodell innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass die Komponenten nicht gefunden werden können, wenn sie aus dem Bild herausragen, selbst wenn sie eine Bewertung größer als MinScoreComp erreichen würden (siehe oben).

Außerdem kann es sein, dass das Komponentenmodell, wenn es in einer der Pyramidenebenen den Bildrand berührt, ebenfalls nicht gefunden wird, auch wenn es im ursprünglichen Bild vollständig enthalten ist. Als Daumenregel gilt hier, dass das Modell dann nicht gefunden werden könnte, wenn seine Distanz zum Bildrand unter $2^{NumLevels-1}$ fällt. Dieses Verhalten kann mit set_system('border_shape_models','true') umgestellt werden, so dass auch Komponenten gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als MinScoreComp erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit der Suche erhöht.

Es ist weiter zu beachten, dass in seltenen Fällen, die typischerweise nur in künstlichen Bildern auftreten, auch dann kein Modell gefunden wird, wenn das Modell in einer Pyramidenebene die reduzierte Bilddomäne berührt. In diesem Fall kann es helfen, die Region der Domäne um $2^{NumLevels-1}$, z.B., mit dilation_circle zu vergrößern.

Beim Tracken der Matches durch die Bildpyramide werden auf jeder Ebene Matches verworfen, die wenig vielversprechend sind. Dadurch ist es möglich, dass Matches verworfen werden, die auf der niedrigsten Pyramidenebene einen höheren Score gehabt hätten. Beispielsweise kann das dazu führen, dass der gefundene Match für `NumMatches` gleich *1* nicht mit dem Match mit dem höchsten Score übereinstimmt, der von `NumMatches` gleich *0* oder > *1* zurückgegeben wird.

Empfehlungen

Falls erwartet wird, dass mehrere Objekte mit einem ähnlichen Score gefunden werden können, aber nur das Modell mit dem höchsten Score zurückgegeben werden soll, kann es vorteilhaft sein, `NumMatches` zu erhöhen, und dann den Match mit dem höchsten Score auszuwählen.

Für einen aussagekräftigen Score und zum Vermeiden falscher Matches wird empfohlen, das Erlauben einer Deformation mit einer Subpixel-Extraktion mit einer der Least-Squares-Adjustment-Methoden zu kombinieren. Falls die Subpixel-Extraktion und/oder die maximale Objektdeformation für jede Komponente einzeln spezifiziert werden sollen, muss in `SubPixelComp` die Subpixel-Extraktion genauso oft angegeben werden, wie die Anzahl an Modellkomponenten in `ComponentModelID`. Nach jedem übergebenen Wert für die Subpixel-Extraktion kann optional ein zweiter Wert übergeben werden, der die maximale Objektdeformation für die jeweilige Komponente beschreibt. Wird nach dem Wert für die Subpixel-Extraktion kein Wert für die Objektdeformation übergeben, wird die entsprechende Komponente ohne Deformation gesucht. Weitere Details sind in der Dokumentation von `find_shape_models` zu finden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild, in dem das Komponentenmodell gefunden werden soll.
- ▷ **`ComponentModelID`** (input_control) ..... component_model ⇝ *handle*
  Handle des Komponentenmodells.
- ▷ **`RootComponent`** (input_control) ..... integer(-array) ⇝ *integer*
  Index der Wurzelkomponente.
  Wertevorschläge: `RootComponent` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8}
- ▷ **`AngleStartRoot`** (input_control) ..... angle.rad(-array) ⇝ *real*
  Kleinste auftretende Rotation der Wurzelkomponente
  Defaultwert: -0.39
  Wertevorschläge: `AngleStartRoot` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **`AngleExtentRoot`** (input_control) ..... angle.rad(-array) ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Defaultwert: 0.79
  Wertevorschläge: `AngleExtentRoot` ∈ {6.28, 3.14, 1.57, 0.79, 0.39, 0.0}
  Restriktion: AngleExtentRoot >= 0
- ▷ **`MinScore`** (input_control) ..... real ⇝ *real*
  Minimale Bewertung der zu findenden Instanzen des Komponentenmodells.
  Defaultwert: 0.5
  Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
  Restriktion: 0 <= MinScore && MinScore <= 1
- ▷ **`NumMatches`** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu findenden Instanzen des Komponentenmodells (oder 0 für alle Treffer).
  Defaultwert: 1
  Wertevorschläge: `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}
- ▷ **`MaxOverlap`** (input_control) ..... real ⇝ *real*
  Maximale Überlappung der zu findenden Instanzen des Komponentenmodells.
  Defaultwert: 0.5
  Wertevorschläge: `MaxOverlap` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
  Restriktion: 0 <= MaxOverlap && MaxOverlap <= 1
- ▷ **`IfRootNotFound`** (input_control) ..... string ⇝ *string*
  Verhalten bei fehlender Wurzelkomponente.
  Defaultwert: 'stop_search'
  Werteliste: `IfRootNotFound` ∈ {'stop_search', 'select_new_root'}

▷ **IfComponentNotFound** (input_control) . . . . . string ⇝ *string*
Verhalten bei fehlenden Komponenten.
Defaultwert: 'prune_branch'
Werteliste: IfComponentNotFound ∈ {'prune_branch', 'search_from_upper', 'search_from_best'}

▷ **PosePrediction** (input_control) . . . . . string ⇝ *string*
Schätzung der Lagen nicht gefundener Komponenten.
Defaultwert: 'none'
Werteliste: PosePrediction ∈ {'none', 'from_neighbors', 'from_all'}

▷ **MinScoreComp** (input_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen der Komponenten.
Defaultwert: 0.5
Wertevorschläge: MinScoreComp ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05
Restriktion: 0 <= MinScoreComp && MinScoreComp <= 1

▷ **SubPixelComp** (input_control) . . . . . string(-array) ⇝ *string*
Subpixelgenauigkeit der Komponentenlagen falls ungleich *'none'*.
Defaultwert: 'least_squares'
Wertevorschläge: SubPixelComp ∈ {'none', 'interpolation', 'least_squares', 'least_squares_high', 'least_squares_very_high', 'max_deformation 1', 'max_deformation 2', 'max_deformation 3', 'max_deformation 4', 'max_deformation 5', 'max_deformation 6'}

▷ **NumLevelsComp** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen für die Komponenten (und unterste zu verwendende Pyramidenebene falls $|\texttt{NumLevelsComp}| = 2$ oder $2n$).
Defaultwert: 0
Werteliste: NumLevelsComp ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **GreedinessComp** (input_control) . . . . . real(-array) ⇝ *real*
„Gierigkeit" der Suchheuristik für die Komponenten.
Defaultwert: 0.9
Wertevorschläge: GreedinessComp ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05
Restriktion: 0 <= GreedinessComp && GreedinessComp <= 1

▷ **ModelStart** (output_control) . . . . . integer(-array) ⇝ *integer*
Startindex jeder gefundenen Instanz des Komponentenmodells auf die Tupel, die die Komponentenmatches beschreiben.

▷ **ModelEnd** (output_control) . . . . . integer(-array) ⇝ *integer*
Endindex jeder gefundenen Instanz des Komponentenmodells auf die Tupel, die die Komponentenmatches beschreiben.

▷ **Score** (output_control) . . . . . real(-array) ⇝ *real*
Bewertung der gefunden Instanzen des Komponentenmodells.

▷ **RowComp** (output_control) . . . . . point.y(-array) ⇝ *real*
Zeilenkoordinate der gefundenen Komponentenmatches.

▷ **ColumnComp** (output_control) . . . . . point.x(-array) ⇝ *real*
Spaltenkoordinate der gefundenen Komponentenmatches.

▷ **AngleComp** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Rotationswinkel der gefundenen Komponentenmatches.

▷ **ScoreComp** (output_control) . . . . . real(-array) ⇝ *real*
Bewertung der gefundenen Komponentenmatches.

▷ **ModelComp** (output_control) . . . . . integer(-array) ⇝ *integer*
Index der gefundenen Komponenten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert find_component_model den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebild vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_trained_component_model`, `create_component_model`, `read_component_model`

Nachfolger

`get_found_component_model`

Alternativen

`find_shape_models`

Siehe auch

`find_shape_model`, `find_shape_models`, `get_shape_model_params`, `get_component_model_params`, `train_model_components`, `set_shape_model_origin`, `smallest_rectangle2`

Modul

Matching

gen_initial_components (ModelImage : InitialComponents : ContrastLow, ContrastHigh, MinSize, Mode, GenericName, GenericValue :)

Erzeugung der initialen Komponenten eines Komponentenmodells.

Im Allgemeinen bestehen zwei Möglichkeiten, `gen_initial_components` zu verwenden. Die erste Möglichkeit sollte gewählt werden, wenn die initialen Komponenten eines Komponentenmodells nicht bekannt sind. Dann erzeugt `gen_initial_components` die initialen Komponenten automatisch aus einem Modellbild. Die zweite Möglichkeit kann gewählt werden, um geeignete Werte für die Merkmalsextraktionsparameter von `train_model_components` und `create_component_model` zu finden. Somit ist die zweite Möglichkeit vergleichbar mit der Funktion von `inspect_shape_model` innerhalb des formbasierten Matchings.

Wenn die erste Möglichkeit verwendet wird, erzeugt `gen_initial_components` die initialen Komponenten eines Komponentenmodells aus einem Modellbild `ModelImage`. Wie bereits erwähnt, ist dies besonders dann hilfreich, wenn die initialen Komponenten eines Komponentenmodells nicht bekannt sind. In diesem Fall können die zurückgelieferten initialen Komponenten dazu genutzt werden, das Komponentenmodell automatisch mit `train_model_components` zu trainieren, was die Extraktion der (endgültigen) Modellkomponenten und deren Relationen umfasst. `gen_initial_components` liefert die initialen Komponenten in einem Regionenobjekttupel `InitialComponents` zurück, das für jede initiale Komponente deren Konturregion enthält.

Zur automatischen Bestimmung der initialen Komponenten muss der Definitionsbereich des Modellbildes `ModelImage` das gesamte zusammengesetzte Objekt inklusive aller Komponenten beinhalten. Mit `Mode` kann die Art der automatischen Berechnung gewählt werden. Zur Zeit ist nur der Modus *'connection'* verfügbar. In diesem Modus erfolgt die automatische Berechnung in zwei Schritten: Im ersten Schritt werden Merkmale extrahiert unter Verwendung der Parameter `ContrastLow`, `ContrastHigh` und `MinSize`. Diese Parameter legen fest, aus welchen Konturen die initialen Komponenten aufgebaut werden sollen, und sollten so gewählt werden, dass die initialen Komponenten nur die signifikanten Merkmale des Modellbildes enthalten. `ContrastLow` und `ContrastHigh` legen den Grauwertkontrast fest, den die Punkte der initialen Komponenten besitzen müssen. Der Kontrast ist ein Maß für die lokalen Grauwertdifferenzen zwischen dem Objekt und dem Hintergrund und zwischen verschiedenen Teilen des Objektes. Dabei wird bei der Segmentierung des Modellbildes ein Verfahren, das ähnlich zum Hysterese-Schwellenwert-Verfahren in `edges_image` ist, eingesetzt. Dabei ist `ContrastLow` der untere Schwellenwert und `ContrastHigh` der obere Schwellenwert. Wird für `ContrastLow` und `ContrastHigh` der gleiche Wert übergeben, dann wird eine einfache Schwellenwertoperation ausgeführt. `MinSize` dient zur Selektion von signifikanten Merkmalen für die initialen Komponenten basierend auf der Größe der zusammenhängenden Konturregionen. Das bedeutet, dass Konturregionen, die weniger als `MinSize` Pixel umfassen, unterdrückt werden.

Die dabei entstehenden zusammenhängenden Konturregionen werden in einem zweiten Schritt iterativ zusammengefasst. Dabei werden zwei Konturregionen vereinigt, falls der Abstand der beiden Regionen einen bestimmten Schwellenwert unterschreitet (siehe unten). Die daraus resultierenden Regionen werden in `InitialComponents` zurückgeliefert und können zum Trainieren des Komponentenmodells an `train_model_components` übergeben werden.

Zur Steuerung der internen Bildverarbeitung können über die Parameter GenericName und GenericValue gezielt Steuerparameter übergeben werden. Dies geschieht, indem bei GenericName die Liste der Namen der Parameter als Strings übergeben werden. Bei GenericValue werden die zugehörigen Werte an den gleichen Indexpositionen übergeben.

Normalerweise ist es nicht nötig, die Werte zu ändern. Dies ist nur dann empfehlenswert, wenn das Ergebnis der automatischen Bestimmung der initialen Komponenten nicht den Erwartungen entspricht. Die zwei Parameter, die verändert werden können, sind *'merge_distance'* und *'merge_fraction'*; beide werden bei der iterativen Zusammenfassung von Konturregionen verwendet (siehe oben). Dazu wird zunächst der Anteil an Konturpixeln einer Konturregion berechnet, die maximal *'merge_distance'* Pixel von einer anderen Konturregion entfernt liegen. Ist dieser Anteil größer als der in *'merge_fraction'* übergebene Wert, dann werden diese beiden Konturregionen zusammengefasst. Daraus folgt, dass je größer *'merge_distance'* und je kleiner *'merge_fraction'* gewählt wird, desto mehr Konturregionen werden zusammengefasst. Vorbesetzung für *'merge_distance'* ist *5* und für *'merge_distance'* *0.5* (entspricht 50 Prozent).

Wenn die zweite Möglichkeit verwendet wird, d.h. wenn bereits näherungsweise bekannt ist, aus welchen Komponenten sich das Komponentenmodell zusammensetzt, kann das Training mit train_model_components ohne vorheriges Ausführen von gen_initial_components gestartet werden. Dabei können die initialen Komponenten vom Benutzer vorgegeben werden und direkt an train_model_components übergeben werden. Sind sowohl die Komponenten als auch die relativen Bewegungen (Relationen) der Komponenten bekannt, so kann sowohl auf gen_initial_components als auch auf train_model_components ganz verzichtet werden und das Komponentenmodell unmittelbar mit create_component_model erstellt werden. Dabei werden die Komponenten direkt an create_component_model übergeben. In diesen beiden Fällen kann gen_initial_components jedoch dazu genutzt werden, die Auswirkung der Merkmalsextraktionsparameter ContrastLow, ContrastHigh und MinSize von train_model_components und create_component_model zu evaluieren und somit geeignete Parameterwerte für eine bestimmte Anwendung zu finden.

Dabei müssen die Bildregionen für die (initialen) Komponenten explizit vorgegeben werden, d.h. für jede (initiale) Komponente wird ein eigenes Bild übergeben, aus dem die (initiale) Komponente generiert werden soll. In diesem Fall besteht das Modellbild ModelImage demzufolge aus einem Bildobjekttupel. Der Definitionsbereich jedes Bildobjektes wird als Region zur Berechnung der entsprechenden (initialen) Komponente verwendet. Dabei muss die Bildmatrix, die allen Bildobjekten des Bildobjekttupels zugrunde liegt, identisch sein. D.h. ModelImage kann nicht in beliebiger Weise mit concat_obj aufgebaut werden, sondern muss aus demselben Bild mit add_channels oder äquivalenten Aufrufen erzeugt werden. Ist dies nicht der Fall, wird eine Fehlermeldung zurückgeliefert. Wird für ContrastLow, ContrastHigh oder MinSize nur ein Wert übergeben, so gilt dieser Wert für alle (initialen) Komponenten. Sollen dagegen für unterschiedliche (initiale) Komponenten unterschiedliche Parameter verwendet werden, kann dies durch die Übergabe von Tupeln von Werten für die drei Parameter erreicht werden. In diesem Fall müssen die Tupel eine Länge habe, die der Anzahl an (initialen) Komponenten, d.h. der Anzahl an Bildobjekten in ModelImage, entspricht. Die Konturregionen der (initialen) Komponenten werden in InitialComponents zurückgeliefert.

Damit ist die zweite Möglichkeit äquivalent zur Funktion von inspect_shape_model innerhalb des formbasierten Matchings. Jedoch werden bei gen_initial_components im Unterschied zu inspect_shape_model die Konturregionen nicht auf mehreren Pyramidenebenen generiert. Deshalb sollte für den Fall, dass die Anzahl an Pyramidenebenen manuell gewählt wird, vorzugsweise inspect_shape_model für jede (initiale) Komponente individuell aufgerufen werden.

Für beide oben beschriebenen Möglichkeiten können die Parameter ContrastLow, ContrastHigh und MinSize auch automatisch bestimmt werden. In diesem Fall muss der Wert *'auto'* übergeben werden. Sollen beide Hysterese-Schwellen automatisch bestimmt werden, so müssen sowohl ContrastLow als auch ContrastHigh jeweils auf *'auto'* gesetzt werden. Soll dagegen lediglich ein Kontrastwert automatisch bestimmt werden, so muss ContrastLow auf *'auto'* und ContrastHigh auf einen beliebigen Wert ungleich *'auto'* gesetzt werden.

Falls das Eingabebild ModelImage einen Kanal besitzt, wird zur Berechnung der Repräsentation des Modells das Verfahren verwendet, das auch bei create_component_model oder create_trained_component_model für die Metriken *'use_polarity'*, *'ignore_global_polarity'* und *'ignore_local_polarity'* verwendet wird. Falls das Eingabebild mehr als einen Kanal besitzt, wird das Verfahren verwendet, das für die Metrik *'ignore_color_polarity'* verwendet wird.

Parameter

▷ **ModelImage** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Eingabebild, aus dem die initialen Komponenten erstellt werden sollen.

▷ **InitialComponents** (output_object) . . . . . region-array ⇝ *object*
Konturregionen der initiale Komponenten.

▷ **ContrastLow** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Unterer Hystereseschwellenwert für den Kontrast der initialen Komponenten im Bild.
Defaultwert: 'auto'
Wertevorschläge: `ContrastLow` ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}
Restriktion: ContrastLow > 0

▷ **ContrastHigh** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Oberer Hystereseschwellenwert für den Kontrast der initialen Komponenten im Bild.
Defaultwert: 'auto'
Wertevorschläge: `ContrastHigh` ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}
Restriktion: ContrastHigh > 0 && ContrastHigh >= ContrastLow

▷ **MinSize** (input_control) . . . . . integer(-array) ⇝ *integer* / string
Mindestgröße der initialen Komponenten.
Defaultwert: 'auto'
Wertevorschläge: `MinSize` ∈ {'auto', 0, 5, 10, 20, 30, 40}
Restriktion: MinSize >= 0

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Art der automatischen Segmentierung.
Defaultwert: 'connection'
Werteliste: `Mode` ∈ {'connection'}

▷ **GenericName** (input_control) . . . . . string(-array) ⇝ *string*
Namen der optionalen Steuerparameter.
Defaultwert: []
Werteliste: `GenericName` ∈ {'merge_distance', 'merge_fraction'}

▷ **GenericValue** (input_control) . . . . . number(-array) ⇝ *real* / integer
Wert der optionalen Steuerparameter.
Defaultwert: []

Beispiel

```
* First example that shows the use of gen_initial_components to automatically
* extract the initial components from a model image.

* Get the model image.
read_image (Image, 'model_image.tif')
* Define the entire model region.
gen_rectangle1 (ModelRegion, 119, 106, 330, 537)
reduce_domain (Image, ModelRegion, ModelImage)
* Automatically generate the initial components.
gen_initial_components (ModelImage, InitialComponents, 40, 40, 20, \
                        'connection', [], [])
* Extract the model components and train the relations.
train_model_components (ModelImage, InitialComponents, TrainingImages, \
                        ModelComponents, 40, 40, 20, 0.85, 0, 0, rad(15), \
                        'speed', 'rigidity', 0.2, 0.5, ComponentTrainingID)
* Create the component model based on the training result.
create_trained_component_model (ComponentTrainingID, -rad(30), rad(60), 10, \
                                0.8, 'auto', 'auto', 'none', \
                                'use_polarity', 'false', ComponentModelID, \
                                RootRanking)

* Second example that shows the use of gen_initial_components to evaluate
* the effect of the feature extraction parameters.
```

```
* Get the model image.
read_image (ModelImage, 'model_image.tif')
* Define the regions for the components.
gen_rectangle2 (ComponentRegions, 318, 109, -1.62, 34, 19)
gen_rectangle2 (Rectangle2, 342, 238, -1.63, 32, 17)
gen_rectangle2 (Rectangle3, 355, 505, 1.41, 25, 17)
concat_obj (ComponentRegions, Rectangle2, ComponentRegions)
concat_obj (ComponentRegions, Rectangle3, ComponentRegions)
add_channels (ComponentRegions, ModelImage, ModelImageReduced)
gen_initial_components (ModelImageReduced, InitialComponents, 15, 40, 15, \
                        'connection', [], [])
* Create the component model by explicitly specifying the relations.
create_component_model (ModelImage, ComponentRegions, 20, 20, rad(25), 0, \
                        rad(360), 15, 40, 15, 10, 0.8, 'auto', 'auto', \
                        'none', 'use_polarity', 'false', ComponentModelID, \
                        RootRanking)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_initial_components` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`draw_region`, `add_channels`, `reduce_domain`

Nachfolger

`train_model_components`

Alternativen

`inspect_shape_model`

Modul

Matching

get_component_model_params (: : ComponentModelID : MinScoreComp, RootRanking, ShapeModelIDs)

Auslesen der Parameter eines Komponentenmodells.

Der Operator `get_component_model_params` gibt die Parameter des Komponentenmodells `ComponentModelID` zurück. Dies ist insbesondere hilfreich, um nach einem Einlesen eines Komponentenmodells mit `read_component_model` die Parameter `MinScoreComp` und `RootRanking` zu überprüfen. Außerdem gibt der Operator die IDs `ShapeModelIDs` der Formmodelle aller Modellkomponenten zurück. Die Reihenfolge der zurückgelieferten IDs entspricht den Indizes der entsprechenden Komponenten im Komponentenmodell `ComponentModelID`. Die IDs können dazu verwendet werden, die Parameter der zugehörigen Formmodelle mit `get_shape_model_params` abzufragen.

Parameter

▷ **ComponentModelID** (input_control) . . . . . component_model ⇝ *handle*
Handle des Komponentenmodells.

▷ **MinScoreComp** (output_control) . . . . . real(-array) ⇝ *real*
Minimale Bewertung der zu findenden Instanzen der Komponenten.

▷ **RootRanking** (output_control) ..... integer(-array) ⇝ *integer*
Reihenfolge der Modellkomponenten, die deren Eignung als Wurzelkomponente zu fungieren ausdrückt.

▷ **ShapeModelIDs** (output_control) ..... shape_model(-array) ⇝ *handle*
Handles der Formmodelle der einzelnen Modellkomponenten.

Beispiel

```
read_component_model ('pliers.cpm', ComponentModelID)
get_component_model_params (ComponentModelID, MinScoreComp, RootRanking, \
                            ShapeModelIDs)
for i := 0 to |ShapeModelIDs|-1 by 1
    get_shape_model_params (ShapeModelIDs[i], NumLevels, AngleStart, \
                            AngleExtent, AngleStep, ScaleMin, ScaleMax, \
                            ScaleStep, Metric, MinContrast)
endfor
```

Ergebnis

Ist der Handle des Komponentenmodells gültig, dann liefert `get_component_model_params` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`create_trained_component_model`, `create_component_model`

Siehe auch

`get_shape_model_params`

Modul

Matching

get_component_model_tree (: Tree, Relations : ComponentModelID, RootComponent, Image : StartNode, EndNode, Row, Column, Phi, Length1, Length2, AngleStart, AngleExtent)

Liefert den Suchbaum eines Komponentenmodells zurück.

`get_component_model_tree` liefert den Suchbaum `Tree` und die zugehörigen Relationen `Relations` des Komponentenmodells, das in `ComponentModelID` übergeben wird, sowohl in Form von Regionen als auch in numerischer Form zurück. `get_component_model_tree` ist besonders hilfreich, um die mit `create_trained_component_model` oder `create_component_model` berechnete Suchreihenfolge der einzelnen Komponenten zu visualisieren.

Da je nach Wurzelkomponente ein anderer Suchbaum verwendet wird, muss die Wurzelkomponente in `RootComponent` übergeben werden. Die Knoten im Baum `Tree` repräsentieren die einzelnen Komponenten im Modell, die Verbindungen zwischen den Knoten zeigen an, welche Komponenten relativ zueinander gesucht werden. Die Lage der Knoten im Baum entspricht der Lage der Komponenten im Modellbild (wenn `Image` = *'model_image'* oder `Image` = *0*) oder in einem Trainingsbild (wenn `Image` ≥ *1*). Falls `Image` ≥ *1* gewählt wird, muss das Komponentenmodell basierend auf dem Trainingsergebnis mit `create_trained_component_model` generiert worden sein.

Sei *n* die Anzahl an Komponenten in `ComponentModelID`. Das Regionenobjekttupel `Relations` der Länge *n* hat folgenden Aufbau: Für jede Komponente wird eine eigene Region zurückgeliefert. Die Positionen aller Komponenten im Bild werden durch Kreise mit einem Radius von 3 Pixel dargestellt. Für jede Komponente, die nicht der

Wurzelkomponente RootComponent entspricht, werden zusätzlich die Positionsrelation und die Winkelrelation relativ zur Vorgänger-Komponente im Baum dargestellt. Die Positionsrelation wird dabei durch ein Rechteck und die Winkelrelation durch einen Kreissektor mit einem Radius von 30 Pixel dargestellt. Der Kreismittelpunkt liegt dabei an der mittleren relativen Position der Komponente. Das Rechteck beschreibt die Referenzpunktbewegung der jeweiligen Komponente relativ zur Lage der Vorgänger-Komponente, der Öffnungswinkel des Kreissektors beschreibt die Variation der relativen Orientierung. Eine relative Orientierung von 0 entspricht der relativen Orientierung der beiden Komponenten im Modellbild.

Zusätzlich zu den Regionen werden der Suchbaum und die entsprechenden Relationen auch in numerischer Form zurückgeliefert. Der Suchbaum wird dabei in Form zweier Tupel StartNode und EndNode der Länge *n* beschrieben, die die jeweiligen Anfangs- und Endknoten aller Verbindungen im Baum enthalten. Dabei enthalten die Knoten die Indizes der entsprechenden Komponenten. Während der Suche wird die Komponenten, die durch den Endknoten beschrieben ist, relativ zur Lage der Komponente, die durch den Anfangsknoten beschrieben ist (Vorgängerkomponente), gesucht. Da die Wurzelkomponente nicht relativ zu einer anderen Komponente gesucht wird und somit keine Vorgängerkomponente besitzt, wird der entsprechende Anfangsknoten auf *-1* gesetzt. Die Relationen werden in Form von numerischen Werten in Row, Column, Phi, Length1, Length2, AngleStart und AngleExtent zurückgeliefert. Diese Parameter sind Tupel der Länge *n* und enthalten die Relationen aller Komponenten relativ zu ihrer Vorgängerkomponente im Baum, wobei die Reihenfolge der Werte in den Tupeln durch den Index der entsprechenden Komponenten festgelegt ist. Die Positionsrelation wird dabei durch die Parameter des entsprechenden Rechtecks Row, Column, Phi, Length1 und Length2 beschrieben (siehe gen_rectangle2). Die Winkelrelation wird durch den Anfangswinkel AngleStart und die Winkelausdehnung AngleExtent beschrieben.

Für die Wurzelkomponente und für Komponenten, die in dem entsprechenden Bild keine Vorgängerkomponente besitzen oder nicht gefunden wurden, wird eine leere Region zurückgeliefert und die entsprechenden Werte der sieben Parameter auf *0* gesetzt.

Parameter

- ▷ **Tree** (output_object) . . . . . region ⇝ *object*
  Suchbaum
- ▷ **Relations** (output_object) . . . . . region-array ⇝ *object*
  Relationen der im Suchbaum verbundenen Komponenten.
- ▷ **ComponentModelID** (input_control) . . . . . component_model ⇝ *handle*
  Handle des Komponentenmodells.
- ▷ **RootComponent** (input_control) . . . . . integer(-array) ⇝ *integer*
  Index der Wurzelkomponente.
  Wertevorschläge: RootComponent ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8}
- ▷ **Image** (input_control) . . . . . string ⇝ *string* / integer
  Bild, für das der Baum zurückgeliefert werden soll.
  Defaultwert: 'model_image'
  Wertevorschläge: Image ∈ {'model_image', 0, 1, 2, 3, 4, 5, 6, 7, 8}
- ▷ **StartNode** (output_control) . . . . . integer(-array) ⇝ *integer*
  Komponentenindex des Startknotens einer Verbindung im Suchbaum.
- ▷ **EndNode** (output_control) . . . . . integer(-array) ⇝ *integer*
  Komponentenindex des Endknotens einer Verbindung im Suchbaum.
- ▷ **Row** (output_control) . . . . . rectangle2.center.y(-array) ⇝ *real*
  Zeilenkoordinate des Mittelpunktes des Rechtecks, das die Relation beschreibt.
- ▷ **Column** (output_control) . . . . . rectangle2.center.x(-array) ⇝ *real*
  Spaltenindex des Mittelpunktes des Rechtecks, das die Relation beschreibt.
- ▷ **Phi** (output_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real*
  Orientierung des Rechtecks, das die Relation beschreibt (Bogenmaß).
  Zusicherung: - pi / 2 < Phi && Phi <= pi / 2
- ▷ **Length1** (output_control) . . . . . rectangle2.hwidth(-array) ⇝ *real*
  Erster Halbmesser (halbe Länge) des Rechtecks, das die Relation beschreibt.
  Zusicherung: Length1 >= 0.0
- ▷ **Length2** (output_control) . . . . . rectangle2.hheight(-array) ⇝ *real*
  Zweiter Halbmesser (halbe Breite) des Rechtecks, das die Relation beschreibt.
  Zusicherung: Length2 >= 0.0 && Length2 <= Length1

▷ **AngleStart** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Kleinster relativer Orientierungswinkel.

▷ **AngleExtent** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Ausdehnung des relativen Orientierungswinkelbereichs.

Beispiel

```
* Read the model image.
read_image (ModelImage, 'model_image.tif')
* Describe the model components.
gen_rectangle2 (ComponentRegions, 318, 109, -1.62, 34, 19)
gen_rectangle2 (Rectangle2, 342, 238, -1.63, 32, 17)
gen_rectangle2 (Rectangle3, 355, 505, 1.41, 25, 17)
concat_obj (ComponentRegions, Rectangle2, ComponentRegions)
concat_obj (ComponentRegions, Rectangle3, ComponentRegions)
* Create the component model.
create_component_model (ModelImage, ComponentRegions, 20, 20, rad(25), 0, \
                        rad(360), 15, 40, 15, 10, 0.8, 0, 0, 'none', \
                        'use_polarity', 'true', ComponentModelID, \
                        RootRanking)
* Get the component model tree.
get_component_model_tree (Tree, Relations, ComponentModelID, RootRanking, \
                          'model_image', StartNode, EndNode, Row, Column, \
                          Phi, Length1, Length2, AngleStart, AngleExtent)
dev_set_colored (12)
dev_display (ModelImage)
dev_display (Tree)
dev_display (Relations)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_component_model_tree` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_trained_component_model`, `create_component_model`

Siehe auch

`train_model_components`

Modul

Matching

get_component_relations (: Relations : ComponentTrainingID, ReferenceComponent, Image : Row, Column, Phi, Length1, Length2, AngleStart, AngleExtent)

Liefert die in einem Trainingsergebnis gespeicherten Relationen zwischen den Modellkomponenten zurück.

`get_component_relations` liefert die Relationen zwischen Modellkomponenten nachdem sie mit `train_model_components` trainiert worden sind. Mit dem Parameter `ReferenceComponent` kann eine Referenzkomponente ausgewählt werden. `get_component_relations` liefert dann die Relationen zwischen der Referenzkomponente und allen übrigen Komponenten im Modellbild (wenn `Image` = *'model_image'* oder `Image` = *0*) oder in einem Trainingsbild (wenn `Image` ≥ *1*). Um die Relationen im *i*. Trainingsbild zu erhalten,

muss Image auf *i* gesetzt werden. Das Ergebnis des mit train_model_components durchgeführten Trainings muss in ComponentTrainingID übergeben werden. ReferenceComponent beschreibt den Index der entsprechenden Referenzkomponente und muss somit im Bereich von 0 bis *n*-1 liegen, wenn *n* die Anzahl der Modellkomponenten darstellt (siehe train_model_components).

Die Relationen werden sowohl in Form von Regionen in Relations als auch in Form von numerischen Werten in Row, Column, Phi, Length1, Length2, AngleStart und AngleExtent zurückgeliefert.

Das Regionenobjekttupel Relations hat folgenden Aufbau: Für jede Komponente wird eine eigene Region zurückgeliefert. Somit beinhaltet Relations *n* Regionen, wobei die Reihenfolge der Regionen im Tupel durch den Index der entsprechenden Komponenten festgelegt ist. Die Positionen aller Komponenten im Bild werden durch Kreise mit einem Radius von 3 Pixel dargestellt. Für jede Komponente, die nicht der Referenzkomponente ReferenceComponent entspricht, werden zusätzlich die Positionsrelation und die Winkelrelation relativ zur Referenzkomponente dargestellt. Die Positionsrelation wird dabei durch ein Rechteck und die Winkelrelation durch einen Kreissektor mit einem Radius von 30 Pixel dargestellt. Der Kreismittelpunkt liegt dabei an der mittleren relativen Position der Komponente. Das Rechteck beschreibt die Referenzpunktbewegung der jeweiligen Komponente relativ zur Lage der Referenzkomponente, der Öffnungswinkel des Kreissektors beschreibt die Variation der relativen Orientierung (siehe train_model_components). Eine relative Orientierung von 0 entspricht der relativen Orientierung der beiden Komponenten im Modellbild. Zeigen die beiden Komponenten in allen Bildern die gleiche relative Orientierung entartet der Kreissektor demzufolge zu einer Geraden.

Neben dem Regionenobjekttupel Relations werden die Relationen auch in Form von numerischen Werten Row, Column, Phi, Length1, Length2, AngleStart und AngleExtent zurückgeliefert. Diese Parameter sind Tupel der Länge *n* und enthalten die Relationen aller Komponenten relativ zur Referenzkomponente, wobei die Reihenfolge der Werte in den Tupeln durch den Index der entsprechenden Komponenten festgelegt ist. Die Positionsrelation wird dabei durch die Parameter des entsprechenden Rechtecks Row, Column, Phi, Length1 und Length2 beschrieben (siehe gen_rectangle2). Die Winkelrelation wird durch den Anfangswinkel AngleStart und die Winkelausdehnung AngleExtent beschrieben. Für die Referenzkomponente wird lediglich deren Position im Bild in Row und Column zurückgeliefert. Alle anderen Werte sind dagegen auf *0* gesetzt.

Wurde die Referenzkomponente im Bild nicht gefunden, so wird ein Tupel leerer Regionen zurückgeliefert und die entsprechenden Parameterwerte werden auf *0* gesetzt.

Der Operator get_component_relations kann dazu genutzt werden, das Ergebnis des mit train_model_components durchgeführten Trainings zu visualisieren. Damit ist es möglich, die in den Trainingsbildern enthaltenen Variationen einzuschätzen. Eventuell kann es sinnvoll sein, mit einem veränderten Satz von Trainingsbildern das Training mit train_model_components erneut durchzuführen.

Parameter

- ▷ **Relations** (output_object) . . . . . region(-array) ⇝ *object*
  Darstellung der Relationen in Form von Regionen.
- ▷ **ComponentTrainingID** (input_control) . . . . . component_training ⇝ *handle*
  Handle des Trainingsergebnisses.
- ▷ **ReferenceComponent** (input_control) . . . . . integer ⇝ *integer*
  Index der Referenzkomponente.
  Restriktion: ReferenceComponent >= 0
- ▷ **Image** (input_control) . . . . . string ⇝ *string* / integer
  Bild, für das die Relationen zwischen den Komponenten zurückgeliefert werden sollen.
  Defaultwert: 'model_image'
  Wertevorschläge: Image ∈ {'model_image', 0, 1, 2, 3, 4, 5, 6, 7, 8}
- ▷ **Row** (output_control) . . . . . rectangle2.center.y(-array) ⇝ *real*
  Zeilenkoordinate des Mittelpunktes des Rechtecks, das die Relation beschreibt.
- ▷ **Column** (output_control) . . . . . rectangle2.center.x(-array) ⇝ *real*
  Spaltenindex des Mittelpunktes des Rechtecks, das die Relation beschreibt.
- ▷ **Phi** (output_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real*
  Orientierung des Rechtecks, das die Relation beschreibt (Bogenmaß).
  Zusicherung: - pi / 2 < Phi && Phi <= pi / 2
- ▷ **Length1** (output_control) . . . . . rectangle2.hwidth(-array) ⇝ *real*
  Erster Halbmesser (halbe Länge) des Rechtecks, das die Relation beschreibt.
  Zusicherung: Length1 >= 0.0

▷ **Length2** (output_control) ..... rectangle2.hheight(-array) ⇝ *real*
Zweiter Halbmesser (halbe Breite) des Rechtecks, das die Relation beschreibt.
Zusicherung: Length2 >= 0.0 && Length2 <= Length1

▷ **AngleStart** (output_control) ..... angle.rad(-array) ⇝ *real*
Kleinster relativer Orientierungswinkel.

▷ **AngleExtent** (output_control) ..... angle.rad(-array) ⇝ *real*
Ausdehnung des relativen Orientierungswinkelbereichs.

Ergebnis

Ist der Handle des Trainingsergebnisses gültig, dann liefert `get_component_relations` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_model_components`

Nachfolger

`train_model_components`

Siehe auch

`gen_rectangle2`

Modul

Matching

get_found_component_model (: FoundComponents : ComponentModelID, ModelStart, ModelEnd, RowComp, ColumnComp, AngleComp, ScoreComp, ModelComp, ModelMatch, MarkOrientation : RowCompInst, ColumnCompInst, AngleCompInst, ScoreCompInst)

Liefert die Komponenten einer gefundenen Instanz eines Komponentenmodells zurück.

`get_found_component_model` liefert die Komponenten einer gefundenen Instanz des Komponentenmodells `ComponentModelID` sowohl in Form von Konturregionen in `FoundComponents` als auch in numerischer Form zurück.

Der Operator `get_found_component_model` kann dazu genutzt werden, das Ergebnis der mit `find_component_model` erhaltenen Matches zu visualisieren.

Die Lage der zurückgelieferten Komponenten entspricht deren Lage im Suchbild, wie sie mit `find_component_model` erhalten wurde. Daher müssen die Parameter `ModelStart`, `ModelEnd`, `RowComp`, `ColumnComp`, `AngleComp`, `ScoreComp` und `ModelComp` an `get_found_component_model` so übergeben werden wie sie von `find_component_model` zurückgeliefert worden sind. In `ModelMatch` muss der Index der gefunden Instanz des Komponentenmodells übergeben werden. `ModelMatch` muss somit zwischen 0 und *m*-1 liegen, wobei *m* gleich der Anzahl an Elemente in den Parametern `ModelStart` und `ModelEnd` ist und somit der Anzahl der gefundenen Instanzen entspricht. Soll zum Beispiel der beste aller gefundenen Matches zurückgeliefert werden, so muss `ModelMatch` auf *0* gesetzt werden.

Bei rotationssymmetrischen Komponenten kann es darüber hinaus hilfreich sein, deren Orientierung zu markieren. Dies kann erreicht werden, indem `MarkOrientation` auf *'true'* gesetzt wird. In diesem Fall wird die Konturregion der jeweiligen Komponente an ihrem Referenzpunkt durch einen Pfeil ergänzt, der in die Referenzrichtung zeigt. Die Referenzrichtung einer Komponente ergibt sich aus deren Orientierung im Modellbild (siehe `train_model_components` oder `create_component_model`) und wird beschrieben durch einen vom Referenzpunkt horizontal nach rechts zeigenden Pfeil.

Zur Vereinfachung werden sowohl die Lageparameter jeder Komponente als auch deren Bewertung (Score) der gefunden Instanz zusätzlich in numerischer Form in `RowCompInst`, `ColumnCompInst`, `AngleCompInst` und

ScoreCompInst zurückgegeben. Diese vier Tupel haben immer die Länge *n*, wobei *n* die Anzahl der Komponenten im Komponentenmodell ComponentModelID beschreibt. Konnte während der Suche eine Komponente nicht gefunden werden, so wird im entsprechenden Element in FoundComponents eine leere Region zurückgeliefert und der Wert des entsprechenden Elements in RowCompInst, ColumnCompInst, AngleCompInst und ScoreCompInst auf *0* gesetzt.

Parameter

- ▷ **FoundComponents** (output_object) ..... region-array ⇝ *object*
  Gefundene Komponenten der ausgewählten Komponentenmodell-Instanz.
- ▷ **ComponentModelID** (input_control) ..... component_model ⇝ *handle*
  Handle des Komponentenmodells.
- ▷ **ModelStart** (input_control) ..... integer(-array) ⇝ *integer*
  Startindex jeder gefundenen Instanz des Komponentenmodells auf die Tupel, die die Komponentenmatches beschreiben.
- ▷ **ModelEnd** (input_control) ..... integer(-array) ⇝ *integer*
  Endindex jeder gefundenen Instanz des Komponentenmodells auf die Tupel, die die Komponentenmatches beschreiben.
- ▷ **RowComp** (input_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinate der gefundenen Komponentenmatches.
- ▷ **ColumnComp** (input_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinate der gefundenen Komponentenmatches.
- ▷ **AngleComp** (input_control) ..... angle.rad(-array) ⇝ *real*
  Rotationswinkel der gefundenen Komponentenmatches.
- ▷ **ScoreComp** (input_control) ..... real(-array) ⇝ *real*
  Bewertung der gefundenen Komponentenmatches.
- ▷ **ModelComp** (input_control) ..... integer(-array) ⇝ *integer*
  Index der gefundenen Komponenten.
- ▷ **ModelMatch** (input_control) ..... integer ⇝ *integer*
  Index der gefundenen Instanz des Komponentenmodells, die zurückgeliefert werden soll.
- ▷ **MarkOrientation** (input_control) ..... string ⇝ *string*
  Markieren der Komponentenorientierung.
  Defaultwert: 'false'
  Werteliste: MarkOrientation ∈ {'true', 'false'}
- ▷ **RowCompInst** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinate aller Komponenten der gewählten Modellinstanz.
- ▷ **ColumnCompInst** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinate aller Komponenten der gewählten Modellinstanz.
- ▷ **AngleCompInst** (output_control) ..... angle.rad(-array) ⇝ *real*
  Rotationswinkel aller Komponenten der gewählten Modellinstanz.
- ▷ **ScoreCompInst** (output_control) ..... real(-array) ⇝ *real*
  Bewertung aller Komponenten der gewählten Modellinstanz.

Beispiel

```
* Read a component model from file.
read_component_model ('pliers.cpm', ComponentModelID)
* Find the component model in a run-time image.
read_image (SearchImage, 'search_image.tif')
find_component_model (SearchImage, ComponentModelID, RootRanking, 0, \
                      rad(360), 0.5, 0, 0.5, 'stop_search', 'prune_branch', \
                      'none', 0.8, 'least_squares', 0, 0.8, ModelStart, \
                      ModelEnd, Score, RowComp, ColumnComp, AngleComp, \
                      ScoreComp, ModelComp)
* Visualize the found instances.
for i := 0 to |ModelStart|-1 by 1
    get_found_component_model (FoundComponents, ComponentModelID, \
                               ModelStart, ModelEnd, RowComp, ColumnComp, \
```

```
                                    AngleComp, ScoreComp, ModelComp, i, 'false', \
                                    RowCompInst, ColumnCompInst, AngleCompInst, \
                                    ScoreCompInst)
    dev_display (FoundComponents)
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_found_component_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_component_model`

Siehe auch

`train_model_components`, `create_component_model`

Modul

Matching

`get_training_components` `( : TrainingComponents : ComponentTrainingID, Components, Image, MarkOrientation : Row, Column, Angle, Score )`

Liefert die initialen Komponenten oder die Modellkomponenten in einem bestimmten Bild zurück.

`get_training_components` liefert alle initialen Komponenten (wenn `Components` = *'initial_components'*) oder alle Modellkomponenten (wenn `Components` = *'model_components'*) in Form von Konturregionen in `TrainingComponents` sowie in numerischer Form zurück. Alternativ dazu können durch direkte Angabe des Index einer initialen Komponente alle deren gefundenen Lagen (d.h. vor Auflösung der Mehrdeutigkeiten in `train_model_components`) erhalten werden.

Die Lage der zurückgelieferten Komponenten entspricht deren Lage im Modellbild (wenn `Image` = *'model_image'* oder `Image` = *0*) oder in einem Trainingsbild (wenn `Image` $\geq$ *1*). Um die Komponenten in der Lage zu erhalten, in der sie im *i*ten Trainingsbild gefunden wurden, muss `Image` auf *i* gesetzt werden. Darüber hinaus kann es bei rotationssymmetrischen Komponenten hilfreich sein, deren Orientierung im jeweiligen Bild zu markieren. Dies kann erreicht werden, indem `MarkOrientation` auf *'true'* gesetzt wird. In diesem Fall wird die Konturregion der jeweiligen Komponente an ihrem Referenzpunkt durch einen Pfeil ergänzt, der in die Referenzrichtung zeigt. Die Referenzrichtung einer Komponente ergibt sich aus deren Orientierung im Modellbild und wird beschrieben durch einen vom Referenzpunkt horizontal nach rechts zeigenden Pfeil.

Zusätzlich zu den Konturregionen wird auch die Lage und die Bewertung aller gefundenen Komponenten in `Row`, `Column`, `Angle` und `Score` zurückgeliefert (siehe `find_shape_model`). Wurde für `Components` *'initial_components'* bzw. *'model_components'* gewählt, so enthalten die Tupel `TrainingComponents`, `Row`, `Column`, `Angle` und `Score` immer die gleiche Anzahl an Elementen wie initiale Komponenten bzw. Modellkomponenten in `ComponentTrainingID` enthalten sind. Wurde eine Komponente im Bild nicht gefunden, so wird für `TrainingComponents` eine leere Region und für die Ausgabe-Steuerparameter jeweils *0* als Wert zurückgeliefert. Wurde für `Components` dagegen der Index einer initialen Komponente übergeben, so enthalten die genannten Tupel genauso viele Elemente wie Matches der jeweiligen initialen Komponente im Bild gefunden wurden.

Der Operator `get_training_components` kann dazu genutzt werden, das Ergebnis `ComponentTrainingID` des mit `train_model_components` durchgeführten Trainings zu visualisieren. Damit ist es möglich, die Eignung der Trainingsbilder einzuschätzen oder auch die Parameterwahl von `train_model_components` zu überprüfen. Eventuell kann es sinnvoll sein, mit einem veränderten Satz von Trainingsbildern oder nach Justieren der Parameter das Training mit `train_model_components` erneut durchzuführen.

Parameter

- ▷ **TrainingComponents** (output_object) ..... region(-array) ⇝ *object*
  Konturregionen der initialen Komponenten oder der Modellkomponenten.
- ▷ **ComponentTrainingID** (input_control) ..... component_training ⇝ *handle*
  Handle des Trainingsergebnisses.
- ▷ **Components** (input_control) ..... string ⇝ *string* / integer
  Art der zurückgelieferten Komponenten oder Index der initialen Komponente.
  Defaultwert: 'model_components'
  Wertevorschläge: `Components` ∈ {'model_components', 'initial_components', 0, 1, 2, 3, 4, 5}
- ▷ **Image** (input_control) ..... string ⇝ *string* / integer
  Bild, für das die Komponenten zurückgeliefert werden sollen.
  Defaultwert: 'model_image'
  Wertevorschläge: `Image` ∈ {'model_image', 0, 1, 2, 3, 4, 5, 6, 7, 8}
- ▷ **MarkOrientation** (input_control) ..... string ⇝ *string*
  Markieren der Komponentenorientierung.
  Defaultwert: 'false'
  Werteliste: `MarkOrientation` ∈ {'true', 'false'}
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinate der gefundenen Instanz aller initialen Komponenten oder Modellkomponenten.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinate der gefundenen Instanz aller initialen Komponenten oder Modellkomponenten.
- ▷ **Angle** (output_control) ..... angle.rad(-array) ⇝ *real*
  Rotationswinkel der gefundenen Instanz aller initialen Komponenten oder Modellkomponenten.
- ▷ **Score** (output_control) ..... real(-array) ⇝ *real*
  Bewertung der gefundenen Instanz aller initialen Komponenten oder Modellkomponenten.

Beispiel

```
* Get the model image.
read_image (ModelImage, 'model_image.tif')
* Define the regions for the initial components.
gen_rectangle2 (InitialComponentRegions, 212, 233, 0.62, 167, 29)
gen_rectangle2 (Rectangle2, 298, 363, 1.17, 162, 34)
gen_rectangle2 (Rectangle3, 63, 444, -0.26, 50, 27)
gen_rectangle2 (Rectangle4, 120, 473, 0, 33, 20)
concat_obj (InitialComponentRegions, Rectangle2, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle3, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle4, InitialComponentRegions)
* Get the training images.
gen_empty_obj (TrainingImages)
for i := 1 to 4 by 1
    read_image (TrainingImage, 'training_image-'+i+'.tif')
    concat_obj (TrainingImages, TrainingImage, TrainingImages)
endfor
* Extract the model components and train the relations.
train_model_components (ModelImage, InitialComponentRegions, \
                        TrainingImages, ModelComponents, 22, 60, 30, 0.6, \
                        0, 0, rad(60), 'speed', 'rigidity', 0.2, 0.4, \
                        ComponentTrainingID)
* Visualize the result of the training.
count_obj (InitialComponentRegions, NumInitComp)
count_obj (TrainingImages, NumTrainings)
for i := 1 to NumTrainings by 1
    select_obj (TrainingImages, TrainingImage, i)
    for j := 0 to NumInitComp-1 by 1
        * Visualize the ambiguous poses of each initial component.
        get_training_components (TrainingComponents, ComponentTrainingID, \
                                 j, i, 'false', Row, Column, Angle, Score)
```

```
    endfor
    * Visualize the final poses of the initial components.
    get_training_components (TrainingComponents, ComponentTrainingID, \
                             'initial_components', i, 'false', \
                             Row, Column, Angle, Score)
    * Visualize the final poses of the model components.
    get_training_components (TrainingComponents, ComponentTrainingID, \
                             'model_components', i, 'false', \
                             Row, Column, Angle, Score)
endfor
```

Ergebnis

Ist der Handle des Trainingsergebnisses gültig, dann liefert `get_training_components` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_model_components`

Nachfolger

`train_model_components`

Siehe auch

`find_shape_model`

Modul

Matching

inspect_clustered_components (
: ModelComponents : ComponentTrainingID, AmbiguityCriterion,
MaxContourOverlap, ClusterThreshold :)

überprüfen der im Training ermittelten starren Modellkomponenten.

`inspect_clustered_components` erzeugt eine Repräsentation der starren Modellkomponenten in Form von Konturregionen, auf Grundlage des Trainingsergebnisses `ComponentTrainingID`. Die resultierenden starren Modellkomponenten werden in Abhängigkeit des Kriteriums zur Lösung der Mehrdeutigkeiten `AmbiguityCriterion`, der maximalen Konturüberlappung `MaxContourOverlap` und des Cluster-Schwellenwertes `ClusterThreshold` berechnet (siehe `train_model_components`). Der Cluster-Schwellenwert beschreibt zum Beispiel das Maß, in dem initiale Komponenten zu starren Modellkomponenten zusammengefasst werden. Je größer der Cluster-Schwellenwert gewählt wird, desto weniger Zusammenschlüsse finden statt. Die ermittelten starren Modellkomponenten werden in `ModelComponents` zurückgeliefert.

Mit Hilfe von `inspect_clustered_components` ist es damit möglich, nach einmaligem Trainieren der Komponenten mit `train_model_components` die Auswirkungen verschiedener Werte der Parameter `AmbiguityCriterion`, `MaxContourOverlap` und `ClusterThreshold` auf die Bildung der starren Modellkomponenten abzuschätzen, ohne das komplette Training mehrmals durchführen zu müssen. Sind die gewünschten Parameterwerte gefunden, können diese effizient in das Trainingsergebnis mit `cluster_model_components` übernommen werden.

Parameter

▷ **ModelComponents** (output_object) ..... region(-array) ⇝ *object*
Konturregionen der starren Modellkomponenten.

▷ **ComponentTrainingID** (input_control) ..... component_training ⇝ *handle*
Handle des Trainingsergebnisses.

▷ **AmbiguityCriterion** (input_control) ..... string ⇝ *string*
Kriterium zur Lösung der Mehrdeutigkeiten.
Defaultwert: 'rigidity'
Werteliste: AmbiguityCriterion ∈ {'distance', 'orientation', 'distance_orientation', 'rigidity'}

▷ **MaxContourOverlap** (input_control) ..... real ⇝ *real*
Maximale Konturüberlappung gefundener initialer Komponenten.
Defaultwert: 0.2
Wertevorschläge: MaxContourOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05
Restriktion: 0 <= MaxContourOverlap && MaxContourOverlap <= 1

▷ **ClusterThreshold** (input_control) ..... real ⇝ *real*
Schwellenwert für das Zusammenfassen initialer Komponenten.
Defaultwert: 0.5
Wertevorschläge: ClusterThreshold ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: 0 <= ClusterThreshold && ClusterThreshold <= 1

Beispiel

```
* Get the model image.
read_image (ModelImage, 'model_image.tif')
* Define the regions for the initial components.
gen_rectangle2 (InitialComponentRegions, 212, 233, 0.62, 167, 29)
gen_rectangle2 (Rectangle2, 298, 363, 1.17, 162, 34)
gen_rectangle2 (Rectangle3, 63, 444, -0.26, 50, 27)
gen_rectangle2 (Rectangle4, 120, 473, 0, 33, 20)
concat_obj (InitialComponentRegions, Rectangle2, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle3, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle4, InitialComponentRegions)
* Get the training images
gen_empty_obj (TrainingImages)
for i := 1 to 4 by 1
    read_image (TrainingImage, 'training_image-'+i$'02'+'.tif')
    concat_obj (TrainingImages, TrainingImage, TrainingImages)
endfor
* Extract the model components and train the relations.
train_model_components (ModelImage, InitialComponentRegions, \
                        TrainingImages, ModelComponents, 22, 60, 30, 0.65, \
                        0, 0, rad(60), 'speed', 'rigidity', 0.2, 0.5, \
                        ComponentTrainingID)
* Find the best value for the parameter ClusterThreshold.
inspect_clustered_components (ModelComponents, ComponentTrainingID, \
                              'rigidity', 0.2, 0.4)
* Adopt the ClusterThreshold into the training result.
cluster_model_components (ModelComponents, ModelComponents, \
                          ComponentTrainingID, 'rigidity', 0.2, 0.4)
* Create the component model based on the training result.
create_trained_component_model (ComponentTrainingID, -rad(30), rad(60), 10, \
                                0.5, 'auto', 'auto', 'none', \
                                'use_polarity', 'false', ComponentModelID, \
                                RootRanking)
```

Ergebnis

Ist der Handle des Trainingsergebnisses gültig, dann liefert inspect_clustered_components den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_model_components`

Nachfolger

`cluster_model_components`

Modul

Matching

```
modify_component_relations ( : : ComponentTrainingID,
    ReferenceComponent, ToleranceComponent, PositionTolerance,
    AngleTolerance : )
```

Verändert die Relationen innerhalb eines Trainingsergebnisses.

`modify_component_relations` verändert die Relationen zwischen den Modellkomponenten innerhalb des Trainingsergebnisses `ComponentTrainingID`. Die Auswahl der zu ändernden Relation(en) erfolgt mit `ReferenceComponent` und `ToleranceComponent`, d.h. die Bewegung der Komponente `ToleranceComponent` relativ zur Komponente `ReferenceComponent` wird verändert. Diese beiden Parameter beschreiben den Index der entsprechenden Komponenten und müssen somit im Bereich von 0 bis n-1 liegen, wenn n die Anzahl der Komponenten darstellt.

Das Maß der Veränderung wird durch die Angabe von Toleranzwerten festgelegt. Durch die Angabe einer Positionstoleranz `PositionTolerance` wird erreicht, dass die Halbachsen des Rechtecks, das die Referenzpunktbewegung beschreibt (siehe `train_model_components`), um `PositionTolerance` Pixel vergrößert werden. Dementsprechend wird durch die Angabe einer Winkeltoleranz `AngleTolerance` erreicht, dass der Winkelbereich, der die Variation der relativen Orientierung beschreibt (siehe `train_model_components`), um `AngleTolerance` in beide Richtungen vergrößert wird. Negative Toleranzwerte führen konsequenter Weise zur Verkleinerung der jeweiligen Relationen. Die Verwendung von `modify_component_relations` ist insbesondere dann sinnvoll, wenn davon ausgegangen werden muss, dass die während des Trainierens verwendeten Trainingsbildern nicht das gesamte Spektrum der relativen Bewegungen abdecken.

Für die Auswahl der zu ändernden Relationen mit Hilfe von `ReferenceComponent` und `ToleranceComponent` sind folgende Möglichkeiten vorgesehen. Für jeden der beiden Parameter können jeweils entweder ein Wert, mehrere Werte, oder *'all'* übergeben werden. Folgende Tabelle veranschaulicht die verschiedenen Möglichkeiten durch Angabe der betroffenen Relation(en) für verschiedene Kombinationen von Parameterwerten. Dabei wird von vier Modellkomponenten (0, 1, 2 und 3) ausgegangen. Werden zum Beispiel `ReferenceComponent` auf *0* und `ToleranceComponent` auf *1* gesetzt, dann wird die Relation (0,1), die die Bewegung der Komponente 1 relativ zur Komponente 0 beschreibt, verändert.

ReferenceComponent	ToleranceComponent	Betroffene Relation(en)
'all'	'all'	(0,1) (0,2) (0,3) (1,0) (1,2) (1,3) (2,0) (2,1) (2,3) (3,0) (3,1) (3,2)
'all'	[1,2]	(0,1) (0,2) (1,2) (2,1) (3,1) (3,2)
[0,1]	'all'	(0,1) (0,2) (0,3) (1,0) (1,2) (1,3)
0	1	(0,1)
0	[1,2]	(0,1) (0,2)
[0,1]	2	(0,2) (1,2)
[0,1,2]	[1,2,3]	(0,1) (1,2) (2,3)

Die Anzahl an übergebenen Toleranzwerten in `PositionTolerance` und `AngleTolerance` muss gleich 1 sein oder mit der Anzahl an betroffenen Relationen übereinstimmen. Im ersten Fall werden alle betroffenen Relationen um den gleichen Wert verändert, wohingegen im zweiten Fall jede Relation individuell durch einen eigenen Wert verändert werden kann.

Parameter

- ▷ **`ComponentTrainingID`** (input_control) . . . . . component_training ⇝ *handle*
  Handle des Trainingsergebnisses.
- ▷ **`ReferenceComponent`** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Modellkomponente(n) relativ zu der die Bewegung(en) verändert werden soll(en).
  Defaultwert: 'all'
  Wertevorschläge: `ReferenceComponent` ∈ {'all', 0, 1, 2, 3, 4, 5, 6}
- ▷ **`ToleranceComponent`** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Modellkomponente(n), deren relative Bewegung(en) verändert werden soll(en).
  Defaultwert: 'all'
  Wertevorschläge: `ToleranceComponent` ∈ {'all', 0, 1, 2, 3, 4, 5, 6}
- ▷ **`PositionTolerance`** (input_control) . . . . . real(-array) ⇝ *real*
  Veränderung der Positionsrelation in Pixel.
  Wertevorschläge: `PositionTolerance` ∈ {1.0, 2.0, 3.0, 4.0, 5.0, 10.0, 20.0, 30.0}
- ▷ **`AngleTolerance`** (input_control) . . . . . angle.rad(-array) ⇝ *real*
  Veränderung der Winkelrelation im Bogenmaß.
  Wertevorschläge: `AngleTolerance` ∈ {0.017, 0.035, 0.052, 0.070, 0.087, 0.175, 0.349}

Ergebnis

Ist der Handle des Trainingsergebnisses gültig, dann liefert `modify_component_relations` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ComponentTrainingID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`train_model_components`

Nachfolger

`create_trained_component_model`

Modul

Matching

read_component_model (: : FileName : ComponentModelID)

Einlesen eines Komponentenmodells von Datei.

`read_component_model` liest ein Komponentenmodell, das mit `write_component_model` geschrieben wurde, aus der Datei `FileName` ein und gibt es in `ComponentModelID` zurück. Die Dateiendung für das Komponentenmodell ist in HALCON 'cbm'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.cbm`

▷ **ComponentModelID** (output_control) . . . . . component_model ⇝ *handle*
Handle des Komponentenmodells.

Ergebnis

Ist der Dateiname korrekt, dann liefert `read_component_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`find_component_model`

Modul

Matching

read_training_components (: : FileName : ComponentTrainingID)

Einlesen eines Komponenten-Trainingsergebnisses von Datei.

`read_training_components` liest ein Komponenten-Trainingsergebnis, das mit `write_training_components` geschrieben wurde, aus der Datei `FileName` ein und gibt es in `ComponentTrainingID` zurück. Die Dateiendung für das Komponenten-Trainingsergebnis ist in HALCON 'ct'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.ct`

▷ **ComponentTrainingID** (output_control) . . . . . component_training ⇝ *handle*
Handle des Trainingsergebnisses.

Ergebnis

Ist der Dateiname korrekt, dann liefert `read_training_components` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`create_trained_component_model`

Siehe auch

`train_model_components`, `clear_training_components`

Modul

Matching

serialize_component_model (
: : ComponentModelID : SerializedItemHandle)

Serialisieren eines Komponentenmodells.

`serialize_component_model` serialisiert die Daten eines Komponentenmodells (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_component_model` in eine Datei geschrieben werden. Das Komponentenmodell wird in dem Handle `ComponentModelID` übergeben. Das serialisierte Komponentenmodell wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_component_model` wieder deserialisiert werden.

Parameter

▷ **ComponentModelID** (input_control) . . . . . component_model ⇝ *handle*
Handle des Komponentenmodells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_component_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_component_model`, `create_trained_component_model`

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_component_model

Modul

Matching

serialize_training_components (: : ComponentTrainingID : SerializedItemHandle)

Serialisieren eines Komponenten-Trainingsergebnisses.

serialize_training_components serialisiert die Daten eines Komponenten-Trainingsergebnisses (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_training_components in eine Datei geschrieben werden. Das Komponenten-Trainingsergebnis wird in dem Handle ComponentTrainingID übergeben. Das serialisierte Komponenten-Trainingsergebnis wird in dem Handle SerializedItemHandle zurückgegeben. und kann mit deserialize_training_components wieder deserialisiert werden.

Parameter

- ▷ **ComponentTrainingID** (input_control) ..... component_training ⇝ *handle*
  Handle des Trainingsergebnisses.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_training_components den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

train_model_components

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_training_components

Modul

Matching

train_model_components (ModelImage, InitialComponents, TrainingImages : ModelComponents : ContrastLow, ContrastHigh, MinSize, MinScore, SearchRowTol, SearchColumnTol, SearchAngleTol, TrainingEmphasis, AmbiguityCriterion, MaxContourOverlap, ClusterThreshold : ComponentTrainingID)

Trainieren von Komponenten und Relationen für das komponentenbasierte Matching.

train_model_components ermittelt die endgültigen (starren) Modellkomponenten und trainiert deren gegenseitige Relationen, d.h. deren relative Bewegungen, auf der Grundlage der initialen Komponenten anhand mehrerer Trainingsbilder. Das Ergebnis des Trainings wird in ComponentTrainingID als Handle zurückgegeben. Das Trainingsergebnis kann anschließend dazu verwendet werden, das eigentliche Komponentenmodell mit create_trained_component_model aufzubauen.

train_model_components sollte in den Fällen verwendet werden, in denen die Relationen der Komponenten nicht bekannt sind und daher automatisch trainiert werden sollen. Sind die Relationen dagegen bekannt, kann auf den Einsatz von train_model_components verzichtet werden und das Komponentenmodell direkt mit create_component_model generiert werden.

Sind die initialen Komponenten automatisch mit gen_initial_components erzeugt worden, so enthält InitialComponents die Konturregionen der initialen Komponenten. Sollen die initialen Komponenten dagegen von Hand vorgegeben werden, so können diese in InitialComponents direkt übergeben werden. Dabei dürfen allerdings keine Konturregionen, sondern die initialen Komponenten umschließende Regionen übergeben werden. Die übergebenen (Kontur-)Regionen beziehen sich auf das Modellbild ModelImage. Wurden die initialen Komponenten mit Hilfe von gen_initial_components bestimmt, sollte das übergebene Modellbild folglich mit dem Modellbild in gen_initial_components übereinstimmen. In jedem Fall ist zu beachten, dass jede initiale Komponente Bestandteil von höchstens einer starren Modellkomponente ist. Der Grund hierfür ist, dass während des Trainings initiale Komponenten zwar bei Bedarf zu starren Modellkomponenten zusammengefasst werden (siehe unten), nicht jedoch in kleinere Einheiten zerlegt und auf verschiedene Modellkomponenten aufgeteilt werden können.

Folgendes Vorgehen wird in train_model_components zum Trainieren angewendet: Im ersten Schritt werden die initialen Komponenten in allen Trainingsbildern gesucht. Es kann vorkommen, dass eine initiale Komponente in einem Trainingsbild mehrfach gefunden wird. Daher werden im zweiten Schritt die daraus resultierenden Mehrdeutigkeiten gelöst, d.h. die wahrscheinlichste Lage jeder initialen Komponente wird ermittelt. Nach dem Lösen der Mehrdeutigkeiten bleibt demnach in jedem Trainingsbild für jede initiale Komponente höchstens eine gefundene Instanz übrig. Im nächsten Schritt werden die (nun eindeutigen) Lagen der initialen Komponenten analysiert. Diejenigen initialen Komponenten, die sich nicht relativ zueinander bewegen, werden zu den endgültigen starren Modellkomponenten zusammengefasst. Schließlich werden im letzten Schritt die Relationen zwischen den Modellkomponenten ermittelt indem ihre relativen Bewegungen über die Sequenz an Trainingsbildern analysiert werden. Die mit den beschriebenen Schritten verbundenen Parameter werden im Folgenden erläutert.

Das Training erfolgt auf der Grundlage der in TrainingImages übergebenen Trainingsbilder. Die Trainingsbilder dürfen jeweils höchstens eine Instanz des zusammengesetzten Objektes zeigen und sollten nach Möglichkeit die relativen Bewegungen aller Modellkomponenten in vollem Umfang beinhalten. Soll z.B. das Komponentenmodell eines Ein-Aus-Schalters trainiert werden, so genügt ein Trainingsbild, in dem der Schalter im ausgeschalten Zustand zu sehen ist für den Fall, dass der Schalter im Modellbild im eingeschalteten Zustand vorliegt, bzw. umgekehrt.

Das Prinzip des Trainings ist es, die initialen Komponenten in den Trainingsbildern wiederzufinden und deren Lage (Position und Orientierung) zu analysieren. Dazu wird für jede initiale Komponente ein Formmodell erzeugt (siehe create_shape_model), das dann dazu genutzt wird, die Lage der initialen Komponente in den Trainingsbildern zu bestimmen (siehe find_shape_model). Je nachdem welcher Modus mit set_system ('pregenerate_shape_models',...) eingestellt ist, wird das komplette Formmodell vorab generiert oder erst während der Suche online erzeugt. Dieser Modus hat sowohl Einfluss auf die Laufzeit als auch auf die Robustheit des Trainings. Weiterhin ist zu beachten, dass bei Verwendung von einkanaligen Bildern sowohl in ModelImage als auch in TrainingImages intern die Metrik *'use_polarity'* für create_shape_model verwendet wird, während bei mehrkanaligen Bildern in ModelImage oder in TrainingImages intern die Metrik *'ignore_color_polarity'* verwendet wird. Schließlich ist zu beachten, dass die Anzahl der Kanäle in ModelImage und TrainingImages zwar unterschiedlich sein darf, z.B. um die Modellgenerierung aus synthetischen Bildern zu erlauben, dass aber die Anzahl der Kanäle für alle Bilder in TrainingImages gleich sein muss. Weitere Informationen können bei create_shape_model nachgelesen werden. Das Generieren der Formmodelle kann mit Hilfe der Parameter ContrastLow, ContrastHigh und MinSize beeinflusst werden. Diese Parameter erfüllen dieselbe Funktion wie in gen_initial_components und können durch die Angabe von *'auto'* auch automatisch bestimmt werden: Sollen beide Hysterese-Schwellen automatisch bestimmt werden, so müssen sowohl ContrastLow als auch ContrastHigh jeweils auf *'auto'* gesetzt werden. Soll dagegen lediglich ein Kontrastwert automatisch bestimmt werden, so muss ContrastLow auf *'auto'* und ContrastHigh auf einen beliebigen Wert ungleich *'auto'* gesetzt werden. Wurden die initialen Komponenten automatisch mit gen_initial_components erzeugt, so sollten die Parameter ContrastLow, ContrastHigh und MinSize mit den selben Werten besetzt werden wie in gen_initial_components.

Zur Steuerung der Suche nach den initialen Komponenten dienen die Parameter MinScore, SearchRowTol, SearchColumnTol, SearchAngleTol und TrainingEmphasis. Der Parameter MinScore legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz der initialen Komponente im Trainingsbild angesehen wird. Je größer der Wert von MinScore gewählt werden kann, desto schneller verläuft das Training. Falls erwartet werden kann, dass die initialen Komponenten in den Trainingsbildern niemals verdeckt sind, kann MinScore auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden (siehe auch find_shape_model).

Im Normalfall werden die Komponenten nur an den Stellen der Trainingsbilder gesucht, an denen die jeweilige Komponente vollständig in das Bild passt. Das bedeutet, dass eine Komponente nicht gefunden werden kann, wenn sie aus dem Bild herausragt, selbst wenn sie eine Bewertung größer als MinScore erreichen würde. Dieses

Verhalten kann mit set_system('border_shape_models','true') umgestellt werden, so dass auch Komponenten gefunden werden, die aus dem Bild herausragen, falls sie eine Bewertung größer als MinScore erreichen. Dabei werden Punkte außerhalb des Bildes als verdeckt angesehen, d.h. sie verringern die Bewertung. Es ist zu beachten, dass dieser Modus die Laufzeit des Trainings erhöht.

Bei einer hohen Anzahl an initialen Komponenten und vielen Trainingsbildern kann das Trainieren unter Umständen sehr lange (bis zu mehreren Minuten) dauern. Um das Trainieren zu beschleunigen besteht die Möglichkeit, den Suchbereich in den Trainingsbildern für die einzelnen initialen Komponenten einzuschränken. Dabei gilt die Lage der initialen Komponenten im Modellbild als Referenzlage. Die Parameter SearchRowTol und SearchColumnTol geben an, in welchem Toleranzbereich relativ zur Referenzposition gesucht werden soll. Wenn z.B. die Position einer initialen Komponente im Modellbild (100,200) ist und für SearchRowTol = *20* und für SearchColumnTol = *10* gewählt wird, dann wird diese initiale Komponente in den Trainingsbildern lediglich innerhalb des achsenparallelen Rechtecks gesucht, dessen linke obere Ecke bei (80,190) und dessen rechte untere Ecke bei (120,210) liegt. Das gleiche gilt für den Winkelsuchbereich, der durch die Angabe der Winkeltoleranz SearchAngleTol auf ±SearchAngleTol eingeschränkt werden kann. Damit ist es möglich durch geschickte Aufnahme der Trainingsbilder den Rechenaufwand während des Trainierens deutlich zu reduzieren. Wird für einen dieser drei Parameter der Wert *-1* übergeben, erfolgt in der entsprechenden Dimension keine Suchraumeinschränkung.

Die Eingabeparameter ContrastLow, ContrastHigh, MinSize, MinScore, SearchRowTol, SearchColumnTol, and SearchAngleTol müssen entweder genau ein Element enthalten, oder dieselbe Anzahl wie initiale Komponenten in InitialComponents. Im ersten Fall wird der Wert des Eingabeparameters für alle initiale Komponenten gleich verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für die entsprechende initiale Komponente in InitialComponents verwendet.

Der Parameter TrainingEmphasis bietet eine weitere Möglichkeit, auf die Geschwindigkeit des Trainingsvorgangs Einfluss zu nehmen und gleichzeitig dessen Robustheit zu steuern. Wird TrainingEmphasis auf *'speed'* gesetzt, so verläuft das Training zwar vergleichsweise schnell, es kann allerdings in einigen Fällen dazu führen, dass einzelne initiale Komponenten in den Trainingsbildern nicht oder in falscher Lage gefunden werden. Das würde zu einer fehlerhaften Berechnung der starren Modellkomponenten und deren Relationen führen. Die Lage der gefundenen initialen Komponenten in den einzelnen Trainingsbildern kann mit Hilfe von get_training_components kontrolliert werden. Bei auftretenden Fehlzuordnungen sollte dann TrainingEmphasis auf *'reliability'* gesetzt und das Training erneut gestartet werden. Dadurch wird eine höhere Robustheit gegenüber Fehlzuordnungen erreicht, allerdings auf Kosten einer längeren Rechenzeit.

Bei der Bestimmung der Lage kann es außerdem zu Mehrdeutigkeiten kommen, falls die initialen Komponenten Rotationssymmetrien aufweisen oder wenn mehrere initiale Komponenten identisch oder sich sehr ähnlich sind. Zur Lösung dieser Mehrdeutigkeiten wird jeweils die wahrscheinlichste Lage für jede initiale Komponenten in jedem Trainingsbild ermittelt. Dazu müssen die einzelnen mehrdeutigen Lagen bewertet werden. Die Lage einer initialen Komponente im Trainingsbild wird dann gut bewertet, wenn die relative Lage zu den übrigen initialen Komponenten möglichst gut mit der entsprechenden relativen Lage im Modellbild übereinstimmt. Die Bewertung der Übereinstimmungen kann mit AmbiguityCriterion gewählt werden. In fast allen Fällen liefert *'rigidity'* die besten Ergebnisse. Dabei wird von der Starrheit des zusammengesetzten Objektes ausgegangen. Je mehr die Starrheit des zusammengesetzten Objektes durch die Lage der initialen Komponente verletzt wird, desto geringer fällt deren Bewertung aus. Bei *'distance'* wird nur die Entfernung zwischen den initialen Komponenten als Maß herangezogen. In diesem Fall erhält die Lage einer initialen Komponente eine gute Bewertung wenn ihre Distanzen zu den übrigen Komponenten mit den entsprechenden Distanzen im Modellbild gut übereinstimmen. Dementsprechend ist bei *'orientation'* nur die relative Orientierung für die Bewertung ausschlaggebend. Schlussendlich kann die gleichzeitige Berücksichtigung von Entfernung und relativer Orientierung mit *'distance_orientation'* erreicht werden. Im Gegensatz zu *'rigidity'* wird bei *'distance_orientation'* die relative Lage der initialen Komponenten vernachlässigt.

Die Lösung der Mehrdeutigkeiten kann auch durch den Parameter MaxContourOverlap beeinflusst werden. Dieser Parameter beschreibt das Ausmaß, zu dem sich die Konturen zweier Matches initialer Komponenten überlappen dürfen. Angenommen, es sollen, z.B. zwei initiale Komponenten 'I' und 'T' in einem Trainingsbild, das den Schriftzug 'IT' enthält, gefunden werden. Dann wird zwar die initiale Komponente 'T' in ihrer richtigen Lage gefunden, aufgrund der Ähnlichkeiten der beiden initialen Komponenten wird die initiale Komponente 'I' allerdings sowohl in ihrer richtigen Lage ('I') als auch in der falschen Lage an der Position des 'T' gefunden. Durch geeignete Wahl von MaxContourOverlap ist es möglich, den falschen Match zu verwerfen: Sollen überlappende Matches toleriert werden, dann sollte MaxContourOverlap auf *1* gesetzt werden. Sollen überlappende Matches vollkommen ausgeschlossen werden, dann sollte MaxContourOverlap auf *0* gesetzt werden. Durch die Wahl eines Wertes zwischen *0* und *1* kann der maximale Prozentsatz an überlappenden Konturpixeln stufenlos eingestellt werden.

Aufgrund der Lagen der initialen Komponenten im Modellbild und in den Trainingsbildern wird dann zunächst entschieden, welche initialen Komponenten zu starren Modellkomponenten zusammengefasst werden können. Zwei initiale Komponenten können genau dann zusammengefasst werden, wenn sie über alle Bilder keine relative Bewegung zeigen. Würde im Falle des Schalters das Trainingsbild den gleichen Schalterzustand zeigen wie das Modellbild, so würde der Algorithmus die entsprechenden initialen Komponenten zusammenfassen in der Annahme, es handele sich um eine starre Modellkomponente. Das Maß, in dem initiale Komponenten zusammengefasst werden, kann über den Parameter `ClusterThreshold` gesteuert werden. Dieser Schwellenwert basiert auf der Wahrscheinlichkeit, dass zwei initiale Komponenten zur gleichen starren Modellkomponente gehören. `ClusterThreshold` gibt somit die minimale Wahrscheinlichkeit an, die zwei initiale Komponenten aufweisen müssen, damit sie zusammengefasst werden. Da es sich um Wahrscheinlichkeiten handelt, muss der Schwellenwert im Bereich von *0* bis *1* liegen. Je größer der Cluster-Schwellenwert gewählt wird, desto weniger Zusammenschlüsse finden statt. Für einen Schwellenwert von *0* werden all initialen Komponenten zu einer starren Komponente zusammengefasst, für einen Schwellenwert von *1* finden keine Zusammenschlüsse statt und jede initiale Komponente wird als starre Modellkomponente übernommen.

Die endgültigen starren Modellkomponenten werden in `ModelComponents` zurückgeliefert. In nachfolgenden Operatoren wird der Index einer Komponenten in `ModelComponents` verwendet, um die Modellkomponente zu bezeichnen. Die Lage der gefundenen Modellkomponenten in den Trainingsbildern kann mit `get_training_components` kontrolliert werden.

Nach der Bestimmung der Modellkomponenten werden deren relative Bewegungen analysiert. Dabei wird für jedes Komponentenpaar die Bewegung der zweiten Komponente relativ zur ersten bestimmt. Dazu werden die Komponenten zunächst auf ihren Referenzpunkt bezogen. Der Referenzpunkt einer Komponente ergibt sich aus dem Schwerpunkt seiner Konturregion, die in `ModelComponents` zurückgeliefert wird und kann mit `area_center` berechnet werden. Die relative Bewegung wird dann durch das kleinste umschließende Rechteck beliebiger Orientierung der relativen Referenzpunktbewegung und durch den kleinsten umschließenden Winkelbereich der relativen Orientierung der zweiten Komponente über alle Bilder repräsentiert. Die ermittelten Relationen können mit `get_component_relations` inspiziert werden.

Parameter

- ▷ **`ModelImage`** (input_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2
  Eingabebild, aus dem die Formmodelle für die initialen Komponenten generiert werden sollen.
- ▷ **`InitialComponents`** (input_object) . . . . . region-array ⇝ *object*
  Konturregionen oder umschließende Regionen der initiale Komponenten.
- ▷ **`TrainingImages`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Zum Trainieren der Modellkomponenten verwendete Trainingsbilder.
- ▷ **`ModelComponents`** (output_object) . . . . . region(-array) ⇝ *object*
  Konturregionen der starren Modellkomponenten.
- ▷ **`ContrastLow`** (input_control) . . . . . integer(-array) ⇝ *integer* / string
  Unterer Hystereseschwellenwert für den Kontrast der initialen Komponenten im Bild.
  Defaultwert: 'auto'
  Wertevorschläge: `ContrastLow` ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}
  Restriktion: ContrastLow > 0
- ▷ **`ContrastHigh`** (input_control) . . . . . integer(-array) ⇝ *integer* / string
  Oberer Hystereseschwellenwert für den Kontrast der initialen Komponenten im Bild.
  Defaultwert: 'auto'
  Wertevorschläge: `ContrastHigh` ∈ {'auto', 10, 20, 30, 40, 60, 80, 100, 120, 140, 160}
  Restriktion: ContrastHigh > 0 && ContrastHigh >= ContrastLow
- ▷ **`MinSize`** (input_control) . . . . . integer(-array) ⇝ *integer* / string
  Mindestgröße zusammenhängender Konturregionen.
  Defaultwert: 'auto'
  Wertevorschläge: `MinSize` ∈ {'auto', 0, 5, 10, 20, 30, 40}
  Restriktion: MinSize >= 0
- ▷ **`MinScore`** (input_control) . . . . . real(-array) ⇝ *real*
  Minimale Bewertung der zu findenden Instanzen der initialen Komponenten
  Defaultwert: 0.5
  Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
  Restriktion: 0 <= MinScore && MinScore <= 1

▷ **SearchRowTol** (input_control) . . . . . integer(-array) ⇝ *integer*
Suchtoleranz in Zeilenrichtung.
Defaultwert: -1
Wertevorschläge: SearchRowTol ∈ {0, 10, 20, 30, 50, 100}
Restriktion: SearchRowTol == -1 || SearchColumnTol >= 0

▷ **SearchColumnTol** (input_control) . . . . . integer(-array) ⇝ *integer*
Suchtoleranz in Spaltenrichtung.
Defaultwert: -1
Wertevorschläge: SearchColumnTol ∈ {0, 10, 20, 30, 50, 100}
Restriktion: SearchColumnTol == -1 || SearchColumnTol >= 0

▷ **SearchAngleTol** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Winkel-Suchtoleranz.
Defaultwert: -1
Wertevorschläge: SearchAngleTol ∈ {0.0, 0.17, 0.39, 0.78, 1.57}
Restriktion: SearchAngleTol == -1 || SearchAngleTol >= 0

▷ **TrainingEmphasis** (input_control) . . . . . string ⇝ *string*
Entscheidung, ob das Trainieren auf dem Grundsatz einer schnellen Berechnung oder einen hohen Robustheit basiert.
Defaultwert: 'speed'
Werteliste: TrainingEmphasis ∈ {'speed', 'reliability'}

▷ **AmbiguityCriterion** (input_control) . . . . . string ⇝ *string*
Kriterium zur Lösung mehrdeutiger Matches der initialen Komponenten in einem Trainingsbild.
Defaultwert: 'rigidity'
Werteliste: AmbiguityCriterion ∈ {'distance', 'orientation', 'distance_orientation', 'rigidity'}

▷ **MaxContourOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Konturüberlappung gefundener initialer Komponenten in einem Trainingsbild.
Defaultwert: 0.2
Wertevorschläge: MaxContourOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.05
Restriktion: 0 <= MaxContourOverlap && MaxContourOverlap <= 1

▷ **ClusterThreshold** (input_control) . . . . . real ⇝ *real*
Schwellenwert für das Zusammenfassen initialer Komponenten.
Defaultwert: 0.5
Wertevorschläge: ClusterThreshold ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: 0 <= ClusterThreshold && ClusterThreshold <= 1

▷ **ComponentTrainingID** (output_control) . . . . . component_training ⇝ *handle*
Handle des Trainingsergebnisses.

Beispiel

```
* Get the model image.
read_image (ModelImage, 'model_image.tif')
* Define the regions for the initial components.
gen_rectangle2 (InitialComponentRegions, 212, 233, 0.62, 167, 29)
gen_rectangle2 (Rectangle2, 298, 363, 1.17, 162, 34)
gen_rectangle2 (Rectangle3, 63, 444, -0.26, 50, 27)
gen_rectangle2 (Rectangle4, 120, 473, 0, 33, 20)
concat_obj (InitialComponentRegions, Rectangle2, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle3, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle4, InitialComponentRegions)
* Get the training images.
gen_empty_obj (TrainingImages)
for i := 1 to 4 by 1
    read_image (TrainingImage, 'training_image-'+i+'.tif')
    concat_obj (TrainingImages, TrainingImage, TrainingImages)
endfor
* Extract the model components and train the relations.
train_model_components (ModelImage, InitialComponentRegions, \
```

```
TrainingImages, ModelComponents, 22, 60, 30, 0.6, \
0, 0, rad(60), 'speed', 'rigidity', 0.2, 0.4, \
ComponentTrainingID)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `train_model_components` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder oder Eingaberegionen vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`gen_initial_components`

Nachfolger

`inspect_clustered_components`, `cluster_model_components`, `modify_component_relations`, `write_training_components`, `get_training_components`, `get_component_relations`, `create_trained_component_model`, `clear_training_components`

Siehe auch

`create_shape_model`, `find_shape_model`

Modul

Matching

`write_component_model` (: : ComponentModelID, FileName :)

Schreiben eines Komponentenmodells auf Datei.

`write_component_model` schreibt ein Komponentenmodell `ComponentModelID` in die Datei `FileName`. Das Modell kann mit `read_component_model` wieder eingelesen werden. Die Dateiendung für das Komponentenmodell ist in HALCON 'cbm'.

Parameter

▷ **`ComponentModelID`** (input_control) ..... component_model ⇝ *handle*
Handle des Komponentenmodells.

▷ **`FileName`** (input_control) ..... filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.cbm`

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert `write_component_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_component_model`, `create_trained_component_model`

Modul

Matching

write_training_components (: : ComponentTrainingID, FileName :)

Schreiben eines Komponenten-Trainingsergebnisses auf Datei.

`write_training_components` schreibt das Komponenten-Trainingsergebnis `ComponentTrainingID` in die Datei `FileName`. Das Trainingsergebnis kann mit `read_training_components` wieder eingelesen werden Die Dateiendung für das Komponenten-Trainingsergebnis ist in HALCON 'ct'.

Parameter

- ▷ **ComponentTrainingID** (input_control) . . . . . component_training ⇝ *handle*
  Handle des Trainingsergebnisses.
- ▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
  Name der Datei.
  Dateiendung: `.ct`

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert `write_training_components` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`train_model_components`

Modul

Matching

17.5 Korrelationsbasiert

clear_ncc_model (: : ModelID :)

Freigabe des Speichers eines NCC-Modells.

`clear_ncc_model` gibt den Speicher eines NCC-Modells, das mit `create_ncc_model` angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `ModelID` ist nach dem Aufruf ungültig.

Parameter

- ▷ **ModelID** (input_control) . . . . . ncc_model ⇝ *handle*
  Handle des Modells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `clear_ncc_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ncc_model`, `read_ncc_model`, `write_ncc_model`, `find_ncc_model`, `find_ncc_models`

Modul

Matching

`create_ncc_model` (`Template` : : `NumLevels`, `AngleStart`, `AngleExtent`, `AngleStep`, `Metric` : `ModelID`)

Vorbereiten eines NCC-Modells für das Matching.

`create_ncc_model` bereitet ein Muster, das als Bild `Template` übergeben wird, als NCC-Modell für das Matching mittels normalisierter Kreuzkorrelation (normalized cross correlation, NCC) vor. Die ROI des Modells wird als der Definitionsbereich von `Template` übergeben.

Das Modell wird auf mehreren Pyramidenebenen in mehreren Rotationen auf jeder Ebene generiert und im Speicher abgelegt. Der Ausgabeparameter `ModelID` ist ein Handle für dieses Modell, der in nachfolgenden Aufrufen von `find_ncc_model` verwendet wird.

Die Anzahl der Pyramidenebenen wird mit `NumLevels` festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von `NumLevels` ist aber darauf zu achten, dass das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens acht). Dies kann anhand des Definitionsbereiches der Bilder in der Ausgabe von `gen_gauss_pyramid` überprüft werden. Falls nicht genügend Modellpunkte erzeugt werden, wird die Anzahl der Pyramidenstufen intern solange reduziert, bis auf der obersten Pyramidenstufe genügend Modellpunkte vorhanden sind. Falls durch diese Anpassung ein Modell mit keiner Pyramidenstufe entstehen würde, d.h. falls schon auf der untersten Pyramidenstufe zu wenige Modellpunkte vorhanden sind, liefert `create_ncc_model` eine Fehlermeldung zurück. Falls für `NumLevels` *'auto'* oder *0* übergeben wird, wählt `create_ncc_model` die Anzahl der Pyramidenstufen automatisch. Die automatisch gewählte Anzahl von Pyramidenstufen kann mit `get_ncc_model_params` abgefragt werden. In seltenen Fällen kann es vorkommen, dass `create_ncc_model` die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, dass das Modell im Bild nicht erkannt wird oder dass sehr niedrige Parameter für `MinScore` in `find_ncc_model` selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in `find_ncc_model` kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen durch Inspektion der Ausgabebilder von `gen_gauss_pyramid` gewählt werden. Dabei sollte `Mode` = *'constant'* und `Scale` = *0.5* verwendet werden.

Die Parameter `AngleStart` und `AngleExtent` legen den Winkelbereich für die möglichen Rotationen des Modells im Bild fest. Das Modell kann also mit `find_ncc_model` nur in diesem Winkelbereich gefunden werden. Der Parameter `AngleStep` gibt die Schrittweite der Winkel in dem gewählten Winkelbereich an. Falls bei `find_ncc_model` keine Subpixelgenauigkeit spezifiziert wird, gibt `AngleStep` also die erreichbare Winkelgenauigkeit an. `AngleStep` sollte aufgrund der Größe des Objektes gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte `AngleStep` für kleinere Modelle größer gewählt werden. Falls `AngleExtent` kein ganzzahliges Vielfaches von `AngleStep` ist, wird `AngleStep` entsprechend angepasst. Um eine Abtastung des Winkelbereiches der möglichen Rotationen zu erreichen, die unabhängig von `AngleStart` ist, wird der Winkelbereich der möglichen Rotationen folgendermaßen angepasst: Wenn kein positiver ganzzahliger Wert n existiert, für den `AngleStart` plus n mal `AngleStep` genau Null ergibt, wird `AngleStart` um höchstens `AngleStep` verringert, und `AngleExtent` wird um `AngleStep` vergrößert.

Das Modell wird in dem gewählten Winkelbereich vorab generiert und im Speicher abgelegt. Der Speicherbedarf zur Speicherung des Modells ist also proportional zur Anzahl der Winkelschritte und zur Anzahl der Punkte im Modell. Wenn also AngleStep zu klein bzw. AngleExtent zu groß gewählt wird, kann es vorkommen, dass das Modell nicht mehr in den (virtuellen) Speicher passt. In diesem Fall muss entweder AngleStep größer oder AngleExtent kleiner gewählt werden. In jedem Fall ist es aus Laufzeitgründen vorteilhaft, wenn das Modell komplett in den Hauptspeicher passt und somit ein Paging durch das Betriebssystem vermieden werden kann. Da die Möglichkeit zur subpixelgenauen Winkelbestimmung in find_ncc_model gegeben ist, kann bei Modellen mit einem Durchmesser von ca. 200 Pixeln AngleStep ≥ 1 gewählt werden. Falls AngleStep $= 'auto'$ oder *0* übergeben wird, wählt create_ncc_model automatisch eine Schrittweite basierend auf der Größe des Modells aus. Die automatisch gewählte Schrittweite kann mit get_ncc_model_params abgefragt werden.

Der Parameter Metric legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird. Falls Metric = *'use_polarity'*, muss das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist. Falls Metric = *'ignore_global_polarity'*, wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_ncc_model erhöht sich in diesem Fall geringfügig.

Als Ursprung (Referenzpunkt) des Modells wird der Schwerpunkt der Region des Modellbildes Template verwendet. Falls ein anderer Ursprung gewünscht wird, kann dieser mit set_ncc_model_origin gesetzt werden.

Parameter

▷ **Template** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Eingabebild, dessen Definitionsbereich zum Aufbau des Modells verwendet wird.

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
Maximale Anzahl von Pyramidenebenen.
Defaultwert: 'auto'
Werteliste: NumLevels ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinste auftretende Rotation des Musters.
Defaultwert: -0.39
Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert: 0.79
Wertevorschläge: AngleExtent ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
Restriktion: AngleExtent >= 0

▷ **AngleStep** (input_control) . . . . . angle.rad ⇝ *real* / string
Schrittweite der Winkel (Auflösung).
Defaultwert: 'auto'
Wertevorschläge: AngleStep ∈ {'auto', 0.0, 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
Restriktion: AngleStep >= 0 && AngleStep <= pi / 16

▷ **Metric** (input_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.
Defaultwert: 'use_polarity'
Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity'}

▷ **ModelID** (output_control) . . . . . ncc_model ⇝ *handle*
Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_ncc_model den Wert TRUE. Wenn der Parameter NumLevels so gewählt wurde, dass das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8506 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

draw_region, reduce_domain, threshold

Nachfolger

find_ncc_model, get_ncc_model_params, clear_ncc_model, write_ncc_model, set_ncc_model_origin, set_ncc_model_param, find_ncc_models

Alternativen

create_shape_model, create_scaled_shape_model, create_aniso_shape_model

Modul

Matching

deserialize_ncc_model (: : SerializedItemHandle : ModelID)

Deserialisieren eines NCC-Modells.

deserialize_ncc_model deserialisiert ein NCC-Modell, welches mit serialize_ncc_model serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Das serialisierte NCC-Modell wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten NCC-Modell mit dem Handle ModelID gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **ModelID** (output_control) . . . . . ncc_model ⇝ *handle*
  Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_ncc_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_ncc_model

Nachfolger

find_ncc_model

Siehe auch

create_ncc_model, clear_ncc_model

Modul

Matching

determine_ncc_model_params (Template : : NumLevels, AngleStart, AngleExtent, Metric, Parameters : ParameterName, ParameterValue)

Berechnung der Parameter eines NCC-Modells.

determine_ncc_model_params berechnet bestimmte Parameter eines NCC-Modells automatisch aus dem Modellbild Template. Die zu berechnenden Parameter werden in Parameters angegeben. Mit determine_ncc_model_params lassen sich dieselben Parameter automatisch bestimmen, die auch mit

create_ncc_model durch Setzen des jeweiligen Parameters auf *'auto'* automatisch bestimmt werden können: die Anzahl der Pyramidenebenen des Modells (Parameters = *'num_levels'*) und/oder die Winkelschrittweite (Parameters = *'angle_step'*). Es kann eine beliebige Kombination dieser Parameter bestimmt werden, indem ein Tupel von Werten in Parameters übergeben wird. Falls alle Parameter automatisch bestimmt werden sollen, kann zur Vereinfachung auch der Wert *'all'* übergeben werden.

determine_ncc_model_params kann hauptsächlich dazu verwendet werden, die obigen Parameter automatisch zu bestimmen, bevor das Modell erzeugt wird. Dies ist vor allem in interaktiven Systemen nützlich, in denen dem Benutzer ein Vorschlag für diese Werte präsentiert wird, die der Benutzer aber modifizieren kann.

Die berechneten Parameter werden in ParameterName und ParameterValue als Name des Parameters und als jeweiliger Wert zurückgeliefert. Dabei haben die Parameter in ParameterName dieselben Namen wie in Parameters, wobei natürlich der Wert *'all'* durch die entsprechenden Parameternamen ersetzt wird.

Die Eingabeparameter (NumLevels, AngleStart, AngleExtent und Metric) haben dieselbe Bedeutung wie bei create_ncc_model. Die Beschreibung der Parameter kann dort nachgelesen werden. Diese Parameter werden von determine_ncc_model_params dazu verwendet, die zu bestimmenden Parameter auf dieselbe Art zu berechnen wie in create_ncc_model.

Zu beachten ist, dass determine_ncc_model_params den Eingabeparameter NumLevels auch automatisch bestimmen kann. Falls dieser Parameter nicht automatisch bestimmt werden soll, d.h. der Name wird nicht in Parameters übergeben, muss für den Parameter ein zulässiger Wert übergeben werden. Insbesondere darf der Parameter nicht auf *'auto'* gesetzt sein. Soll die Anzahl an Pyramidenebenen automatisch bestimmt werden, d.h. Parameters enthält den Wert *'num_levels'*, dann kann HALCON zwar einen geeigneten Wert ermitteln, zusätzlich besteht jedoch die Möglichkeit, eine obere Grenze anzugeben:

Falls die maximale Anzahl der Pyramidenstufen vorgegeben werden soll, kann der Eingabeparameter NumLevels auf den betreffenden Wert gesetzt werden. Falls in diesem Fall Parameters den Wert *'num_levels'* enthält, kann der berechnete Wert der Anzahl der Pyramidenebenen höchstens NumLevels betragen. Falls NumLevels = *'auto'* oder *0* gesetzt wird, wird die Anzahl der Pyramidenebenen ohne Beschränkung so hoch wie möglich bestimmt.

Parameter

- ▷ **Template** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild, dessen Definitionsbereich für das Modell verwendet wird.
- ▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer* / string
  Maximale Anzahl von Pyramidenebenen.
  Defaultwert: 'auto'
  Werteliste: NumLevels ∈ {'auto', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
  Kleinste auftretende Rotation des Modells.
  Defaultwert: -0.39
  Wertevorschläge: AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **AngleExtent** (input_control) . . . . . angle.rad ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Defaultwert: 0.79
  Wertevorschläge: AngleExtent ∈ {6.29, 3.14, 1.57, 0.79, 0.39}
  Restriktion: AngleExtent >= 0
- ▷ **Metric** (input_control) . . . . . string ⇝ *string*
  Art der zum Matchen verwendeten Metrik.
  Defaultwert: 'use_polarity'
  Werteliste: Metric ∈ {'use_polarity', 'ignore_global_polarity'}
- ▷ **Parameters** (input_control) . . . . . string(-array) ⇝ *string*
  Zu berechnende Parameter.
  Defaultwert: 'all'
  Werteliste: Parameters ∈ {'all', 'num_levels', 'angle_step'}
- ▷ **ParameterName** (output_control) . . . . . string-array ⇝ *string*
  Name des berechneten Parameters.
- ▷ **ParameterValue** (output_control) . . . . . number-array ⇝ *real* / integer
  Wert des berechneten Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert determine_ncc_model_params den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`create_ncc_model`

Siehe auch

`find_ncc_model`

Modul

Matching

find_ncc_model (Image : : ModelID, AngleStart, AngleExtent, MinScore, NumMatches, MaxOverlap, SubPixel, NumLevels : Row, Column, Angle, Score)

Suche der besten Matches eines NCC-Modells in einem Bild.

`find_ncc_model` findet die besten `NumMatches` Instanzen des NCC-Modells `ModelID` im Eingabebild `Image`. Das Modell muss zuvor mit `create_ncc_model` erzeugt oder mit `read_ncc_model` eingelesen worden sein.

Die Position und Rotation der gefundenen Instanzen des Modells wird in `Row`, `Column` und `Angle` zurückgeliefert. Zusätzlich wird in `Score` die Bewertung der gefundenen Instanzen zurückgegeben. Nähere Informationen finden Sie in den parameterspezifischen Abschnitten.

Es ist zu beachten, dass die NCC sehr empfindlich gegenüber Verdeckungen und Störobjekten sowie gegenüber nichtlinearen Beleuchtungsänderungen im Bild ist. Falls ein Modell selbst bei Verdeckungen, Störobjekten oder nichtlinearen Beleuchtungsänderungen gefunden werden soll, sollte die Suche mit dem formbasierten Matching durchgeführt werden (siehe z.B. `create_shape_model`).

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes `Image` gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des NCC-Modells mit `create_ncc_model` verwendet wurde. Ein eventuell mit `set_ncc_model_origin` anders gesetzter Ursprung wird hier nicht berücksichtigt. Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als `MinScore` erreichen würde (siehe unten). Auf Grund von Rundungseffekten kann es jedoch passieren, dass das Modell für am Rand gefundene Instanzen bis zu $0.5 * 2^{\texttt{NumLevels}-1}$ Pixel außerhalb des Definitionsbereiches des Bildes liegt.

AngleStart und AngleExtent: Die Parameter `AngleStart` und `AngleExtent` legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Der Winkelbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit `create_ncc_model` angegeben worden ist.

MinScore: Der Parameter `MinScore` legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von `MinScore` gewählt werden kann, desto schneller verläuft die Suche.

NumMatches: Mit `NumMatches` kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als `NumMatches` Instanzen eine Bewertung größer als `MinScore` erreichen, werden nur die besten `NumMatches` Instanzen zurückgeliefert. Falls weniger als `NumMatches` Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter `MinScore` hat Vorrang vor `NumMatches`. Sollen alle Modellinstanzen, deren Bewertung `MinScore` übersteigt, im Bild gefunden werden, muss `NumMatches` auf *0* gesetzt werden.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Falls das Modell sich wiederholende Strukturen aufweist, kann es vorkommen, dass mehrere Instanzen mit gleichen Rotationen an ähnlichen

Positionen im Bild gefunden werden. Mit dem Parameter MaxOverlap kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als MaxOverlap überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung des Definitionsbereiches der Modelle (siehe smallest_rectangle2). Bei MaxOverlap $= 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei MaxOverlap $= 1$ werden alle gefundenen Instanzen zurückgeliefert.

SubPixel: Der Parameter SubPixel gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls SubPixel auf *'false'* gesetzt wird, wird die Lage des Modells nur pixelgenau bzw. mit der bei create_ncc_model angegebenen Winkelauflösung bestimmt. Falls SubPixel auf *'true'* gesetzt wird, werden sowohl die Position als auch die Rotation subpixelgenau bestimmt. Dabei wird die Lage des Modells anhand der Score-Funktion interpoliert. Dieser Modus kostet fast keine Rechenzeit und liefert eine hohe Genauigkeit. Im Normalfall sollte also SubPixel $=$ *'true'* gewählt werden. Für eine subpixelgenaue Bestimmung der Lage muss das Modell auf der untersten verwendeten Pyramidenebene einen Abstand von mindestens 2 Pixel vom Bildrand haben. Wenn die gefundene Instanz des Modells näher am Bildrand liegt, wird die Lage des Modells nur pixelgenau bzw. mit der bei create_ncc_model angegebenen Winkelauflösung bestimmt, selbst wenn SubPixel auf *'true'* gesetzt ist.

NumLevels: Mit NumLevels wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit create_ncc_model angegebenen Bereich beschnitten. Falls NumLevels als *0* angegeben wird, wird die mit create_ncc_model angegebene Anzahl verwendet.

In manchen Fällen kann es sein, dass die Anzahl der Pyramidenebenen, die beispielsweise automatisch mit create_ncc_model ermittelt wurde, zu hoch ist. Dann werden eventuell Instanzen, die eine sehr hohe finale Bewertung gehabt hätten, bereits auf der höchsten Pyramidenebene ausgeschlossen und damit nicht gefunden. Anstatt nun MinScore auf einen sehr geringen Wert zu setzen, um alle Matches zu finden, kann der Wert von NumLevels mit get_ncc_model_params ermittelt werden und dann ein etwas geringerer Wert in find_ncc_model genutzt werden. Diese Herangehensweise führt oft zu besseren Ergebnissen hinsichtlich Geschwindigkeit und Robustheit.

Optional kann NumLevels einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, Laufzeit einzusparen. Allerdings ist in diesem Modus im Allgemeinen die Genauigkeit der gefundenen Lageparameter geringer als im Normalfall, in dem die Matches bis auf die unterste Pyramidenebene verfolgt werden. Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden.

Ausgabeparameter im Detail

Row, Column und Angle: Die Position und Rotation der gefundenen Instanzen des Modells wird in Row, Column und Angle zurückgeliefert. Die Koordinaten Row und Column beschreiben die Position des Ursprungs des Modells im Suchbild. Die Werte sind allerdings bereits geringfügig angepasst, so dass sie direkt zur Erstellung einer Transformationsmatrix verwendet werden können, die für ein Alignment oder die Visualisierung des Modells benötigt wird. Diese Anpassung der Koordinaten hat mit der Art und Weise zu tun, in der HALCON ikonische Objekte transformiert, siehe hierzu affine_trans_pixel. Das Beispiel weiter unten zeigt, wie die Transformationsmatrix erzeugt wird und wie man damit die exakten Koordinaten berechnet.

Beachten Sie, dass die Visualisierung aber auch mit der Prozedur dev_display_ncc_matching_results realisiert werden kann.

Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des NCC-Modells mit create_ncc_model verwendet wurde. Ein anderer Ursprung kann mit set_ncc_model_origin festgelegt werden.

Score: In Score wird die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist die normalisierte Kreuzkorrelation (engl.: normalized cross correlation) zwischen dem Muster $t(r, c)$ und dem Bild $i(r, c)$:

$$\mathrm{ncc}(r,c) = \frac{1}{n} \sum_{(u,v) \in R} \frac{t(u,v) - m_t}{\sqrt{s_t^2}} \cdot \frac{i(r+u, c+v) - m_i(r,c)}{\sqrt{s_i^2(r,c)}}$$

Hierbei bezeichnet n die Anzahl der Punkte im Muster, R den Definitionsbereich (ROI) des Musters, m_t den mittleren Grauwert des Musters

$$m_t = \frac{1}{n} \sum_{(u,v) \in R} t(u,v)$$

s_t^2 die Varianz der Grauwerte des Musters

$$s_t^2 = \frac{1}{n} \sum_{(u,v) \in R} (t(u,v) - m_t)^2$$

$m_i(r,c)$ den mittleren Grauwert an der Position (r,c) über alle Punkte im Muster (d.h. die Punkte im Muster sind um (r,c) verschoben)

$$m_i(r,c) = \frac{1}{n} \sum_{(u,v) \in R} i(r+u, c+v)$$

und $s_i^2(r,c)$ die Varianz der Grauwerte an der Position (r,c) über alle Punkte im Muster

$$s_i^2(r,c) = \frac{1}{n} \sum_{(u,v) \in R} (i(r+u, c+v) - m_i(r,c))^2$$

Die NCC misst, wie gut das Muster und das Bild an einem bestimmten Punkt (r,c) übereinstimmen. Sie nimmt Werte zwischen -1 und 1 an. Je größer der Absolutbetrag der Korrelation ist, desto besser ist die Übereinstimmung zwischen Muster und Bild. Ein Wert von 1 bedeutet, dass die Grauwerte im Bild eine lineare Transformation der Grauwerte im Muster sind:

$$i(r+u, c+v) = at(u,v) + b$$

wobei $a > 0$. Ein Wert von -1 bedeutet, dass die Grauwerte im Bild eine lineare Transformation der Grauwerte im Muster mit $a < 0$ sind. Das Muster hat in diesem Fall im Bild also genau die umgekehrte Polarität. Aufgrund der obigen Eigenschaft ist die NCC invariant gegenüber linearen Beleuchtungsänderungen.

Die NCC wie oben definiert wird verwendet, falls das Modell mit `Metric` = *'use_polarity'* erzeugt wurde. Falls das Modell mit `Metric` = *'ignore_global_polarity'* erzeugt wurde, wird der Absolutbetrag von $\mathrm{ncc}(r,c)$ als Bewertung verwendet.

Setzen eines Timeout

Mittels des Operators `set_ncc_model_param` können sie einen *'timeout'* für `find_ncc_model` angeben. Wenn `find_ncc_model` diesen *'timeout'* erreicht endet er ohne Ergebnis und gibt den Fehlercode 9400 (H_ERR_TIMEOUT) zurück.

Darstellung der Resultate

Die Verwendung der Prozedur `dev_display_ncc_matching_results` wird zur Darstellung der Resultate formbasierten Matchings stark empfohlen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Achtung

`find_ncc_model` kann teilweise auf OpenCL-Geräten ausgeführt werden, wenn diese die cl_khr_global_int32_base_atomics OpenCL-Erweiterung unterstützen. Es wird nur die Suche auf der obersten Pyramidenstufe auf dem OpenCL-Gerät durchgeführt, während das Tracking der potentiellen Matches auf der CPU durchgeführt wird. Wenn die Domäne des Bildes in dem gesucht werden soll deutlich kleiner ist als die Größe des Bildes, sollte daher `crop_domain` verwendet werden, um die Menge der von dem OpenCL

auf das Host-System zu übertragenden Daten zu reduzieren. Abhängig von einer Vielzahl von Faktoren kann `find_ncc_model` auf OpenCL-Geräten sowohl deutlich schneller, als auch deutlich langsamer laufen. Daher kann man hier keine generelle Aussage treffen und muss jeweils mit Bildern aus der Anwendung testen, ob der Einsatz von OpenCL für `find_ncc_model` Sinn macht oder nicht.

Außerdem sei bemerkt, dass der intern benötigte Speicher mit der Anzahl der verwendeten Threads anwächst.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild, in dem das Modell gefunden werden soll.
- ▷ **`ModelID`** (input_control) ..... ncc_model ⇝ *handle*
  Handle des Modells.
- ▷ **`AngleStart`** (input_control) ..... angle.rad ⇝ *real*
  Kleinste auftretende Rotation des Modells.
  Defaultwert: -0.39
  Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **`AngleExtent`** (input_control) ..... angle.rad ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Defaultwert: 0.79
  Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
  Restriktion: AngleExtent >= 0
- ▷ **`MinScore`** (input_control) ..... real ⇝ *real*
  Minimale Bewertung der zu findenden Instanzen des Modells.
  Defaultwert: 0.8
  Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
- ▷ **`NumMatches`** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu findenden Instanzen des Modells (oder 0 für alle Treffer).
  Defaultwert: 1
  Wertevorschläge: `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}
- ▷ **`MaxOverlap`** (input_control) ..... real ⇝ *real*
  Maximale Überlappung der zu findenden Instanzen des Modells.
  Defaultwert: 0.5
  Wertevorschläge: `MaxOverlap` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich: $0 \leq$ `MaxOverlap` ≤ 1
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
- ▷ **`SubPixel`** (input_control) ..... string ⇝ *string*
  Subpixelgenauigkeit.
  Defaultwert: 'true'
  Werteliste: `SubPixel` ∈ {'false', 'true'}
- ▷ **`NumLevels`** (input_control) ..... integer(-array) ⇝ *integer*
  Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls $|$`NumLevels`$| = 2$).
  Defaultwert: 0
  Werteliste: `NumLevels` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **`Row`** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinate der gefundenen Instanzen des Modells.
- ▷ **`Column`** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinate der gefundenen Instanzen des Modells.
- ▷ **`Angle`** (output_control) ..... angle.rad-array ⇝ *real*
  Rotationswinkel der gefundenen Instanzen des Modells.
- ▷ **`Score`** (output_control) ..... real-array ⇝ *real*
  Bewertung der gefundenen Instanzen des Modells.

Beispiel

```
create_ncc_model (TemplateImage, 'auto', rad(-45), rad(90), 'auto', \
                  'use_polarity', ModelID)
find_ncc_model (SearchImage, ModelID, rad(-45), rad(90), 0.7, 1, \
                0.5, 'true', 0, Row, Column, Angle, Score)
* Create transformation matrix
vector_angle_to_rigid (0, 0, 0, Row, Column, Angle, HomMat2D)
* Calculate true position of the model origin in the search image
affine_trans_pixel (HomMat2D, 0, 0, RowObject, ColumnObject)
* display the results
dev_display_ncc_matching_results (ModelID, 'red', Row, Column, \
                                  Angle, 0)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_ncc_model` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_ncc_model`, `read_ncc_model`, `set_ncc_model_origin`

Nachfolger

`clear_ncc_model`

Alternativen

`find_shape_model`, `find_scaled_shape_model`, `find_aniso_shape_model`, `find_shape_models`, `find_scaled_shape_models`, `find_aniso_shape_models`

Modul

Matching

find_ncc_models (Image : : ModelIDs, AngleStart, AngleExtent, MinScore, NumMatches, MaxOverlap, SubPixel, NumLevels : Row, Column, Angle, Score, Model)

Suche der besten Matches mehrerer NCC-Modelle.

`find_ncc_models` findet die besten `NumMatches` Instanzen der NCC-Modelle, die in dem Tupel `ModelIDs` übergeben werden, im Eingabebild `Image`. Die Modelle müssen zuvor mit `create_ncc_model` erzeugt oder mit `read_ncc_model` eingelesen worden sein. Im Gegensatz zu `find_ncc_model` kann also eine Mehrzahl von Modellen gleichzeitig im selben Bild gesucht werden.

Die Position und Rotation der gefundenen Instanzen des Modells wird in `Row`, `Column` und `Angle` zurückgeliefert. In `Score` wird die Bewertung der gefundenen Instanzen zurückgegeben. Der gefundene Modelltyp wird in `Model` zurückgeliefert. Nähere Informationen finden Sie in den parameterspezifischen Abschnitten.

Besonderheiten der Parametersemantik

Gegenüber `find_ncc_model` unterscheidet sich die Semantik aller Eingabeparameter etwas. Alle Eingabeparameter müssen entweder genau ein Element enthalten oder dieselbe Anzahl von Elementen wie `ModelIDs`. (`NumLevels` kann auch zwei oder zweimal die Anzahl von Elementen wie `ModelIDs` enthalten; siehe unten.) Im ersten Fall wird der Wert des Eingabeparameters für alle Modelle gleich verwendet. Im zweiten Fall wird das jeweilige Element des Eingabeparameters für das entsprechende Modell in `ModelIDs` verwendet. Details hierzu werden im zugehörigen Abschnitt weiter unten beschrieben. Ein Aufruf von `find_ncc_models` mit mehreren Werten für `ModelIDs`, `NumMatches` und `MaxOverlap` entspricht von der Wirkung her also mehreren unabhängigen Aufrufen von `find_ncc_model` mit den jeweiligen Parametern, ist allerdings wesentlich effizienter.

Eingabeparameter im Detail

Eingabebild und seine Region: Der Definitionsbereich des Bildes `Image` gibt den Suchbereich für den Referenzpunkt des Modells an, d.h. für den Schwerpunkt der Region des Bildes, das zur Erzeugung des NCC-Modells mit `create_ncc_model` verwendet wurde. Ein eventuell mit `set_ncc_model_origin` anders gesetzter Ursprung wird nicht berücksichtigt. Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild passt. Das bedeutet, dass das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als `MinScore` erreichen würde. Auf Grund von Rundungseffekten kann es jedoch passieren, dass das Modell für am Rand gefundene Instanzen bis zu $0.5 * 2^{\texttt{NumLevels}-1}$ Pixel außerhalb des Definitionsbereiches des Bildes liegt.

`Image` kann ein einzelnes Bildobjekt beinhalten oder ein Bildobjektarray. Falls `Image` ein einzelnes Bildobjekt enthält, wird dessen Region als Suchbereich für alle Modelle in `ModelIDs` verwendet. Falls `Image` mehrere Bildobjekte enthält, wird die jeweilige Region als Suchbereich für das entsprechende Modell aus `ModelIDs` verwendet. In diesem Fall müssen die Bilder, abgesehen von ihrer Region, identisch sein. D.h. `Image` kann nicht in beliebiger Weise mit `concat_obj` aufgebaut werden, sondern muss aus demselben Bild mit `add_channels` oder äquivalenten Aufrufen erzeugt werden. Falls das nicht der Fall ist, wird eine Fehlermeldung zurückgeliefert.

AngleStart und AngleExtent: Die Parameter `AngleStart` und `AngleExtent` legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Der Winkelbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit `create_ncc_model` angegeben worden ist.

Außerdem ist zu beachten, dass es in manchen Fällen vorkommen kann, dass Instanzen gefunden werden, deren Rotation geringfügig außerhalb des übergebenen Winkelbereichs liegt. Dies kann dann auftreten, wenn der übergebene Winkelbereich kleiner ist als der Bereich, der bei der Erzeugung des Modells angegeben worden ist.

MinScore: Der Parameter `MinScore` legt fest, welche Bewertung ein potentieller Match mindestens besitzen muss, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von `MinScore` gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, dass das Modell niemals verdeckt wird, kann `MinScore` auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden. Werden die Matches nicht bis zur untersten Pyramidenebene verfolgt (siehe unten), kann es in manchen Fällen vorkommen, dass Instanzen gefunden werden, deren Score geringfügig unter dem Wert von `MinScore` liegen.

NumMatches: Mit `NumMatches` kann angegeben werden, wie viele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als `NumMatches` Instanzen eine Bewertung größer als `MinScore` erreichen, werden nur die besten `NumMatches` Instanzen zurückgeliefert. Falls weniger als `NumMatches` Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter `MinScore` hat Vorrang vor `NumMatches`. Sollen alle Modellinstanzen, deren Bewertung `MinScore` übersteigt, im Bild gefunden werden, muss `NumMatches` auf *0* gesetzt werden.

Falls `NumMatches` einen Wert enthält, liefert `find_ncc_models` die `NumMatches` besten Instanzen des Modells unabhängig von der Art des Modells zurück. Falls z.B. in `ModelIDs` zwei Modelle übergeben werden und $\texttt{NumMatches} = 2$ gewählt wird, kann es sein, dass zwei Instanzen des ersten Modells und keine des zweiten Modells, eine Instanz des ersten Modells und eine des zweiten Modells oder keine Instanz des ersten Modells und zwei des zweiten Modells zurückgeliefert werden. Falls hingegen `NumMatches` mehrere Werte enthält, werden so viele Instanzen des jeweiligen Modells in `ModelIDs` zurückgeliefert, wie durch das entsprechende Element von `NumMatches` angegeben. Falls z.B. $\texttt{NumMatches} = [1,1]$ gewählt wird, wird eine Instanz des ersten Modells und eine des zweiten Modells zurückgeliefert.

MaxOverlap: Falls das Modell Symmetrien aufweist, kann es vorkommen, dass mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter `MaxOverlap` kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als `MaxOverlap` überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe `smallest_rectangle2`). Bei $\texttt{MaxOverlap} = 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei $\texttt{MaxOverlap} = 1$ werden alle gefundenen Instanzen zurückgeliefert.

Falls `MaxOverlap` einen Wert enthält, wird die Überlappung für alle gefundenen Instanzen des Modells unabhängig von der Art des Modells berechnet. Es werden also Instanzen verschiedener und gleicher Modelle eliminiert, die sich zu stark überlappen. Falls hingegen mehrere Werte in `MaxOverlap` übergeben

werden, wird die Überlappung nur innerhalb der gefundenen Instanzen des jeweiligen Typs der Modelle in ModelIDs berechnet. Es werden dann nur sich zu stark überlappende Instanzen gleicher Modelle eliminiert. Modelle verschiedenen Typs können sich in diesem Modus vollständig überlappen.

SubPixel: Der Parameter SubPixel gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls SubPixel auf *'false'* gesetzt wird, wird die Lage des Modells nur pixelgenau bzw. mit der bei create_ncc_model angegebenen Winkelauflösung bestimmt. Falls SubPixel auf *'true'* gesetzt wird, werden sowohl die Position als auch die Rotation subpixelgenau bestimmt. Dabei wird die Lage des Modells anhand der Score-Funktion interpoliert.

NumLevels: Mit NumLevels wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit create_ncc_model angegebenen Bereich beschnitten. Falls NumLevels als *0* angegeben wird, wird die mit create_ncc_model angegebene Anzahl verwendet.

In manchen Fällen kann es sein, dass die Anzahl der Pyramidenebenen, die beispielsweise automatisch mit create_ncc_model ermittelt wurde, zu hoch ist. Dann werden eventuell Instanzen, die eine sehr hohe finale Bewertung gehabt hätten, bereits auf der höchsten Pyramidenebene ausgeschlossen und damit nicht gefunden. Anstatt nun MinScore auf einen sehr geringen Wert zu setzen, um alle Matches zu finden, kann der Wert von NumLevels mit get_ncc_model_params ermittelt werden und dann ein etwas geringerer Wert in find_ncc_models genutzt werden. Diese Herangehensweise führt oft zu besseren Ergebnissen hinsichtlich Geschwindigkeit und Robustheit.

Optional kann NumLevels einen zweiten Wert enthalten, der die unterste Pyramidenebene spezifiziert, bis auf welche die Matches verfolgt werden sollen. Ein Wert von *[4,2]* bedeutet also, dass das Matching auf der vierten Pyramidenebene begonnen wird und auf der zweituntersten Pyramidenebene beendet wird (die unterste Pyramidenebene hat den Wert 1). Dieser Mechanismus kann dazu verwendet werden, Laufzeit einzusparen. Allerdings ist in diesem Modus im Allgemeinen die Genauigkeit der gefundenen Lageparameter geringer als im Normalfall, in dem die Matches bis auf die unterste Pyramidenebene verfolgt werden. Falls die unterste zu verwendende Pyramidenebene zu groß gewählt wird, kann es vorkommen, dass die gewünschte Genauigkeit nicht mehr erreicht werden kann, oder dass falsche Instanzen des Modells gefunden werden, weil das Modell auf den oberen Pyramidenstufen nicht eindeutig genug ist, um eine sichere Selektion der korrekten Instanz des Modells zu ermöglichen. In diesem Fall muss ein kleinerer Wert für die unterste zu verwendende Pyramidenebene gewählt werden.

Falls die unterste Pyramidenebene für jedes Modell einzeln spezifiziert werden soll, müssen in NumLevels zweimal die Anzahl von Elementen in ModelIDs angegeben werden. Dabei sind die Anzahl der Pyramidenebenen und die untersten Pyramidenebenen verschränkt anzugeben. Falls z.B. zwei Modelle in ModelIDs angegeben sind, die oberste Pyramidenebene für das erste Modell 5 und für das zweite Modell 4 sein soll und die unterste Pyramidenebene für das erste Modell 2 und für das zweite 1 sein soll, so ist NumLevels $= [5, 2, 4, 1]$ zu wählen. Falls genau zwei Modelle in ModelIDs angegeben sind, ergibt sich als Spezialfall, dass, falls die unterste Pyramidenebene spezifiziert werden soll, die oberste und unterste Pyramidenebene für beide Modelle explizit spezifiziert werden muss, selbst, wenn sie gleich sind. Ein Tupel der Länge zwei in NumLevels wird in diesem Fall nämlich als Spezifikation der obersten Pyramidenebene für die zwei Modelle interpretiert.

Ausgabeparameter im Detail

Row, Column und Angle: Die Position und Rotation der gefundenen Instanzen der Modelle wird in Row, Column und Angle zurückgeliefert. Die Koordinaten Row und Column sind die Koordinaten des Ursprungs des Modells im Suchbild. Normalerweise ist der Ursprung des Modells der Schwerpunkt der Region des Bildes, das zur Erzeugung des NCC-Modells mit create_ncc_model verwendet wurde. Ein anderer Ursprung kann mit set_ncc_model_origin festgelegt werden.

Beachten Sie, dass die Koordinaten Row und Column nicht exakt mit dem Ursprung des Modells übereinstimmen und daher nicht direkt verwendet werden sollten. Die Werte sind dafür optimiert, die Transformationsmatrix zu erzeugen, mit der die Matching-Ergebnisse für verschiedene Aufgaben verwendet werden können, z.B. um ROIs für andere Bildverarbeitungsschritte einem Objekt nachzuführen. Das bei find_ncc_model angegebene Beispiel zeigt, wie diese Matrix erzeugt wird und wie man damit das Modell an der gefundenen Position im Suchbild visualisiert und die exakten Koordinaten berechnet.

Beachten Sie, dass die Visualisierung aber auch mit der Prozedur dev_display_ncc_matching_results realisiert werden kann.

Score: In Score wird die Bewertung der gefundenen Instanzen zurückgegeben. Die Berechnung von Score ist in find_ncc_model beschrieben.

Model: Der gefundene Modelltyp wird in `Model` zurückgeliefert. Die Elemente von `Model` sind ein Index in das Tupel `ModelIDs`, d.h. sie können Werte von 0 bis $|$`ModelIDs`$| - 1$ enthalten. Ein Wert von 0 in einem Element von `Model` entspricht also einer Instanz des ersten in `ModelIDs` übergebenen Modells.

Setzen eines Timeout

Mittels des Operators `set_ncc_model_param` können sie einen *'timeout'* für `find_ncc_models` angeben. Falls die durch `ModelIDs` referenzierten Modelle unterschiedliche Werte für *'timeout'* besitzen, wählt `find_ncc_models` den niedrigsten aus. Wenn `find_ncc_models` diesen *'timeout'* erreicht endet er ohne Ergebnis und gibt den Fehlercode 9400 (H_ERR_TIMEOUT) zurück.

Darstellung der Resultate

Die Verwendung der Prozedur `dev_display_ncc_matching_results` wird zur Darstellung der Resultate formbasierten Matchings stark empfohlen.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Achtung

Es sei bemerkt, dass der intern benötigte Speicher mit der Anzahl der verwendeten Threads anwächst.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Eingabebild, in dem die Modelle gefunden werden sollen.
- ▷ **`ModelIDs`** (input_control) ..... ncc_model(-array) ⇝ *handle*
  Handle der Modelle.
- ▷ **`AngleStart`** (input_control) ..... angle.rad(-array) ⇝ *real*
  Kleinste auftretende Rotation des Modelle.
  Defaultwert: -0.39
  Wertevorschläge: `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **`AngleExtent`** (input_control) ..... angle.rad(-array) ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Defaultwert: 0.79
  Wertevorschläge: `AngleExtent` ∈ {6.29, 3.14, 1.57, 0.79, 0.39, 0.0}
  Restriktion: AngleExtent >= 0
- ▷ **`MinScore`** (input_control) ..... real(-array) ⇝ *real*
  Minimale Bewertung der zu findenden Instanzen des Modelle.
  Defaultwert: 0.8
  Wertevorschläge: `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich: $0 \leq$ `MinScore` ≤ 1
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
- ▷ **`NumMatches`** (input_control) ..... integer(-array) ⇝ *integer*
  Anzahl der zu findenden Instanzen des Modelle (oder 0 für alle Treffer).
  Defaultwert: 1
  Wertevorschläge: `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}
- ▷ **`MaxOverlap`** (input_control) ..... real(-array) ⇝ *real*
  Maximale Überlappung der zu findenden Instanzen des Modelle.
  Defaultwert: 0.5
  Wertevorschläge: `MaxOverlap` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich: $0 \leq$ `MaxOverlap` ≤ 1
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.05
- ▷ **`SubPixel`** (input_control) ..... string(-array) ⇝ *string*
  Subpixelgenauigkeit.
  Defaultwert: 'true'
  Werteliste: `SubPixel` ∈ {'false', 'true'}

▷ **NumLevels** (input_control) ..... integer(-array) ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen (und unterste zu verwendende Pyramidenebene falls |NumLevels| = 2).
Defaultwert: 0
Werteliste: NumLevels ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Row** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinate der gefundenen Instanzen des Modelle.

▷ **Column** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinate der gefundenen Instanzen des Modelle.

▷ **Angle** (output_control) ..... angle.rad-array ⇝ *real*
Rotationswinkel der gefundenen Instanzen des Modelle.

▷ **Score** (output_control) ..... real-array ⇝ *real*
Bewertung der gefundenen Instanzen des Modelle.

▷ **Model** (output_control) ..... integer-array ⇝ *integer*
Indices der gefundenen Instanzen des Modelle.

Beispiel

```
read_image (Image, 'pcb_focus/pcb_focus_telecentric_061')
gen_rectangle1 (ROI_0, 236, 241, 313, 321)
gen_circle (ROI_1, 281, 653, 41)
reduce_domain (Image, ROI_0, ImageReduced1)
reduce_domain (Image, ROI_1, ImageReduced2)
create_ncc_model (ImageReduced1, 'auto', rad(-45), rad(90), 'auto', \
                  'use_polarity', ModelID1)
create_ncc_model (ImageReduced2, 'auto', rad(-45), rad(90), 'auto', \
                  'use_polarity', ModelID2)
ModelIDs:=[ModelID1, ModelID2]
find_ncc_models (Image, ModelIDs, rad(-45), rad(90), 0.7, [1,1], 0.5, \
                 'true', 0, Row, Column, Angle, Score, Model)
dev_display_ncc_matching_results (ModelIDs, 'red', Row, Column, \
                                  Angle, Model)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert find_ncc_models den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

create_ncc_model, read_ncc_model, set_ncc_model_origin

Nachfolger

clear_ncc_model

Alternativen

find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_shape_models, find_scaled_shape_models, find_aniso_shape_models

Modul

Matching

get_ncc_model_origin (: : ModelID : Row, Column)

Auslesen des Ursprungs (Referenzpunktes) eines NCC-Modells.

Der Operator get_ncc_model_origin gibt den Ursprung (Referenzpunkt) des NCC-Modells ModelID zurück. Der Ursprung wird relativ zum Schwerpunkt der Region des Bildes, das zur Erzeugung des NCC-Modells mit create_ncc_model verwendet wurde, angegeben. Ein Ursprung von (0,0) heißt also, dass der Schwerpunkt der Region des Modellbildes verwendet wird. Ein Ursprung von (-20,-40) heißt, dass der Ursprung links oberhalb des Schwerpunktes liegt.

Parameter

- ▷ **ModelID** (input_control) ..... ncc_model ⇝ *handle*
  Handle des Modells.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenkoordinate des Ursprungs des NCC-Modells.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenkoordinate des Ursprungs des NCC-Modells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert get_ncc_model_origin den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_ncc_model, read_ncc_model, set_ncc_model_origin

Nachfolger

find_ncc_model

Siehe auch

area_center

Modul

Matching

get_ncc_model_params (: : ModelID : NumLevels, AngleStart, AngleExtent, AngleStep, Metric)

Auslesen der Parameter eines NCC-Modells.

Der Operator get_ncc_model_params gibt die zur Erzeugung des NCC-Modells ModelID mit create_ncc_model verwendeten Parameter zurück. Dies ist insbesondere hilfreich, um die Parameter NumLevels und AngleStep zu überprüfen, falls diese durch create_ncc_model automatisch bestimmt wurden. Weiterhin kann get_ncc_model_params verwendet werden, um die Parameter zu überprüfen, falls das Modell mit read_ncc_model eingelesen wurde.

Parameter

- ▷ **ModelID** (input_control) ..... ncc_model ⇝ *handle*
  Handle des Modells.
- ▷ **NumLevels** (output_control) ..... integer ⇝ *integer*
  Anzahl von Pyramidenebenen.
- ▷ **AngleStart** (output_control) ..... angle.rad ⇝ *real*
  Kleinste auftretende Rotation des Musters.
- ▷ **AngleExtent** (output_control) ..... angle.rad ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Zusicherung: AngleExtent >= 0

▷ **AngleStep** (output_control) . . . . . angle.rad ⇝ *real*
Schrittweite der Winkel (Auflösung).
Zusicherung: AngleStep >= 0

▷ **Metric** (output_control) . . . . . string ⇝ *string*
Art der zum Matchen verwendeten Metrik.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `get_ncc_model_params` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_ncc_model`, `read_ncc_model`

Siehe auch

`find_ncc_model`

Modul

Matching

get_ncc_model_region (: ModelRegion : ModelID :)

Auslesen der Region mit der das NCC-Modell erstellt wurde.

Der Operator `get_ncc_model_region` gibt in `ModelRegion` die Region zurück, die verwendet wurde um das NCC-Modell `ModelID` zu erzeugen. Diese Region kann z.B. zur Visualisierung der gefundenen Instanzen des Modells verwendet werden. Dabei ist darauf zu achten, dass die Lage von `ModelRegion` so normalisiert worden ist, dass der Referenzpunkt (siehe `set_ncc_model_origin`) des Modells an der Pixelposition $(0, 0)$ liegt. D.h., die Region muss lediglich an die gefundene Stelle verschoben werden (und eventuell um diesen Punkt rotiert werden). `get_ncc_model_region` ignoriert den Wert des Systemparameters *'clip_region'*.

Parameter

▷ **ModelRegion** (output_object) . . . . . region ⇝ *object*
Modellregion des NCC-Modells.

▷ **ModelID** (input_control) . . . . . ncc_model ⇝ *handle*
Handle des Modells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `get_ncc_model_region` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_ncc_model`

Modul

Matching

read_ncc_model (: : FileName : ModelID)

Einlesen eines NCC-Modells von Datei.

read_ncc_model liest ein NCC-Modell, das mit write_ncc_model geschrieben wurde, aus der Datei FileName ein. Die Dateiendung für das NCC-Modell ist in HALCON 'ncm'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Dateiendung: .ncm

▷ **ModelID** (output_control) . . . . . ncc_model ⇝ *handle*
Handle des Modells.

Ergebnis

Ist der Dateiname korrekt, dann liefert read_ncc_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

find_ncc_model, find_ncc_models

Siehe auch

create_ncc_model, clear_ncc_model

Modul

Matching

serialize_ncc_model (: : ModelID : SerializedItemHandle)

Serialisieren eines NCC-Modells.

serialize_ncc_model serialisiert die Daten eines NCC-Modells (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_ncc_model in eine Datei geschrieben werden. Das NCC-Modell wird in dem Handle ModelID übergeben. Das serialisierte NCC-Modell wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_ncc_model wieder deserialisiert werden.

Parameter

▷ **ModelID** (input_control) . . . . . ncc_model ⇝ *handle*
Handle des Modells.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_ncc_model den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_ncc_model`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_ncc_model`

Modul

Matching

set_ncc_model_origin (: : ModelID, Row, Column :)

Setzen des Ursprungs (Referenzpunktes) eines NCC-Modells.

Der Operator `set_ncc_model_origin` setzt den Ursprung (Referenzpunkt) des NCC-Modells `ModelID` auf einen neuen Wert. Der Ursprung wird relativ zum Schwerpunkt der Region des Bildes, das zur Erzeugung des NCC-Modells mit `create_ncc_model` verwendet wurde, angegeben. Ein Ursprung von (0,0) heißt also, dass der Schwerpunkt der Region des Modellbildes verwendet wird. Ein Ursprung von (-20,-40) heißt, dass der Ursprung links oberhalb des Schwerpunktes liegt.

Parameter

- ▷ **ModelID** (input_control) ..... ncc_model ⇝ *handle*
  Handle des Modells.
- ▷ **Row** (input_control) ..... point.y ⇝ *real*
  Zeilenkoordinate des Ursprungs des NCC-Modells.
- ▷ **Column** (input_control) ..... point.x ⇝ *real*
  Spaltenkoordinate des Ursprungs des NCC-Modells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `set_ncc_model_origin` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ncc_model`, `read_ncc_model`

Nachfolger

`find_ncc_model`, `get_ncc_model_origin`, `find_ncc_models`

Siehe auch

`area_center`

Modul

Matching

set_ncc_model_param (: : ModelID, GenParamName, GenParamValue :)

Setzen ausgewählter Parameter eines NCC-Modells.

Der Operator set_ncc_model_param setzt ausgewählte Parameter im NCC-Modell ModelID. Die folgenden Parameter können modifiziert werden:

'timeout'

Setzt die maximale Laufzeit der Operatoren die verwendet werden um das NCC-Modell ModelID zu finden (mit find_ncc_model). Dies ist dann besonders nützlich, wenn eine maximale Zykluszeit sichergestellt werden muss. Der *'timeout'* muss in Millisekunden angegeben werden. Die zeitliche Genauigkeit des Timeouts hängt von vielen Faktoren ab, darunter ist die Größe des Modells, die Geschwindigkeit des Rechners und der *'timer_mode'* der über set_system gesetzt wurde. Beachten Sie, dass sich die Laufzeit der Modellsuche bei aktiviertem Timeout um bis zu 10 Prozent erhöht. Um den Timeout zu deaktivieren, können sie entweder einen negativen Wert oder *'false'* angeben.

Parameter

▷ **ModelID** (input_control) . . . . . ncc_model ⇝ *handle*
Handle des Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name-array ⇝ *string*
Parameternamen.
Werteliste: GenParamName ∈ {'timeout'}

▷ **GenParamValue** (input_control) . . . . . attribute.value-array ⇝ *real* / integer / string
Parameterwerte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_ncc_model_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ModelID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_ncc_model

Nachfolger

find_ncc_model

Modul

Matching

write_ncc_model (: : ModelID, FileName :)

Schreiben eines NCC-Modells auf Datei.

write_ncc_model schreibt ein NCC-Modell in die Datei FileName. Das Modell kann mit read_ncc_model wieder eingelesen werden. Die Dateiendung für das NCC-Modell ist in HALCON 'ncm'.

Parameter

▷ **ModelID** (input_control) . . . . . ncc_model ⇝ *handle*
Handle des Modells.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: .ncm

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert `write_ncc_model` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_ncc_model`

Modul

Matching

Kapitel 18

Matrix

18.1 Arithmetik

abs_matrix (: : MatrixID : MatrixAbsID)

Berechnet den Absolutbetrag der Elemente einer Matrix.

Der Operator abs_matrix berechnet den Absolutbetrag aller Elemente der Eingabematrix Matrix, die durch das Matrix Handle MatrixID gegeben ist. Es wird eine neue Matrix MatrixAbs mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle MatrixAbsID der Matrix MatrixAbs zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixAbs}_{ij} \quad = \quad |\texttt{Matrix}_{ij}|.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \texttt{MatrixAbs} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixID** (input_control) ..... matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **MatrixAbsID** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Absolutbeträgen der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert abs_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

abs_matrix_mod

Modul

Foundation

abs_matrix_mod (: : MatrixID :)

Berechnet den Absolutbetrag der Elemente einer Matrix.

Der Operator abs_matrix_mod berechnet den Absolutbetrag aller Elemente der Eingabematrix Matrix, die durch das Matrix Handle MatrixID gegeben ist. Die Eingabematrix wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die Formel für die Berechnung lautet:

$$\mathtt{Matrix}_{ij} \quad = \quad |\mathtt{Matrix}_{ij}|.$$

Beispiel:

$$\mathtt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \mathtt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert abs_matrix_mod den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

abs_matrix

Modul

Foundation

add_matrix (: : MatrixAID, MatrixBID : MatrixSumID)

Addiert zwei Matrizen.

Der Operator add_matrix berechnet die Summe der Eingabematrizen MatrixA und MatrixB, die durch die Matrix Handles MatrixAID und MatrixBID gegeben sind. Beide Eingabematrizen müssen die gleichen Dimensionen besitzen. Es wird eine neue Matrix MatrixSum mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle MatrixSumID der Matrix MatrixSum zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die Formel für die Berechnung lautet:

$$\mathtt{MatrixSum} \quad = \quad \mathtt{MatrixA} \quad + \quad \mathtt{MatrixB}.$$

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \texttt{MatrixB} = \begin{bmatrix} 2.0 & 8.0 & -3.0 \\ -4.0 & -1.0 & 5.0 \\ 2.0 & -4.0 & 7.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixSum} = \begin{bmatrix} 5.0 & 9.0 & -5.0 \\ -9.0 & 6.0 & 7.0 \\ -7.0 & -8.0 & 8.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixAID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix A.
- ▷ **MatrixBID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix B.
- ▷ **MatrixSumID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit den Summen der Eingabematrizen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`add_matrix_mod`

Siehe auch

`sub_matrix`, `sub_matrix_mod`

Modul

Foundation

add_matrix_mod (: : MatrixAID, MatrixBID :)

Addiert zwei Matrizen.

Der Operator `add_matrix_mod` berechnet die Summe der Eingabematrizen `MatrixA` und `MatrixB`, die durch die Matrix Handles `MatrixAID` und `MatrixBID` gegeben sind. Beide Eingabematrizen müssen die gleichen Dimensionen besitzen. Die Eingabematrix `MatrixA` wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixA} \quad = \quad \texttt{MatrixA} \quad + \quad \texttt{MatrixB}.$$

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \texttt{MatrixB} = \begin{bmatrix} 2.0 & 8.0 & -3.0 \\ -4.0 & -1.0 & 5.0 \\ 2.0 & -4.0 & 7.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixA} = \begin{bmatrix} 5.0 & 9.0 & -5.0 \\ -9.0 & 6.0 & 7.0 \\ -7.0 & -8.0 & 8.0 \end{bmatrix}$$

Parameter

▷ **MatrixAID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix A.

▷ **MatrixBID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix B.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_matrix_mod` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixAID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`add_matrix`

Siehe auch

`sub_matrix`, `sub_matrix_mod`

Modul

Foundation

div_element_matrix (: : MatrixAID, MatrixBID : MatrixDivID)

Dividiert Matrizen elementweise.

Der Operator `div_element_matrix` dividiert die zwei Eingabematrizen `MatrixA` und `MatrixB` Element für Element. Die Matrizen sind durch die Matrix Handles `MatrixAID` und `MatrixBID` gegeben. Beide Eingabematrizen müssen die gleiche Dimension besitzen. Die neu erstellte Matrix `MatrixDiv` enthält das Ergebnis. Der Operator gibt das Matrix Handle `MatrixDivID` der Matrix `MatrixDiv` zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixDiv}_{ij} \;=\; \texttt{MatrixA}_{ij} \;/\; \texttt{MatrixB}_{ij}.$$

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 4.0 & 8.0 & 0.0 \\ -6.0 & 7.0 & 2.0 \\ -9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 2.0 & 8.0 & -3.0 \\ -4.0 & -1.0 & 4.0 \\ -2.0 & -4.0 & 2.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixDiv} = \begin{bmatrix} 2.0 & 1.0 & 0.0 \\ 1.5 & -7.0 & 0.5 \\ 4.5 & -1.0 & 0.5 \end{bmatrix}$$

Parameter

▷ **MatrixAID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix A.

▷ **MatrixBID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix B.

▷ **MatrixDivID** (output_control) ..... matrix ⇝ *handle*
Matrix Handle mit den dividierten Werten der Eingabematrizen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `div_element_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`div_element_matrix_mod`

Siehe auch

`mult_element_matrix`, `mult_element_matrix_mod`, `scale_matrix`, `scale_matrix_mod`

Modul

Foundation

div_element_matrix_mod (: : MatrixAID, MatrixBID :)

Dividiert Matrizen elementweise.

Der Operator `div_element_matrix_mod` dividiert die zwei Eingabematrizen `MatrixA` und `MatrixB` Element für Element. Die Matrizen sind durch die Matrix Handles `MatrixAID` und `MatrixBID` gegeben. Beide Eingabematrizen müssen die gleiche Dimension besitzen. Die Eingabematrix `MatrixA` wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixA}_{ij} \quad = \quad \texttt{MatrixA}_{ij} \quad / \quad \texttt{MatrixB}_{ij}.$$

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 4.0 & 8.0 & 0.0 \\ -6.0 & 7.0 & 2.0 \\ -9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 2.0 & 8.0 & -3.0 \\ -4.0 & -1.0 & 4.0 \\ -2.0 & -4.0 & 2.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixA} = \begin{bmatrix} 2.0 & 1.0 & 0.0 \\ 1.5 & -7.0 & 0.5 \\ 4.5 & -1.0 & 0.5 \end{bmatrix}$$

Parameter

▷ **MatrixAID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix A.

▷ **MatrixBID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix B.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `div_element_matrix_mod` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixAID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`div_element_matrix`

Siehe auch

`mult_element_matrix`, `mult_element_matrix_mod`, `scale_matrix`, `scale_matrix_mod`

Modul

Foundation

invert_matrix (: : MatrixID, MatrixType, Epsilon : MatrixInvID)

Invertiert eine Matrix.

Der Operator `invert_matrix` berechnet die Inverse der Matrix `Matrix`. Es wird eine neue Matrix `MatrixInv` mit dem Ergebnis generiert und das Matrix Handle `MatrixInvID` dieser Matrix zurückgegeben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich.

Die Inverse wird berechnet, wenn `Epsilon` = 0 ist. Der Matrixtyp wird über `MatrixType` gewählt. Zur Verfügung stehen *'general'* für allgemeine, *'symmetric'* für symmetrische, *'positive_definite'* für symmetrisch positiv definite, *'tridiagonal'* für tridiagonale Matrizen, *'upper_triangular'* für obere, *'permuted_upper_triangular'* für permutierte obere, *'lower_triangular'* für untere und *'permuted_lower_triangular'* für permutierte untere Dreiecksmatrizen.

Beispiel 1:

$$\texttt{Matrix} = \begin{bmatrix} 1.0 & 3.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 5.0 & 5.0 & 7.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'general' \qquad \texttt{Epsilon} = 0$$

$$\rightarrow \qquad \texttt{MatrixInv} = \begin{bmatrix} -1.25 & 1.50 & -0.75 \\ -0.50 & 2.00 & -1.50 \\ 1.25 & -2.50 & 1.75 \end{bmatrix}$$

Beispiel 2:

$$\texttt{Matrix} = \begin{bmatrix} 1.0 & 3.0 & 3.0 \\ 0 & 2.0 & 6.0 \\ 0 & 0 & 10.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'upper_triangular' \qquad \texttt{Epsilon} = 0$$

$$\rightarrow \qquad \texttt{MatrixInv} = \begin{bmatrix} 1.00 & -1.50 & 0.60 \\ 0 & 0.50 & -0.30 \\ 0 & 0 & 0.10 \end{bmatrix}$$

Beispiel 3:

$$\texttt{Matrix} = \begin{bmatrix} 1.0 & 3.0 & 3.0 \\ 0 & 0 & 10.0 \\ 0 & 2.0 & 6.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'permuted_upper_triangular' \qquad \texttt{Epsilon} = 0$$

$$\rightarrow \qquad \texttt{MatrixInv} = \begin{bmatrix} 1.00 & -1.50 & 0.60 \\ 0 & 0.50 & -0.30 \\ 0 & 0 & 0.10 \end{bmatrix}$$

Die Pseudoinverse wird berechnet, wenn `Epsilon` > 0 gesetzt ist. Alle Singulärwerte, die während der Berechnung entstehen und kleiner als `Epsilon` × dem größten Singulärwert sind, werden auf 0 gesetzt. Für diese Werte wird keine interne Division vorgenommen um einer Division durch 0 vorzubeugen. Als Matrixtyp muss `MatrixType` = *'general'* gesetzt werden.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 & 5.0 \\ -5.0 & 7.0 & 2.0 & -6.0 \\ -9.0 & -4.0 & 1.0 & 4.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'general' \qquad \texttt{Epsilon} = 2.2204\text{e-}16$$

$$\rightarrow \qquad \texttt{MatrixInv} = \begin{bmatrix} -0.0021 & -0.0482 & -0.0813 \\ 0.1435 & 0.1137 & -0.0137 \\ -0.0519 & -0.0015 & 0.0028 \\ 0.1518 & 0.0056 & 0.0526 \end{bmatrix}$$

Bemerkung: Die relative Genauigkeit der Gleitkommarepräsentation des genutzten Datentyps (double) ist `Epsilon` = 2.2204e-16.

Es ist ebenfalls zu beachten, dass in den Beispielen Unterschiede in der Bedeutung der Werte in den Ausgabematrizen zu finden sind: Die Ergebnisse der Elemente sind per Definition ein bestimmter Wert, wenn dieser als ganze Zahl dargestellt ist, z.B. 0 oder 1. Der Wert ist berechnet, wenn er als Gleitpunktzahl dargestellt ist, z.B. 0.0 oder 1.0.

Achtung

Wird `MatrixType` = *'symmetric'*, *'positive_definite'* oder *'upper_triangular'* gewählt, so muss der obere Dreieckteil der Eingabematrix `Matrix` die relevante Information der Matrix beinhalten. Der exakt untere Teil der Matrix wird nicht verwendet. Wird `MatrixType` = *'lower_triangular'* gewählt, so muss der untere Dreieckteil der Eingabematrix `Matrix` die relevante Information der Matrix beinhalten. Der exakt obere Teil der Matrix wird nicht verwendet. Wird `MatrixType` = *'tridiagonal'* gesetzt, wird nur die Hauptdiagonale und die beiden Nebendiagonalen der Eingabematrix `Matrix` genutzt. Die anderen Teile der Matrix werden nicht verwendet. Wenn der verwendete Teil der Eingabematrix `Matrix` nicht vom spezifizierten Typ ist, wird eine Fehlerbehandlung durchgeführt.

Parameter

- ▷ **`MatrixID`** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **`MatrixType`** (input_control) . . . . . string ⇝ *string*
  Der Matrixtyp der Eingabematrix.
  Defaultwert: 'general'
  Werteliste: `MatrixType` ∈ {'general', 'symmetric', 'positive_definite', 'tridiagonal', 'upper_triangular', 'permuted_upper_triangular', 'lower_triangular', 'permuted_lower_triangular'}

▷ **Epsilon** (input_control) . . . . . real ⇝ *real*
Inversionstyp.
Defaultwert: 0.0
Wertevorschläge: `Epsilon` ∈ {0.0, 2.2204e-16}

▷ **MatrixInvID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit der inversen Matrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `invert_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`invert_matrix_mod`

Siehe auch

`transpose_matrix`, `transpose_matrix_mod`

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

invert_matrix_mod (: : MatrixID, MatrixType, Epsilon :)

Invertiert eine Matrix.

Der Operator `invert_matrix_mod` berechnet die Inverse der Matrix `Matrix`. Die Eingabematrix wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich.

Die Inverse wird berechnet, wenn `Epsilon` = 0 ist. Der Matrixtyp wird über `MatrixType` gewählt. Zur Verfügung stehen *'general'* für allgemeine, *'symmetric'* für symmetrische, *'positive_definite'* für symmetrisch positiv definite, *'tridiagonal'* für tridiagonale Matrizen, *'upper_triangular'* für obere, *'permuted_upper_triangular'* für permutierte obere, *'lower_triangular'* für untere und *'permuted_lower_triangular'* für permutierte untere Dreiecksmatrizen.

Beispiel 1:

$$\texttt{Matrix} = \begin{bmatrix} 1.0 & 3.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 5.0 & 5.0 & 7.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'general' \qquad \texttt{Epsilon} = 0$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} -1.25 & 1.50 & -0.75 \\ -0.50 & 2.00 & -1.50 \\ 1.25 & -2.50 & 1.75 \end{bmatrix}$$

Beispiel 2:

$$\texttt{Matrix} = \begin{bmatrix} 1.0 & 3.0 & 3.0 \\ 0 & 2.0 & 6.0 \\ 0 & 0 & 10.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'upper_triangular' \qquad \texttt{Epsilon} = 0$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 1.00 & -1.50 & 0.60 \\ 0 & 0.50 & -0.30 \\ 0 & 0 & 0.10 \end{bmatrix}$$

Beispiel 3:

$$\texttt{Matrix} = \begin{bmatrix} 1.0 & 3.0 & 3.0 \\ 0 & 0 & 10.0 \\ 0 & 2.0 & 6.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'permuted_upper_triangular' \qquad \texttt{Epsilon} = 0$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 1.00 & -1.50 & 0.60 \\ 0 & 0.50 & -0.30 \\ 0 & 0 & 0.10 \end{bmatrix}$$

Die Pseudoinverse wird berechnet, wenn `Epsilon` > 0 gesetzt ist. Alle Singulärwerte, die während der Berechnung entstehen und kleiner als `Epsilon` × dem größten Singulärwert sind, werden auf 0 gesetzt. Für diese Werte wird keine interne Division vorgenommen um einer Division durch 0 vorzubeugen. Als Matrixtyp muss `MatrixType` = *'general'* gesetzt werden.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 & 5.0 \\ -5.0 & 7.0 & 2.0 & -6.0 \\ -9.0 & -4.0 & 1.0 & 4.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'general' \qquad \texttt{Epsilon} = 2.2204\text{e-}16$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} -0.0021 & -0.0482 & -0.0813 \\ 0.1435 & 0.1137 & -0.0137 \\ -0.0519 & -0.0015 & 0.0028 \\ 0.1518 & 0.0056 & 0.0526 \end{bmatrix}$$

Bemerkung: Die relative Genauigkeit der Gleitkommarepräsentation des genutzten Datentyps (double) ist `Epsilon` = 2.2204e-16.

Es ist ebenfalls zu beachten, dass in den Beispielen Unterschiede in der Bedeutung der Werte in den Ausgabematrizen zu finden sind: Die Ergebnisse der Elemente sind per Definition ein bestimmter Wert, wenn dieser als ganze Zahl dargestellt ist, z.B. 0 oder 1. Der Wert ist berechnet, wenn er als Gleitpunktzahl dargestellt ist, z.B. 0.0 oder 1.0.

Achtung

Wird `MatrixType` = *'symmetric'*, *'positive_definite'* oder *'upper_triangular'* gewählt, so muss der obere Dreiecksteil der Eingabematrix `Matrix` die relevante Information der Matrix beinhalten. Der exakt untere Teil der Matrix wird nicht verwendet. Wird `MatrixType` = *'lower_triangular'* gewählt, so muss der untere Dreiecksteil der Eingabematrix `Matrix` die relevante Information der Matrix beinhalten. Der exakt obere Teil der Matrix wird nicht verwendet. Wird `MatrixType` = *'tridiagonal'* gesetzt, wird nur die Hauptdiagonale und die beiden Nebendiagonalen der Eingabematrix `Matrix` genutzt. Die anderen Teile der Matrix werden nicht verwendet. Wenn der verwendete Teil der Eingabematrix `Matrix` nicht vom spezifizierten Typ ist, wird eine Fehlerbehandlung durchgeführt.

`invert_matrix_mod` verändert den Inhalt einer bereits bestehenden Matrix.

Parameter

- ▷ **`MatrixID`** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **`MatrixType`** (input_control) . . . . . string ⇝ *string*
  Der Matrixtyp der Eingabematrix.
  Defaultwert: 'general'
  Werteliste: `MatrixType` ∈ {'general', 'symmetric', 'positive_definite', 'tridiagonal', 'upper_triangular', 'permuted_upper_triangular', 'lower_triangular', 'permuted_lower_triangular'}
- ▷ **`Epsilon`** (input_control) . . . . . real ⇝ *real*
  Inversionstyp.
  Defaultwert: 0.0
  Wertevorschläge: `Epsilon` ∈ {0.0, 2.2204e-16}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `invert_matrix_mod` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`invert_matrix`

Siehe auch

`transpose_matrix`, `transpose_matrix_mod`

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

mult_element_matrix (: : MatrixAID, MatrixBID : MatrixMultID)

Multipliziert Matrizen elementweise.

Der Operator `mult_element_matrix` multipliziert die zwei Eingabematrizen `MatrixA` und `MatrixB` Element für Element. Die Matrizen sind durch die Matrix Handles `MatrixAID` und `MatrixBID` gegeben. Beide Eingabematrizen müssen die gleiche Dimension besitzen. Es wird eine neue Matrix `MatrixMult` generiert und das Ergebnis in dieser gespeichert. Der Operator gibt das Matrix Handle `MatrixMultID` der Matrix `MatrixMult` zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixMult}_{ij} = \texttt{MatrixA}_{ij} \cdot \texttt{MatrixB}_{ij}.$$

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 4.0 & 8.0 & 0.0 \\ -6.0 & 7.0 & 2.0 \\ -9.0 & 4.0 & 1.0 \end{bmatrix} \quad \texttt{MatrixB} = \begin{bmatrix} 2.0 & 8.0 & -3.0 \\ -4.0 & -1.0 & 4.0 \\ -2.0 & -4.0 & 2.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixMult} = \begin{bmatrix} 8.0 & 64.0 & 0.0 \\ 24.0 & -7.0 & 8.0 \\ 18.0 & -16.0 & 2.0 \end{bmatrix}$$

Parameter

▷ **MatrixAID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix A.

▷ **MatrixBID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix B.

▷ **MatrixMultID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den multiplizierten Werten der Eingabematrizen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `mult_element_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`mult_element_matrix_mod`

Siehe auch

`div_element_matrix`, `div_element_matrix_mod`, `scale_matrix`, `scale_matrix_mod`

Modul

Foundation

mult_element_matrix_mod (: : MatrixAID, MatrixBID :)

Multipliziert Matrizen elementweise.

Der Operator `mult_element_matrix_mod` multipliziert die zwei Eingabematrizen `MatrixA` und `MatrixB` Element für Element. Die Matrizen sind durch die Matrix Handles `MatrixAID` und `MatrixBID` gegeben. Beide Eingabematrizen müssen die gleiche Dimension besitzen. Die Eingabematrix `MatrixA` wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixA}_{ij} \quad = \quad \texttt{MatrixA}_{ij} \quad \cdot \quad \texttt{MatrixB}_{ij}.$$

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 4.0 & 8.0 & 0.0 \\ -6.0 & 7.0 & 2.0 \\ -9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 2.0 & 8.0 & -3.0 \\ -4.0 & -1.0 & 4.0 \\ -2.0 & -4.0 & 2.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixA} = \begin{bmatrix} 8.0 & 64.0 & 0.0 \\ 24.0 & -7.0 & 8.0 \\ 18.0 & -16.0 & 2.0 \end{bmatrix}$$

Parameter

▷ **MatrixAID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix A.

▷ **MatrixBID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix B.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `mult_element_matrix_mod` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixAID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`mult_element_matrix`

Siehe auch

`div_element_matrix`, `div_element_matrix_mod`, `scale_matrix`, `scale_matrix_mod`

Modul

Foundation

mult_matrix (: : MatrixAID, MatrixBID, MultType : MatrixMultID)

Multipliziert zwei Matrizen.

Der Operator `mult_matrix` berechnet das Produkt der Eingabematrizen `MatrixA` und `MatrixB`, die durch die Matrix Handles `MatrixAID` und `MatrixBID` definiert sind. Es wird eine neue Matrix `MatrixMult` mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle `MatrixMultID` der Matrix `MatrixMult` zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Optional können eine oder beide Eingabematrizen für die Multiplikation transponiert werden.

Der Multiplikationstyp kann über `MultType` gewählt werden:

'AB': Die Matrizen `MatrixA` und `MatrixB` werden nicht transponiert. Die Formel für die Berechnung lautet:

$$\texttt{MatrixMult} = \texttt{MatrixA} \cdot \texttt{MatrixB}.$$

Die Anzahl der Spalten der Matrix `MatrixA` muss identisch mit der Anzahl der Zeilen der Matrix `MatrixB` sein.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & -3.0 \\ 2.0 & -5.0 \\ -3.0 & 2.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & -3.0 & 1.0 & 1.0 \\ 2.0 & -1.0 & -2.0 & -1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixMult} = \begin{bmatrix} 3.0 & -6.0 & 9.0 & 6.0 \\ -4.0 & -1.0 & 12.0 & 7.0 \\ -5.0 & 7.0 & -7.0 & -5.0 \end{bmatrix}$$

'ATB': Die Matrix `MatrixA` wird transponiert. Die Matrix `MatrixB` wird nicht transponiert. Die Formel für die Berechnung lautet:

$$\texttt{MatrixMult} \quad = \quad \texttt{MatrixA}^T \quad \cdot \quad \texttt{MatrixB}.$$

Die Anzahl der Zeilen der Matrix `MatrixA` muss identisch mit der Anzahl der Zeilen der Matrix `MatrixB` sein.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & 2.0 & -3.0 \\ -3.0 & -5.0 & 2.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & -3.0 & 1.0 & 1.0 \\ 2.0 & -1.0 & -2.0 & -1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixMult} = \begin{bmatrix} 3.0 & -6.0 & 9.0 & 6.0 \\ -4.0 & -1.0 & 12.0 & 7.0 \\ -5.0 & 7.0 & -7.0 & -5.0 \end{bmatrix}$$

'ABT': Die Matrix `MatrixA` wird nicht transponiert. Die Matrix `MatrixB` wird transponiert. Die Formel für die Berechnung lautet:

$$\texttt{MatrixMult} \quad = \quad \texttt{MatrixA} \quad \cdot \quad \texttt{MatrixB}^T.$$

Die Anzahl der Spalten der Matrix `MatrixA` muss identisch mit der Anzahl der Spalten der Matrix `MatrixB` sein.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & -3.0 \\ 2.0 & -5.0 \\ -3.0 & 2.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & 2.0 \\ -3.0 & -1.0 \\ 1.0 & -2.0 \\ 1.0 & -1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixMult} = \begin{bmatrix} 3.0 & -6.0 & 9.0 & 6.0 \\ -4.0 & -1.0 & 12.0 & 7.0 \\ -5.0 & 7.0 & -7.0 & -5.0 \end{bmatrix}$$

'ATBT': Die Matrizen `MatrixA` und `MatrixB` werden transponiert. Die Formel für die Berechnung lautet:

$$\texttt{MatrixMult} \quad = \quad \texttt{MatrixA}^T \quad \cdot \quad \texttt{MatrixB}^T.$$

Die Anzahl der Zeilen der Matrix `MatrixA` muss identisch mit der Anzahl der Spalten der Matrix `MatrixB` sein.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & 2.0 & -3.0 \\ -3.0 & -5.0 & 2.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & 2.0 \\ -3.0 & -1.0 \\ 1.0 & -2.0 \\ 1.0 & -1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixMult} = \begin{bmatrix} 3.0 & -6.0 & 9.0 & 6.0 \\ -4.0 & -1.0 & 12.0 & 7.0 \\ -5.0 & 7.0 & -7.0 & -5.0 \end{bmatrix}$$

Parameter

▷ **MatrixAID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix A.

▷ **MatrixBID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix B.

▷ **MultType** (input_control) . . . . . string ⇝ *string*
Typ der Eingabematrix.
Defaultwert: 'AB'
Werteliste: `MultType` ∈ {'AB', 'ATB', 'ABT', 'ATBT'}

▷ **MatrixMultID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle der multiplizierten Matrizen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `mult_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`mult_matrix_mod`

Siehe auch

`mult_element_matrix`, `mult_element_matrix_mod`, `div_element_matrix`, `div_element_matrix_mod`, `transpose_matrix`, `transpose_matrix_mod`

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

mult_matrix_mod (: : MatrixAID, MatrixBID, MultType :)

Multipliziert zwei Matrizen.

Der Operator `mult_matrix_mod` berechnet das Produkt der Eingabematrizen `MatrixA` und `MatrixB`, die durch die Matrix Handles `MatrixAID` und `MatrixBID` definiert sind. Die Eingabematrix `MatrixA` wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Optional können eine oder beide Eingabematrizen für die Multiplikation transponiert werden.

Der Multiplikationstyp kann über `MultType` gewählt werden:

'AB': Die Matrizen `MatrixA` und `MatrixB` werden nicht transponiert. Die Formel für die Berechnung lautet:

$$\texttt{MatrixA} = \texttt{MatrixA} \cdot \texttt{MatrixB}.$$

Die Anzahl der Spalten der Matrix `MatrixA` muss identisch mit der Anzahl der Zeilen der Matrix `MatrixB` sein.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & -3.0 \\ 2.0 & -5.0 \\ -3.0 & 2.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & -3.0 & 1.0 & 1.0 \\ 2.0 & -1.0 & -2.0 & -1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixA} = \begin{bmatrix} 3.0 & -6.0 & 9.0 & 6.0 \\ -4.0 & -1.0 & 12.0 & 7.0 \\ -5.0 & 7.0 & -7.0 & -5.0 \end{bmatrix}$$

'ATB': Die Matrix `MatrixA` wird transponiert. Die Matrix `MatrixB` wird nicht transponiert. Die Formel für die Berechnung lautet:

$$\texttt{MatrixA} = \texttt{MatrixA}^T \cdot \texttt{MatrixB}.$$

Die Anzahl der Zeilen der Matrix `MatrixA` muss identisch mit der Anzahl der Zeilen der Matrix `MatrixB` sein.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & 2.0 & -3.0 \\ -3.0 & -5.0 & 2.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & -3.0 & 1.0 & 1.0 \\ 2.0 & -1.0 & -2.0 & -1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixA} = \begin{bmatrix} 3.0 & -6.0 & 9.0 & 6.0 \\ -4.0 & -1.0 & 12.0 & 7.0 \\ -5.0 & 7.0 & -7.0 & -5.0 \end{bmatrix}$$

'ABT': Die Matrix `MatrixA` wird nicht transponiert. Die Matrix `MatrixB` wird transponiert. Die Formel für die Berechnung lautet:

$$\texttt{MatrixA} \quad = \quad \texttt{MatrixA} \quad \cdot \quad \texttt{MatrixB}^T.$$

Die Anzahl der Spalten der Matrix `MatrixA` muss identisch mit der Anzahl der Spalten der Matrix `MatrixB` sein.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & -3.0 \\ 2.0 & -5.0 \\ -3.0 & 2.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & 2.0 \\ -3.0 & -1.0 \\ 1.0 & -2.0 \\ 1.0 & -1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixA} = \begin{bmatrix} 3.0 & -6.0 & 9.0 & 6.0 \\ -4.0 & -1.0 & 12.0 & 7.0 \\ -5.0 & 7.0 & -7.0 & -5.0 \end{bmatrix}$$

'ATBT': Die Matrizen `MatrixA` und `MatrixB` werden transponiert. Die Formel für die Berechnung lautet:

$$\texttt{MatrixA} \quad = \quad \texttt{MatrixA}^T \quad \cdot \quad \texttt{MatrixB}^T.$$

Die Anzahl der Zeilen der Matrix `MatrixA` muss identisch mit der Anzahl der Spalten der Matrix `MatrixB` sein.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & 2.0 & -3.0 \\ -3.0 & -5.0 & 2.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & 2.0 \\ -3.0 & -1.0 \\ 1.0 & -2.0 \\ 1.0 & -1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixA} = \begin{bmatrix} 3.0 & -6.0 & 9.0 & 6.0 \\ -4.0 & -1.0 & 12.0 & 7.0 \\ -5.0 & 7.0 & -7.0 & -5.0 \end{bmatrix}$$

Parameter

- ▷ **`MatrixAID`** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix A.
- ▷ **`MatrixBID`** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix B.
- ▷ **`MultType`** (input_control) . . . . . string ⇝ *string*
  Typ der Eingabematrix.
  Defaultwert: 'AB'
  Werteliste: `MultType` ∈ {'AB', 'ATB', 'ABT', 'ATBT'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `mult_matrix_mod` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixAID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`mult_matrix`

Siehe auch

`mult_element_matrix`, `mult_element_matrix_mod`, `div_element_matrix`, `div_element_matrix_mod`, `transpose_matrix`, `transpose_matrix_mod`

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

pow_element_matrix (: : MatrixID, MatrixExpID : MatrixPowID)

Berechnet die Potenz der Elemente einer Matrix.

Der Operator `pow_element_matrix` berechnet die Potenz aller Elemente der Eingabematrix `Matrix` mit den Elementen der Eingabematrix `MatrixExp`. Die Eingabematrizen `Matrix` und `MatrixExp` sind durch die Matrix Handles `MatrixID` und `MatrixExpID` gegeben. Beide Eingabematrizen müssen die gleichen Dimensionen besitzen. Es wird eine neue Matrix `MatrixPow` mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle `MatrixPowID` der Matrix `MatrixPow` zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixPow}_{ij} \quad = \quad \texttt{Matrix}_{ij}^{\texttt{MatrixExp}_{ij}}.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \texttt{MatrixExp} = \begin{bmatrix} 1.0 & 2.0 & 2.0 \\ 2.0 & 0.0 & 3.0 \\ 0.5 & 3.0 & 1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{MatrixPow} = \begin{bmatrix} 3.0 & 1.0 & 4.0 \\ 25.0 & 1.0 & 8.0 \\ 3.0 & 64.0 & 1.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix der Basis.

▷ **MatrixExpID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix mit den Exponenten.

▷ **MatrixPowID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den potenzierten Werten der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `pow_element_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`pow_element_matrix_mod`, `pow_scalar_element_matrix`, `pow_scalar_element_matrix_mod`

Siehe auch

`sqrt_matrix`, `sqrt_matrix_mod`

Modul

Foundation

pow_element_matrix_mod (: : MatrixID, MatrixExpID :)

Berechnet die Potenz der Elemente einer Matrix.

Der Operator `pow_element_matrix_mod` berechnet die Potenz aller Elemente der Eingabematrix `Matrix` mit den Elementen der Eingabematrix `MatrixExp`. Die Eingabematrizen `Matrix` und `MatrixExp` sind durch die Matrix Handles `MatrixID` und `MatrixExpID` gegeben. Beide Eingabematrizen müssen die gleichen Dimensionen besitzen. Die Eingabematrix `MatrixID` wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{Matrix}_{ij} \quad = \quad \texttt{Matrix}_{ij}^{\texttt{MatrixExp}_{ij}}.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \texttt{MatrixExp} = \begin{bmatrix} 1.0 & 2.0 & 2.0 \\ 2.0 & 0.0 & 3.0 \\ 0.5 & 3.0 & 1.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 4.0 \\ 25.0 & 1.0 & 8.0 \\ 3.0 & 64.0 & 1.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix der Basis.

▷ **MatrixExpID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix mit den Exponenten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `pow_element_matrix_mod` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`pow_element_matrix`, `pow_scalar_element_matrix`, `pow_scalar_element_matrix_mod`

Siehe auch

`sqrt_matrix`, `sqrt_matrix_mod`

Modul

Foundation

pow_matrix (: : MatrixID, MatrixType, Power : MatrixPowID)

Berechnet die Potenz einer Matrix.

Der Operator `pow_matrix` berechnet die Potenz der Eingabematrix `Matrix` mit einem konstanten Wert. Die Eingabematrix `Matrix` ist durch das Matrix Handle `MatrixID` gegeben. Die Potenz `Power` ist durch den Parameter `Power` gegeben. Es wird eine neue Matrix `MatrixPow` mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle `MatrixPowID` der Matrix `MatrixPow` zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich.

Der Matrixtyp wird über `MatrixType` gewählt. Zur Verfügung stehen *'general'* für allgemeine, *'symmetric'* für symmetrische, *'positive_definite'* für symmetrisch positiv definite, *'tridiagonal'* für tridiagonale Matrizen, *'upper_triangular'* für obere, *'permuted_upper_triangular'* für permutierte obere, *'lower_triangular'* für untere und *'permuted_lower_triangular'* für permutierte untere Dreiecksmatrizen. Die Formel für die Berechnung lautet:

$$\texttt{MatrixPow} \quad = \quad \texttt{Matrix}^{\texttt{Power}}.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \texttt{Power} = \begin{bmatrix} 2.0 \end{bmatrix} \qquad \texttt{MatrixType} = 'general'$$

$$\rightarrow \qquad \texttt{MatrixPow} = \begin{bmatrix} 32.0 & 18.0 & 10.0 \\ 68.0 & 62.0 & 26.0 \\ 56.0 & 41.0 & 27.0 \end{bmatrix}$$

Achtung

Wird `MatrixType` = *'symmetric'*, *'positive_definite'* oder *'upper_triangular'* gewählt, so muss der obere Dreiecksteil der Eingabematrix `Matrix` die relevante Information der Matrix beinhalten. Der exakt untere Teil der Matrix wird nicht verwendet. Wird `MatrixType` = *'lower_triangular'* gewählt, so muss der untere Dreiecksteil der Eingabematrix `Matrix` die relevante Information der Matrix beinhalten. Der exakt obere Teil der Matrix wird nicht verwendet. Wenn der verwendete Teil der Eingabematrix `Matrix` nicht vom spezifizierten Typ ist, wird eine Fehlerbehandlung durchgeführt.

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **MatrixType** (input_control) . . . . . string ⇝ *string*
  Der Matrixtyp der Eingabematrix.
  Defaultwert: 'general'
  Werteliste: MatrixType ∈ {'general', 'symmetric', 'positive_definite', 'upper_triangular', 'permuted_upper_triangular', 'lower_triangular', 'permuted_lower_triangular'}
- ▷ **Power** (input_control) . . . . . number ⇝ *real* / integer
  Die Potenz.
  Defaultwert: 2.0
  Wertevorschläge: Power ∈ {0.1, 0.2, 0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 10.0}
- ▷ **MatrixPowID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit der potenzierten Matrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert pow_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

pow_matrix_mod, eigenvalues_symmetric_matrix, eigenvalues_general_matrix

Siehe auch

sqrt_matrix, sqrt_matrix_mod

Modul

Foundation

pow_matrix_mod (: : MatrixID, MatrixType, Power :)

Berechnet die Potenz einer Matrix.

Der Operator pow_matrix_mod berechnet die Potenz der Eingabematrix Matrix mit einem konstanten Wert. Die Eingabematrix Matrix ist durch das Matrix Handle MatrixID gegeben. Die Potenz Power ist durch den Parameter Power gegeben. Die Eingabematrix wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Der Matrixtyp wird über MatrixType gewählt. Zur Verfügung stehen *'general'* für allgemeine, *'symmetric'* für symmetrische, *'positive_definite'* für symmetrisch positiv definite, *'tridiagonal'* für tridiagonale Matrizen, *'upper_triangular'* für obere, *'permuted_upper_triangular'* für permutierte obere, *'lower_triangular'* für untere und *'permuted_lower_triangular'* für permutierte untere Dreiecksmatrizen. Die Formel für die Berechnung lautet:

$$\mathtt{Matrix} \;=\; \mathtt{Matrix}^{\mathtt{Power}}.$$

Beispiel:

$$\mathtt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \mathtt{Power} = \begin{bmatrix} 2.0 \end{bmatrix} \qquad \mathtt{MatrixType} = 'general'$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 32.0 & 18.0 & 10.0 \\ 68.0 & 62.0 & 26.0 \\ 56.0 & 41.0 & 27.0 \end{bmatrix}$$

Achtung

Wird `MatrixType` = *'symmetric'*, *'positive_definite'* oder *'upper_triangular'* gewählt, so muss der obere Dreieckteil der Eingabematrix `Matrix` die relevante Information der Matrix beinhalten. Der exakt untere Teil der Matrix wird nicht verwendet. Wird `MatrixType` = *'lower_triangular'* gewählt, so muss der untere Dreieckteil der Eingabematrix `Matrix` die relevante Information der Matrix beinhalten. Der exakt obere Teil der Matrix wird nicht verwendet. Wenn der verwendete Teil der Eingabematrix `Matrix` nicht vom spezifizierten Typ ist, wird eine Fehlerbehandlung durchgeführt.

`pow_matrix_mod` verändert den Inhalt einer bereits bestehenden Matrix.

Parameter

- ▷ **`MatrixID`** (input_control) ..... matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **`MatrixType`** (input_control) ..... string ⇝ *string*
  Der Matrixtyp der Eingabematrix.
  Defaultwert: 'general'
  Werteliste: `MatrixType` ∈ {'general', 'symmetric', 'positive_definite', 'upper_triangular', 'permuted_upper_triangular', 'lower_triangular', 'permuted_lower_triangular'}
- ▷ **`Power`** (input_control) ..... number ⇝ *real* / integer
  Die Potenz.
  Defaultwert: 2.0
  Wertevorschläge: `Power` ∈ {0.1, 0.2, 0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 10.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `pow_matrix_mod` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`pow_matrix`, `eigenvalues_symmetric_matrix`, `eigenvalues_general_matrix`

Siehe auch

`sqrt_matrix`, `sqrt_matrix_mod`

Modul

Foundation

`pow_scalar_element_matrix` (: : MatrixID, Power : MatrixPowID)

Berechnet die Potenz der Elemente einer Matrix.

Der Operator pow_scalar_element_matrix berechnet die Potenz aller Elemente der Eingabematrix Matrix mit einem konstanten Wert. Die Eingabematrix Matrix ist durch das Matrix Handle MatrixID gegeben. Die Potenz Power ist durch den Parameter Power gegeben. Es wird eine neue Matrix MatrixPow mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle MatrixPowID der Matrix MatrixPow zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die Formel für die Berechnung lautet:

$$\mathtt{MatrixPow}_{ij} \quad = \quad \mathtt{Matrix}_{ij}{}^{\mathtt{Power}}.$$

Beispiel:

$$\mathtt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \mathtt{Power} = \begin{bmatrix} 2.0 \end{bmatrix}$$

$$\rightarrow \qquad \mathtt{MatrixPow} = \begin{bmatrix} 9.0 & 1.0 & 4.0 \\ 25.0 & 49.0 & 4.0 \\ 81.0 & 16.0 & 1.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **Power** (input_control) . . . . . number ⇝ *real* / integer
  Die Potenz.
  Defaultwert: 2.0
  Wertevorschläge: Power ∈ {0.1, 0.2, 0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 10.0}
- ▷ **MatrixPowID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit den potenzierten Werten der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert pow_scalar_element_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

pow_scalar_element_matrix_mod, pow_element_matrix, pow_element_matrix_mod

Siehe auch

sqrt_matrix, sqrt_matrix_mod

Modul

Foundation

pow_scalar_element_matrix_mod (: : MatrixID, Power :)

Berechnet die Potenz der Elemente einer Matrix.

Der Operator pow_scalar_element_matrix_mod berechnet die Potenz aller Elemente der Eingabematrix Matrix mit einem konstanten Wert. Die Eingabematrix Matrix ist durch das Matrix Handle MatrixID gegeben. Die Potenz Power ist durch den Parameter Power gegeben. Die Eingabematrix wird mit dem Ergebnis

überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die Formel für die Berechnung lautet:

$$\texttt{Matrix}_{ij} \quad = \quad \texttt{Matrix}_{ij}^{\texttt{Power}}.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix} \qquad \texttt{Power} = \begin{bmatrix} 2.0 \end{bmatrix}$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 9.0 & 1.0 & 4.0 \\ 25.0 & 49.0 & 4.0 \\ 81.0 & 16.0 & 1.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **Power** (input_control) . . . . . number ⇝ *real* / integer
Die Potenz.
Defaultwert: 2.0
Wertevorschläge: Power ∈ {0.1, 0.2, 0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 10.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert pow_scalar_element_matrix_mod den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

pow_scalar_element_matrix, pow_element_matrix, pow_element_matrix_mod

Siehe auch

sqrt_matrix, sqrt_matrix_mod

Modul

Foundation

scale_matrix (: : MatrixID, Factor : MatrixScaledID)

Skaliert eine Matrix.

Der Operator scale_matrix skaliert die Matrix Matrix mit einem konstanten Wert. Die Matrix ist durch das Matrix Handle MatrixID gegeben. Es wird eine neue Matrix MatrixScaled mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle MatrixScaledID zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixScaled}_{ij} = \texttt{Factor} \cdot \texttt{Matrix}_{ij}.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -2.0 & 6.0 & 2.0 \\ -5.0 & -4.0 & 1.0 \end{bmatrix} \qquad \texttt{Factor} = 1.5$$

$$\rightarrow \qquad \texttt{MatrixScaled} = \begin{bmatrix} 4.5 & 1.5 & -3.0 \\ -3.0 & 9.0 & 3.0 \\ -7.5 & -6.0 & 1.5 \end{bmatrix}$$

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **Factor** (input_control) . . . . . number ⇝ *real* / integer
  Skalierungsfaktor.
  Defaultwert: 2.0
  Wertevorschläge: Factor ∈ {0.1, 0.2, 0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 10.0}
- ▷ **MatrixScaledID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit den skalierten Elementen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert scale_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

scale_matrix_mod

Siehe auch

mult_element_matrix, mult_element_matrix_mod, div_element_matrix, div_element_matrix_mod

Modul

Foundation

scale_matrix_mod (: : MatrixID, Factor :)

Skaliert eine Matrix.

Der Operator scale_matrix_mod skaliert die Matrix Matrix mit einem konstanten Wert. Die Matrix ist durch das Matrix Handle MatrixID gegeben. Die Eingabematrix wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die Formel für die Berechnung lautet:

$$\texttt{Matrix}_{ij} = \texttt{Factor} \cdot \texttt{Matrix}_{ij}.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -2.0 & 6.0 & 2.0 \\ -5.0 & -4.0 & 1.0 \end{bmatrix} \qquad \texttt{Factor} = 1.5$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 4.5 & 1.5 & -3.0 \\ -3.0 & 9.0 & 3.0 \\ -7.5 & -6.0 & 1.5 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **Factor** (input_control) . . . . . number ⇝ *real* / integer
Skalierungsfaktor.
Defaultwert: 2.0
Wertevorschläge: Factor ∈ {0.1, 0.2, 0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 10.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert scale_matrix_mod den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

scale_matrix

Siehe auch

mult_element_matrix, mult_element_matrix_mod, div_element_matrix, div_element_matrix_mod

Modul

Foundation

solve_matrix (: : MatrixLHSID, MatrixLHSType, Epsilon, MatrixRHSID : MatrixResultID)

Berechnet die Lösung eines Gleichungssystems.

Der Operator solve_matrix berechnet die Lösung eines linearen Gleichungssystems oder eines linearen Kleinsten-Quadrate Problems. Die Eingabematrizen MatrixLHS und MatrixRHS sind durch die Matrix Handles MatrixLHSID und MatrixRHSID gegeben. Die Anzahl der Zeilen der Matrizen MatrixLHS und MatrixRHS müssen identisch sein. Der Operator gibt das Matrix Handle MatrixResultID der Matrix MatrixResult zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Für lineare Gleichungssysteme werden die Gleichungen

$$\texttt{MatrixLHS} \cdot \texttt{MatrixResult} = \texttt{MatrixRHS}$$

gelöst. Hierfür muss die Matrix MatrixLHS quadratisch und der Parameter Epsilon 0 sein. Der Matrixtyp der Matrix MatrixLHS kann über den Parameter MatrixLHSType gewählt werden. Die folgenden Werte stehen zur Verfügung: *'general'* für allgemeine, *'symmetric'* für symmetrische, *'positive_definite'* für symmetrisch positiv definite und *'tridiagonal'* für tridiagonale Matrizen, *'upper_triangular'* für obere, *'permuted_upper_triangular'* für permutierte obere, *'lower_triangular'* für untere und *'permuted_lower_triangular'* für permutierte untere Dreiecksmatrizen.

Beispiel:

$$\texttt{MatrixLHS} = \begin{bmatrix} 6.0 & 5.0 & 3.0 \\ 5.0 & 7.0 & 3.0 \\ 3.0 & 3.0 & 4.0 \end{bmatrix} \qquad \texttt{MatrixRHS} = \begin{bmatrix} -1.0 & 6.0 \\ 3.0 & -3.0 \\ 5.0 & 4.0 \end{bmatrix}$$

$$\texttt{MatrixLHSType} = \text{'}positive_definite\text{'} \qquad \texttt{Epsilon} = 0$$

$$\rightarrow \qquad \texttt{MatrixResult} = \begin{bmatrix} -2.0 & 3.0 \\ 1.0 & -3.0 \\ 2.0 & 1.0 \end{bmatrix}$$

Für ein lineares Kleinste-Quadrate Problem oder die Wahl Epsilon ungleich 0 muss die Matrix MatrixLHS nicht quadratisch sein. Die Kleinste-Quadrate Lösung wird mittels einer Singulärwertzerlegung der Matrix MatrixLHS durch Minimieren von

$$||\texttt{MatrixRHS} - \texttt{MatrixLHS} \cdot \texttt{MatrixResult}||$$

berechnet. Alle Singulärwerte, die kleiner als Epsilon $\times$ dem größten Singulärwert sind, werden 0 gesetzt. Für diese Werte wird keine interne Division vorgenommen um einer Division durch 0 vorzubeugen. Ebenfalls darf die Matrix MatrixLHS in diesem Fall einen Rangdefekt aufweisen. Als Matrixtyp muss MatrixLHSType = *'general'* gewählt werden.

Beispiel:

$$\texttt{MatrixLHS} = \begin{bmatrix} 6.0 & 5.0 & 3.0 \\ 3.0 & 7.0 & -3.0 \\ 5.0 & 12.0 & 4.0 \\ 5.0 & 4.0 & 12.0 \\ 4.0 & 6.0 & 8.0 \end{bmatrix} \qquad \texttt{MatrixRHS} = \begin{bmatrix} 29.0 \\ 10.0 \\ 35.0 \\ 43.0 \\ 25.0 \end{bmatrix}$$

$$\texttt{MatrixLHSType} = \text{'}general\text{'} \qquad \texttt{Epsilon} = 2.2204e-16$$

$$\rightarrow \qquad \texttt{MatrixResult} = \begin{bmatrix} 3.4914 \\ 0.7114 \\ 1.6213 \end{bmatrix}$$

Bemerkung: Die relative Genauigkeit der Gleitkommarepräsentation des genutzten Datentyps (double) ist Epsilon = 2.2204e-16.

Achtung

Wird MatrixLHSType = *'symmetric'*, *'positive_definite'* oder *'upper_triangular'* gewählt, so muss der obere Dreiecksteil der Eingabematrix MatrixLHS die relevante Information der Matrix beinhalten. Der exakt untere Teil der Matrix wird nicht verwendet. Wird MatrixLHSType = *'lower_triangular'* gewählt, so muss der untere Dreiecksteil der Eingabematrix MatrixLHS die relevante Information der Matrix beinhalten. Der exakt obere Teil der Matrix wird nicht verwendet. Wird MatrixLHSType = *'tridiagonal'* gesetzt, wird nur die Hauptdiagonale und die beiden Nebendiagonalen der Eingabematrix MatrixLHS genutzt. Die anderen Teile der Matrix werden nicht verwendet. Wenn der verwendete Teil der Eingabematrix MatrixLHS nicht vom spezifizierten Typ ist, wird eine Fehlerbehandlung durchgeführt.

Parameter

▷ **MatrixLHSID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix der linken Seite.

▷ **MatrixLHSType** (input_control) . . . . . string ⇝ *string*
Der Matrixtyp der Eingabematrix der linken Seite.
Defaultwert: 'general'
Werteliste: `MatrixLHSType` ∈ {'general', 'symmetric', 'positive_definite', 'tridiagonal', 'upper_triangular', 'permuted_upper_triangular', 'lower_triangular', 'permuted_lower_triangular'}

▷ **Epsilon** (input_control) . . . . . real ⇝ *real*
Lösungstyp und Grenze zum Setzen der Singulärwerte zu 0.
Defaultwert: 0.0
Wertevorschläge: `Epsilon` ∈ {0.0, 2.2204e-16}

▷ **MatrixRHSID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix der rechten Seite.

▷ **MatrixResultID** (output_control) . . . . . matrix ⇝ *handle*
Neues Matrix Handle mit der Lösung.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `solve_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

sqrt_matrix (: : MatrixID : MatrixSqrtID)

Berechnet die Quadratwurzel der Elemente einer Matrix.

Der Operator `sqrt_matrix` berechnet die Quadratwurzel aller Elemente der Eingabematrix `Matrix`, die durch das Matrix Handle `MatrixID` gegeben ist. Es wird eine neue Matrix `MatrixSqrt` mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle `MatrixSqrtID` der Matrix `MatrixSqrt` zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixSqrt}_{ij} \quad = \quad \sqrt{\texttt{Matrix}_{ij}}.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \texttt{MatrixSqrt} = \begin{bmatrix} 1.7321 & 1.0000 & 1.4142 \\ 2.2361 & 2.6458 & 1.4142 \\ 3.0000 & 2.0000 & 1.0000 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **MatrixSqrtID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den Beträgen der Quadratwurzel der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `sqrt_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`sqrt_matrix_mod`

Siehe auch

`pow_scalar_element_matrix`, `pow_scalar_element_matrix_mod`

Modul

Foundation

sqrt_matrix_mod (: : MatrixID :)

Berechnet die Quadratwurzel der Elemente einer Matrix.

Der Operator `sqrt_matrix_mod` berechnet die Quadratwurzel aller Elemente der Eingabematrix `Matrix`, die durch das Matrix Handle `MatrixID` gegeben ist. Die Eingabematrix wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\mathtt{Matrix}_{ij} \quad = \quad \sqrt{\mathtt{Matrix}_{ij}}.$$

Beispiel:

$$\mathtt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ 5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \mathtt{Matrix} = \begin{bmatrix} 1.7321 & 1.0000 & 1.4142 \\ 2.2361 & 2.6458 & 1.4142 \\ 3.0000 & 2.0000 & 1.0000 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `sqrt_matrix_mod` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`sqrt_matrix`

Modul

Foundation

sub_matrix (: : MatrixAID, MatrixBID : MatrixSubID)

Subtraktion zweier Matrizen.

Der Operator `sub_matrix` berechnet die Differenz der Eingabematrizen `MatrixA` und `MatrixB`, die durch die Matrix Handles `MatrixAID` und `MatrixBID` gegeben sind. Beide Eingabematrizen müssen die gleichen Dimensionen besitzen. Es wird eine neue Matrix `MatrixSub` mit dem Ergebnis generiert. Der Operator gibt das Matrix Handle `MatrixSubID` der Matrix `MatrixSub` zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixSub} = \texttt{MatrixA} - \texttt{MatrixB}.$$

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \texttt{MatrixB} = \begin{bmatrix} 2.0 & 8.0 & -3.0 \\ -4.0 & -1.0 & 5.0 \\ 2.0 & -4.0 & 7.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixSub} = \begin{bmatrix} 1.0 & -7.0 & 1.0 \\ -1.0 & 8.0 & -3.0 \\ -11.0 & 0.0 & -6.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixAID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix A.
- ▷ **MatrixBID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix B.
- ▷ **MatrixSubID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit den Differenzen der Eingabematrizen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `sub_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

sub_matrix_mod

Siehe auch

add_matrix, add_matrix_mod

Modul

Foundation

sub_matrix_mod (: : MatrixAID, MatrixBID :)

Subtraktion zweier Matrizen.

Der Operator sub_matrix_mod berechnet die Differenz der Eingabematrizen MatrixA und MatrixB, die durch die Matrix Handles MatrixAID und MatrixBID gegeben sind. Beide Eingabematrizen müssen die gleichen Dimensionen besitzen. Die Eingabematrix MatrixA wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die Formel für die Berechnung lautet:

$$\texttt{MatrixA} = \texttt{MatrixA} - \texttt{MatrixB}.$$

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \texttt{MatrixB} = \begin{bmatrix} 2.0 & 8.0 & -3.0 \\ -4.0 & -1.0 & 5.0 \\ 2.0 & -4.0 & 7.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixA} = \begin{bmatrix} 1.0 & -7.0 & 1.0 \\ -1.0 & 8.0 & -3.0 \\ -11.0 & 0.0 & -6.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixAID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix A.
- ▷ **MatrixBID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix B.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert sub_matrix_mod den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixAID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

sub_matrix

Siehe auch

add_matrix, add_matrix_mod

Modul

Foundation

transpose_matrix (: : MatrixID : MatrixTransposedID)

Transponiert eine Matrix.

Der Operator transpose_matrix gibt die Transponierte der Eingabematrix Matrix zurück. Die Eingabematrix ist durch das Matrix Handle MatrixID definiert. Es wird eine neue Matrix MatrixTransposed mit dem Ergebnis generiert und das Matrix Handle MatrixTransposedID dieser Matrix zurückgegeben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \texttt{MatrixTransposed} = \begin{bmatrix} 3.0 & -5.0 & -9.0 \\ 1.0 & 7.0 & -4.0 \\ -2.0 & 2.0 & 1.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixID** (input_control) ..... matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **MatrixTransposedID** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit der Transponierten der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert transpose_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

transpose_matrix_mod

Siehe auch

invert_matrix, invert_matrix_mod

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

transpose_matrix_mod (: : MatrixID :)

Transponiert eine Matrix.

Der Operator transpose_matrix_mod gibt die Transponierte der Eingabematrix Matrix zurück. Die Eingabematrix ist durch das Matrix Handle MatrixID definiert. Die Eingabematrix wird mit dem Ergebnis überschrieben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 3.0 & -5.0 & -9.0 \\ 1.0 & 7.0 & -4.0 \\ -2.0 & 2.0 & 1.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert transpose_matrix_mod den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

transpose_matrix

Siehe auch

invert_matrix, invert_matrix_mod

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

18.2 Datei

deserialize_matrix (: : SerializedItemHandle : MatrixID)

Deserialisieren einer serialisierten Matrix.

deserialize_matrix deserialisiert eine Matrix, welche mit serialize_matrix serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte Matrix wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einer dafür

automatisch erzeugten Matrix mit dem Handle MatrixID gespeichert. Zugriff auf die Elemente einer Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **MatrixID** (output_control) ..... matrix ⇝ *handle*
Matrix handle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_matrix

Nachfolger

get_full_matrix

Modul

Foundation

read_matrix (: : FileName : MatrixID)

Einlesen eine Matrix von einer Datei.

read_matrix liest eine Matrix, die mit write_matrix geschrieben wurde, aus der Datei FileName ein. Die Dateiendung für die Matrix ist in HALCON 'mtx'. Das Dateiformat kann binär oder ASCII sein. Jede Zeile der Datei enthält eine Zeile der Matrix mit einer Anzahl von Spalten. Die Spalten sind durch Leerzeichen oder Tabulatoren getrennt. Jede Zeile muss dieselbe Anzahl an Spalten beinhalten. Leere Zeilen werden ignoriert.

Die eingelesenen Ergebnisse werden in der generierten Matrix Matrix gespeichert. Es wird das Matrix Handle MatrixID dieser Matrix zurückgegeben. Zugriff auf die Elemente der Matrizen ist z.B. mit dem Operator get_full_matrix möglich.

Parameter

▷ **FileName** (input_control) ..... filename.read ⇝ *string*
Name der Datei.
Dateiendung: .mtx

▷ **MatrixID** (output_control) ..... matrix ⇝ *handle*
Matrix Handle.

Ergebnis

Ist der Dateiname korrekt, dann liefert read_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`get_full_matrix`

Modul

Foundation

serialize_matrix (: : MatrixID : SerializedItemHandle)

Serialisieren einer Matrix.

`serialize_matrix` serialisiert die Daten einer Matrix (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_matrix` in eine Datei geschrieben werden. Die Matrix wird in dem Handle `MatrixID` übergeben. Die serialisierte Matrix wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_matrix` wieder deserialisiert werden.

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle.
- ▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_matrix`

Modul

Foundation

write_matrix (: : MatrixID, FileFormat, FileName :)

Schreiben einer Matrix auf eine Datei.

`write_matrix` schreibt eine Matrix in die Datei `FileName`. Die Matrix kann mit `read_matrix` wieder eingelesen werden. Die Dateiendung für die Matrix ist in HALCON 'mtx'. Der Dateityp kann über den Parameter `FileFormat` gewählt werden. Die folgenden Werte stehen zur Verfügung: *'binary'* für ein binäres Dateiformat und *'ascii'* für ein Textformat. Im Textformat enthält jede Zeile der Datei eine Zeile der Matrix. Die Spalten sind durch Leerzeichen getrennt. Der Vorteil bei der Nutzung des binären Dateiformats gegenüber dem Textformat liegt am geringeren Speicherbedarf der Ausgabedatei.

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **FileFormat** (input_control) . . . . . string ⇝ *string*
  Dateiformat.
  Defaultwert: 'binary'
  Werteliste: `FileFormat` ∈ {'binary', 'ascii'}

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: .mtx

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert write_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Modul

Foundation

18.3 Eigenwerte

eigenvalues_general_matrix (: : MatrixID,
ComputeEigenvectors : EigenvaluesRealID, EigenvaluesImagID,
EigenvectorsRealID, EigenvectorsImagID)

Berechnet die Eigenwerte und optional die Eigenvektoren einer allgemeinen Matrix.

Der Operator eigenvalues_general_matrix berechnet alle Eigenwerte und optional die rechten oder linken Eigenvektoren der quadratischen, allgemeinen Eingangsmatrix Matrix. Die Eingangsmatrix ist durch das Matrix Handle MatrixID definiert. Die berechneten Eigenvektoren sind auf die Länge 1 normiert.

Der Operator generiert die neuen Matrizen EigenvaluesReal und EigenvaluesImag mit den Real- und Imaginärteilen der berechneten Eigenwerte. Jede Matrix besteht aus einer Spalte und n Zeilen, wobei n die Anzahl der Zeilen der Eingabematrix Matrix ist. Im Gegensatz zu dem Operator eigenvalues_symmetric_matrix ist die Reihenfolge der Eigenwerte nicht definiert. Der Operator gibt die Matrix Handles EigenvaluesRealID und EigenvaluesImagID zurück. Die Real- und Imaginärteile der berechneten Eigenvektoren werden, falls gewünscht, in den neuen Matrizen EigenvectorsReal and EigenvectorsImag gespeichert. In diesem Fall werden gültige Matrix Handles EigenvectorsRealID und EigenvectorsImagID zurückgegeben. Zugriff auf die Elemente der Matrizen ist z.B. mit dem Operator get_full_matrix möglich.

Der Berechnungstyp der Eigenvektoren kann über den Parameter ComputeEigenvectors gewählt werden. Keine Eigenvektoren werden berechnet, wenn ComputeEigenvectors = *'none'* eingestellt ist. Sollen die linken Eigenvektoren ausgegeben werden, so ist *'left'* zu wählen. Sind die rechten Eigenvektoren gewünscht, muss *'right'* gewählt werden.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & 4.0 & -8.0 \\ 5.0 & 7.0 & 3.0 \\ 4.0 & -1.0 & 4.0 \end{bmatrix} \qquad \texttt{ComputeEigenvectors} = \textit{'right'}$$

$$\rightarrow \quad \texttt{EigenvaluesReal} = \begin{bmatrix} 3.3110 \\ 3.3110 \\ 10.3781 \end{bmatrix} \qquad \texttt{EigenvaluesImag} = \begin{bmatrix} 5.4143 \\ -5.4143 \\ 0.0 \end{bmatrix}$$

$$\texttt{EigenvectorsReal} = \begin{bmatrix} 0.6353 & 0.6353 & 0.4813 \\ -0.4764 & -0.4764 & 0.8605 \\ -0.0246 & -0.0246 & 0.1669 \end{bmatrix}$$

$$\texttt{EigenvectorsImag} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ -0.2485 & 0.2485 & 0.0 \\ -0.5542 & 0.5542 & 0.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixID** (input_control) ..... matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **ComputeEigenvectors** (input_control) ..... string ⇝ *string*
  Berechnung der Eigenvektoren.
  Defaultwert: 'none'
  Werteliste: ComputeEigenvectors ∈ {'none', 'left', 'right'}
- ▷ **EigenvaluesRealID** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Realteilen der Eigenwerte.
- ▷ **EigenvaluesImagID** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Imaginärteilen der Eigenwerte.
- ▷ **EigenvectorsRealID** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Realteilen der Eigenvektoren.
- ▷ **EigenvectorsImagID** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Imaginärteilen der Eigenvektoren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert eigenvalues_general_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix, get_diagonal_matrix

Siehe auch

eigenvalues_symmetric_matrix, generalized_eigenvalues_symmetric_matrix, generalized_eigenvalues_general_matrix

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

eigenvalues_symmetric_matrix (: : MatrixID,
ComputeEigenvectors : EigenvaluesID, EigenvectorsID)

Berechnet die Eigenwerte und optional die Eigenvektoren einer symmetrischen Matrix.

Der Operator eigenvalues_symmetric_matrix berechnet alle Eigenwerte und optional die Eigenvektoren der symmetrischen Matrix Matrix, die durch das Matrix Handle MatrixID gegeben ist. Es wird eine neue Matrix Eigenvalues mit den Eigenwerten in aufsteigender Reihenfolge und optional eine neue Matrix Eigenvectors mit den Eigenvektoren generiert. Der Operator gibt die Matrix Handle EigenvaluesID und EigenvectorsID der Matrizen Eigenvalues und Eigenvectors zurück. Zugriff auf die Elemente der Matrizen ist z.B. mit dem Operator get_full_matrix möglich.

Die Berechnung der Eigenvektoren kann über ComputeEigenvectors = *'true'* oder ComputeEigenvectors = *'false'* ausgewählt werden.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & 5.0 & 3.0 \\ 5.0 & 7.0 & 3.0 \\ 3.0 & 3.0 & 4.0 \end{bmatrix} \qquad \texttt{ComputeEigenvectors} = 'true'$$

$$\rightarrow \quad \texttt{Eigenvalues} = \begin{bmatrix} 1.4195 \\ 2.1507 \\ 13.4298 \end{bmatrix} \quad \texttt{Eigenvectors} = \begin{bmatrix} 0.7842 & 0.0626 & 0.6174 \\ -0.5667 & 0.4776 & 0.6714 \\ -0.2529 & -0.8763 & 0.4100 \end{bmatrix}$$

Achtung

Der obere Dreiecksteil der Eingabematrix `Matrix` muss die relevante Information der Matrix beinhalten. Der exakt untere Teil der Matrix wird nicht verwendet. Wenn der verwendete Teil der Eingabematrix `Matrix` nicht vom spezifizierten Typ ist, wird eine Fehlerbehandlung durchgeführt.

Parameter

- ▷ **`MatrixID`** (input_control) ..... matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **`ComputeEigenvectors`** (input_control) ..... string ⇝ *string*
  Berechnung der Eigenvektoren.
  Defaultwert: 'false'
  Werteliste: `ComputeEigenvectors` ∈ {'true', 'false'}
- ▷ **`EigenvaluesID`** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Eigenwerten.
- ▷ **`EigenvectorsID`** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Eigenvektoren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `eigenvalues_symmetric_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`eigenvalues_general_matrix`

Siehe auch

`generalized_eigenvalues_symmetric_matrix`,
`generalized_eigenvalues_general_matrix`

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

`generalized_eigenvalues_general_matrix` `( : : MatrixAID, MatrixBID, ComputeEigenvectors : EigenvaluesRealID, EigenvaluesImagID, EigenvectorsRealID, EigenvectorsImagID )`

Berechnet die generalisierten Eigenwerte und optional die generalisierten Eigenvektoren allgemeiner Matrizen.

Der Operator `generalized_eigenvalues_general_matrix` berechnet alle generalisierten Eigenwerte und optional die rechten oder linken generalisierten Eigenvektoren der quadratischen, allgemeinen Eingangsmatrizen `MatrixA` und `MatrixB`. Beide Eingabematrizen müssen die gleichen Dimensionen besitzen. Die Eingangsmatrizen sind durch die Matrix Handle `MatrixAID` und `MatrixBID` definiert. Die berechneten Eigenvektoren sind auf die Länge 1 normiert.

Der Operator generiert die neuen Matrizen EigenvaluesReal und EigenvaluesImag mit den Real- und Imaginärteilen der berechneten Eigenwerte. Jede Matrix besteht aus einer Spalte und n Zeilen, wobei n die Anzahl der Zeilen oder Spalten der Eingabematrizen ist. Im Gegensatz zu dem Operator generalized_eigenvalues_symmetric_matrix ist die Reihenfolge der generalisierten Eigenwerte nicht definiert. Der Operator gibt die Matrix Handles EigenvaluesRealID und EigenvaluesImagID zurück. Die Real- und Imaginärteile der zugehörigen Eigenvektoren werden, falls gewünscht, in den neuen Matrizen EigenvectorsReal und EigenvectorsImag gespeichert. In diesem Fall enthält die j-te Spalte der Matrizen der Eigenvektoren den zugehörigen Eigenvektor zum j-ten Eigenwert. Zusätzlich werden die gültigen Matrix Handles EigenvectorsRealID und EigenvectorsImagID zurückgegeben. Zugriff auf die Elemente der Matrizen ist z.B. mit dem Operator get_full_matrix oder get_sub_matrix möglich.

Der Berechnungstyp der Eigenvektoren kann über den Parameter ComputeEigenvectors gewählt werden. Wenn ComputeEigenvectors = *'none'* eingestellt ist, werden keine Eigenvektoren berechnet, und der Operator wird schneller. In diesen Fall sind die Matrix Handles EigenvectorsRealID und EigenvectorsImagID ungültig. Sollen die rechten generalisierten Eigenvektoren ausgegeben werden, so ist *'right'* zu wählen. Die Formel für die Berechnung lautet

$$\texttt{MatrixA} \cdot x_j = \lambda_j \cdot \texttt{MatrixB} \cdot x_j,$$

wobei λ_j den j-ten (komplexen) Eigenwert und x_j den zugehörigen (komplexen) Eigenvektor darstellt.

Sind die linken generalisierten Eigenvektoren gewünscht, muss *'left'* gewählt werden. Die Formel für die Berechnung lautet

$$x^H{}_j \ \texttt{MatrixA} \cdot = x^H{}_j \cdot \lambda_j \cdot \texttt{MatrixB},$$

wobei $x^H{}_j$ die Konjugiert-Transponierte von x_j darstellt.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 6.0 & 4.0 & -8.0 \\ 5.0 & 7.0 & 3.0 \\ 4.0 & -1.0 & 4.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ -5.0 & 7.0 & 2.0 \\ 9.0 & 4.0 & 1.0 \end{bmatrix}$$

$$\texttt{ComputeEigenvectors} = \textit{'right'}$$

$$\rightarrow \quad \texttt{EigenvaluesReal} = \begin{bmatrix} 0.5363 \\ 0.5363 \\ -6.1616 \end{bmatrix} \qquad \texttt{EigenvaluesImag} = \begin{bmatrix} 0.4208 \\ -0.4208 \\ 0.0 \end{bmatrix}$$

$$\texttt{EigenvectorsReal} = \begin{bmatrix} 0.3500 & 0.3500 & 0.0410 \\ -0.9565 & -0.9565 & 0.3267 \\ -0.2757 & -0.2757 & -1.0000 \end{bmatrix}$$

$$\texttt{EigenvectorsImag} = \begin{bmatrix} -0.4644 & 0.4644 & 0.0 \\ 0.0435 & -0.0435 & 0.0 \\ -0.1869 & 0.1869 & 0.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixAID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix A.
- ▷ **MatrixBID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix B.
- ▷ **ComputeEigenvectors** (input_control) . . . . . string ⇝ *string*
  Berechnung der Eigenvektoren.
  Defaultwert: 'none'
  Werteliste: ComputeEigenvectors ∈ {'none', 'left', 'right'}
- ▷ **EigenvaluesRealID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit den Realteilen der Eigenwerte.
- ▷ **EigenvaluesImagID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit den Imaginärteilen der Eigenwerte.
- ▷ **EigenvectorsRealID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit den Realteilen der Eigenvektoren.

▷ **EigenvectorsImagID** (output_control) ..... matrix ⇝ *handle*
Matrix Handle mit den Imaginärteilen der Eigenvektoren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert generalized_eigenvalues_general_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix, get_diagonal_matrix

Siehe auch

generalized_eigenvalues_symmetric_matrix, eigenvalues_symmetric_matrix, eigenvalues_general_matrix

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

generalized_eigenvalues_symmetric_matrix (: : MatrixAID, MatrixBID, ComputeEigenvectors : EigenvaluesID, EigenvectorsID)

Berechnet die generalisierten Eigenwerte und optional die generalisierten Eigenvektoren symmetrischer Eingabematrizen.

Der Operator generalized_eigenvalues_symmetric_matrix berechnet alle generalisierten Eigenwerte und optional die generalisierten Eigenvektoren der symmetrischen Matrix MatrixA und der symmetrisch positiv definiten MatrixB. Die Eingangsmatrizen sind durch die Matrix Handle MatrixAID und MatrixBID definiert. Beide Eingabematrizen müssen die gleichen Dimensionen besitzen. Es wird eine neue Matrix Eigenvalues mit den generalisierten Eigenwerten in aufsteigender Reihenfolge und optional eine neue Matrix Eigenvectors mit den generalisierten Eigenvektoren generiert. Jede j-te Spalte der Matrix Eigenvectors enthält den zugehörigen Eigenvektor zum j-ten Eigenwert. Der Operator gibt die Matrix Handle EigenvaluesID und EigenvectorsID der Matrizen Eigenvalues und Eigenvectors zurück. Zugriff auf die Elemente der Matrizen ist z.B. mit dem Operator get_full_matrix oder get_sub_matrix möglich.

Die Berechnung der generalisierten Eigenvektoren kann über ComputeEigenvectors = *'true'* ausgewählt werden. Die Formel für die Berechnung lautet

$$\texttt{MatrixA} \cdot x_j = \lambda_j \cdot \texttt{MatrixB} \cdot x_j,$$

wobei λ_j den j-ten Eigenwert und x_j den zugehörigen Eigenvektor darstellt.

Keine generalisierten Eigenvektoren werden berechnet, wenn ComputeEigenvectors = *'false'* eingestellt ist. In diesem Fall ist das Matrix Handle EigenvectorsID ungültig.

Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 4.0 & -3.0 & 2.0 \\ -3.0 & 1.0 & 4.0 \\ 2.0 & 4.0 & 8.0 \end{bmatrix} \qquad \texttt{MatrixB} = \begin{bmatrix} 6.0 & 5.0 & 3.0 \\ 5.0 & 7.0 & 3.0 \\ 3.0 & 3.0 & 4.0 \end{bmatrix}$$

$$\texttt{ComputeEigenvectors} = 'true'$$

$$\rightarrow \quad \texttt{Eigenvalues} = \begin{bmatrix} -0.4463 \\ 1.8747 \\ 4.5472 \end{bmatrix} \quad \texttt{Eigenvectors} = \begin{bmatrix} -0.1579 & 0.1851 & -0.6358 \\ -0.3377 & -0.2568 & 0.4311 \\ 0.2382 & 0.5065 & 0.3185 \end{bmatrix}$$

Achtung

Die oberen Dreiecksteile der Eingabematrizen `MatrixA` und `MatrixB` müssen die relevante Information der Matrizen beinhalten. Die exakt unteren Teile der Matrizen werden nicht verwendet. Wenn die verwendeten Teile der Eingabematrizen `MatrixA` oder `MatrixB` nicht vom spezifizierten Typ sind, wird eine Fehlerbehandlung durchgeführt.

Parameter

- ▷ **`MatrixAID`** (input_control) ..... matrix ⇝ *handle*
  Matrix Handle der symmetrischen Eingabematrix A.
- ▷ **`MatrixBID`** (input_control) ..... matrix ⇝ *handle*
  Matrix Handle der symmetrisch positiv definiten Eingabematrix B.
- ▷ **`ComputeEigenvectors`** (input_control) ..... string ⇝ *string*
  Berechnung der Eigenvektoren.
  Defaultwert: 'false'
  Werteliste: `ComputeEigenvectors` ∈ {'true', 'false'}
- ▷ **`EigenvaluesID`** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Eigenwerten.
- ▷ **`EigenvectorsID`** (output_control) ..... matrix ⇝ *handle*
  Matrix Handle mit den Eigenvektoren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `generalized_eigenvalues_symmetric_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`generalized_eigenvalues_general_matrix`

Siehe auch

`eigenvalues_symmetric_matrix`, `eigenvalues_general_matrix`

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

18.4 Generierung

`clear_matrix` (: : MatrixID :)

Löscht die Matrix aus dem Speicher.

Der Operator clear_matrix entfernt die übergebene Matrix Matrix aus dem Speicher, die durch das Matrix Handle MatrixID gegeben ist. Nach dem Aufruf kann die Matrix Matrix nicht mehr genutzt werden. Das Matrix Handle MatrixID wird ungültig.

Parameter

▷ **MatrixID** (input_control) ..... matrix(-array) ⇝ *handle*
Matrix Handle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert clear_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_matrix

Modul

Foundation

copy_matrix (: : MatrixID : MatrixCopyID)

Kopiert eine Matrix.

Der Operator copy_matrix erstellt die neue Matrix MatrixCopy und kopiert alle Elemente der Eingabematrix Matrix in die neue Matrix. Die Eingabematrix ist durch das Matrix Handle MatrixID definiert. Der Operator gibt das Matrix Handle MatrixCopyID der Matrix MatrixCopy zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Mit diesem Operator kann eine Matrix gesichert werden, bevor die Matrix von den Operatoren set_value_matrix, set_full_matrix, set_sub_matrix oder set_diagonal_matrix verändert wird.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \texttt{MatrixCopy} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **MatrixCopyID** (output_control) ..... matrix ⇝ *handle*
Matrix Handle der kopierten Matrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert copy_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Alternativen

repeat_matrix

Modul

Foundation

create_matrix (: : Rows, Columns, Value : MatrixID)

Legt eine Matrix an.

Der Operator create_matrix legt eine neue Matrix Matrix mit Rows Zeilen und Columns Spalten an und gibt das Matrix Handle MatrixID zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Der Parameter Value ist ein String oder ein Tupel von Gleitpunkt- oder ganzen Zahlen. Ganze Zahlen werden automatisch in Gleitpunktzahlen konvertiert.

Wenn Value den Wert *'identity'* enthält, muss Rows und Columns gleich sein und eine Einheitsmatrix wird generiert.

Beispiel:

$$\texttt{Rows} = 3 \qquad \texttt{Columns} = 3 \qquad \texttt{Value} = \text{'}identity\text{'}$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Wenn der Parameter Value einen einzelnen Wert beinhaltet, werden alle Elemente der Matrix auf diesen Wert gesetzt.

Beispiel:

$$\texttt{Rows} = 3 \qquad \texttt{Columns} = 3 \qquad \texttt{Value} = 7$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 7.0 & 7.0 & 7.0 \\ 7.0 & 7.0 & 7.0 \\ 7.0 & 7.0 & 7.0 \end{bmatrix}$$

Wenn Value aus so vielen Werten wie die Hauptdiagonale besteht, werden alle Hauptdiagonalelemente mit den Werten des Parameters Value besetzt (d.h., die Anzahl der Elemente in Value ist identisch).

Beispiel:

$$\texttt{Rows} = 3 \qquad \texttt{Columns} = 4 \qquad \texttt{Value} = [3, 7, 1]$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 3.0 & 0 & 0 & 0 \\ 0 & 7.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \end{bmatrix}$$

Ebenfalls ist es möglich, alle Elemente mit unterschiedlichen Werten zu besetzen. In diesem Fall müssen in dem Parameter Value alle Werte zeilenweise vorliegen, d.h. Zeile für Zeile gespeichert sein. Außerdem muss die Anzahl der in Value übergebenen Werte mit der Anzahl aller Elemente der Matrix, d.h. Rows $\times$ Columns, identisch sein.

Beispiel:

$$\texttt{Rows} = 3 \qquad \texttt{Columns} = 3 \qquad \texttt{Value} = [3, 1, -2, -5, 7, 2, -9, -4, 1]$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix}$$

Es ist zu beachten, dass in den Beispielen Unterschiede in der Bedeutung der Werte in den Ausgabematrizen zu finden sind: Wenn ein Wert als ganze Zahl dargestellt ist, z.B. 0 oder 1, ist der Wert dieses Elements per Definition dieser bestimmte Wert. Wenn der Wert als Gleitpunktzahl dargestellt ist, z.B. 0.0 oder 1.0, ist der Wert von dem Operator berechnet worden.

Parameter

- ▷ **Rows** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der Zeilen der Matrix.
  Defaultwert: 3
  Wertevorschläge: Rows ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
  Restriktion: Rows >= 1
- ▷ **Columns** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der Spalten der Matrix.
  Defaultwert: 3
  Wertevorschläge: Columns ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
  Restriktion: Columns >= 1
- ▷ **Value** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Werte mit denen die Elemente der Matrix initialisiert werden.
  Defaultwert: 0
  Wertevorschläge: Value ∈ {0, 1, 'identity'}
- ▷ **MatrixID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

set_full_matrix, clear_matrix

Modul

Foundation

repeat_matrix (: : MatrixID, Rows, Columns : MatrixRepeatedID)

Wiederhole eine Matrix.

Der Operator repeat_matrix erstellt die neue Matrix MatrixRepeated und kopiert alle Elemente der Eingabematrix Matrix n mal in die neue Matrix, wobei n = Rows · Columns. Die neue Matrix hat als Dimensionen Rows × Zeilen der Eingabematrix Matrix und Columns × Spalten der Eingabematrix Matrix. Die Eingabematrix ist durch das Matrix Handle MatrixID definiert. Der Operator gibt das Matrix Handle MatrixRepeatedID der Matrix MatrixRepeated zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & -1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \end{bmatrix} \quad \texttt{Rows} = 2 \quad \texttt{Columns} = 3$$

$$\rightarrow \quad \texttt{MatrixRepeated} = \begin{bmatrix} 3.0 & -1.0 & -2.0 & 3.0 & -1.0 & -2.0 & 3.0 & -1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 & -5.0 & 7.0 & 2.0 & -5.0 & 7.0 & 2.0 \\ 3.0 & -1.0 & -2.0 & 3.0 & -1.0 & -2.0 & 3.0 & -1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 & -5.0 & 7.0 & 2.0 & -5.0 & 7.0 & 2.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **Rows** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Kopien der Matrix in Zeilenrichtung.
Defaultwert: 2
Wertevorschläge: Rows ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20}
Restriktion: Rows >= 1

▷ **Columns** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Kopien der Matrix in Spaltenrichtung.
Defaultwert: 2
Wertevorschläge: Columns ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20}
Restriktion: Columns >= 1

▷ **MatrixRepeatedID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle der mehrfach kopierten Matrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert repeat_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Siehe auch

copy_matrix

Modul

Foundation

18.5 Merkmale

determinant_matrix (: : MatrixID, MatrixType : Value)

Berechnet die Determinante einer Matrix.

Der Operator determinant_matrix berechnet die Determinante der Eingabematrix Matrix, die durch das Matrix Handle MatrixID gegeben ist. Der Typ der Eingabematrix Matrix kann über den Parameter MatrixType gewählt werden. Die folgenden Werte stehen zur Verfügung: *'general'* für allgemeine, *'symmetric'* für symmetrische, *'positive_definite'* für symmetrisch positiv definite und *'tridiagonal'* für tridiagonale Matrizen, *'upper_triangular'* für obere, *'permuted_upper_triangular'* für permutierte obere, *'lower_triangular'* für untere und *'permuted_lower_triangular'* für permutierte untere Dreiecksmatrizen. Die Formel für die Berechnung lautet:

$$\texttt{Value} \quad = \quad \det \texttt{Matrix}.$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & 2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \texttt{Value} = -134.0$$

Achtung

Wird MatrixType = *'symmetric'*, *'positive_definite'* oder *'upper_triangular'* gewählt, so muss der obere Dreieckteil der Eingabematrix Matrix die relevante Information der Matrix beinhalten. Der exakt untere Teil der Matrix wird nicht verwendet. Wird MatrixType = *'lower_triangular'* gewählt, so muss der untere Dreieckteil der Eingabematrix Matrix die relevante Information der Matrix beinhalten. Der exakt obere Teil der Matrix wird nicht verwendet. Wird MatrixType = *'tridiagonal'* gesetzt, wird nur die Hauptdiagonale und die beiden Nebendiagonalen der Eingabematrix Matrix genutzt. Die anderen Teile der Matrix werden nicht verwendet. Wenn der verwendete Teil der Eingabematrix Matrix nicht vom spezifizierten Typ ist, wird eine Fehlerbehandlung durchgeführt.

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **MatrixType** (input_control) . . . . . string ⇝ *string*
  Der Matrixtyp der Eingabematrix.
  Defaultwert: 'general'
  Werteliste: MatrixType ∈ {'general', 'symmetric', 'positive_definite', 'tridiagonal', 'upper_triangular', 'permuted_upper_triangular', 'lower_triangular', 'permuted_lower_triangular'}
- ▷ **Value** (output_control) . . . . . real ⇝ *real*
  Determinante der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert determinant_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

get_size_matrix (: : MatrixID : Rows, Columns)

Liefert die Größe einer Matrix.

Der Operator get_size_matrix gibt the Anzahl der Zeilen Rows und der Spalten Columns der Eingabematrix Matrix zurück. Die Eingabematrix Matrix wird durch das Matrix Handle MatrixID definiert.

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **Rows** (output_control) . . . . . integer ⇝ *integer*
Anzahl der Zeilen der Matrix.

▷ **Columns** (output_control) . . . . . integer ⇝ *integer*
Anzahl der Spalten der Matrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_size_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`clear_matrix`

Modul

Foundation

max_matrix (: : MatrixID, MaxType : MatrixMaxID)

Gibt den elementweisen Maximalwert der Matrix zurück.

Der Operator `max_matrix` gibt den Maximalwert der Elemente der Eingabematrix `Matrix` zurück. Die Eingabematrix ist durch das Matrix Handle `MatrixID` definiert. Es wird eine neue Matrix `MatrixMax` mit dem Ergebnis generiert und das Matrix Handle `MatrixMaxID` dieser Matrix zurückgegeben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich.

Die Art der Maximalwertbildung der Matrix kann mittels des Parameters `MaxType` gewählt werden:

'columns': Die Maximalwertbildung erfolgt für jede Spalte der Matrix `Matrix` einzeln. Die Ergebnismatrix `MatrixMax` hat eine Zeile und die gleiche Anzahl an Spalten wie die Eingabematrix.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MaxType} = 'columns'$$
$$\rightarrow \qquad \texttt{MatrixMax} = \begin{bmatrix} 8.0 & 4.0 & 7.0 \end{bmatrix}$$

'rows': Die Maximalwertbildung erfolgt für jede Zeile der Matrix `Matrix` einzeln. Die Ergebnismatrix `MatrixMax` hat die gleiche Anzahl an Zeilen wie die Eingabematrix und eine Spalte.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MaxType} = 'rows'$$
$$\rightarrow \qquad \texttt{MatrixMax} = \begin{bmatrix} 8.0 \\ 7.0 \end{bmatrix}$$

'full': Die Maximalwertbildung erfolgt über alle Elemente der Matrix `Matrix`. Die Ergebnismatrix `MatrixMax` hat eine Zeile und eine Spalte.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MaxType} = 'full'$$
$$\rightarrow \qquad \texttt{MatrixMax} = \begin{bmatrix} 8.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **MaxType** (input_control) . . . . . string ⇝ *string*
Art der Maximalwertbildung.
Defaultwert: 'columns'
Werteliste: MaxType ∈ {'columns', 'rows', 'full'}

▷ **MatrixMaxID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den Maximalwerten der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert max_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Siehe auch

min_matrix

Modul

Foundation

mean_matrix (: : MatrixID, MeanType : MatrixMeanID)

Gibt den elementweisen Mittelwert einer Matrix zurück.

Der Operator mean_matrix gibt den Mittelwert der Elemente der Eingabematrix Matrix zurück. Die Eingabematrix ist durch das Matrix Handle MatrixID definiert. Es wird eine neue Matrix MatrixMean mit dem Ergebnis generiert und das Matrix Handle MatrixMeanID dieser Matrix zurückgegeben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Die Art der Mittelwertbildung der Matrix kann mittels des Parameters MeanType gewählt werden:

'columns': Die Mittelwertbildung erfolgt für jede Spalte der Matrix Matrix einzeln. Die Ergebnismatrix MatrixMean hat eine Zeile und die gleiche Anzahl an Spalten wie die Eingabematrix.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MeanType} = 'columns'$$

$$\rightarrow \qquad \texttt{MatrixMean} = \begin{bmatrix} 1.0 & 3.0 & 2.0 \end{bmatrix}$$

'rows': Die Mittelwertbildung erfolgt für jede Zeile der Matrix Matrix einzeln. Die Ergebnismatrix MatrixMean hat die gleiche Anzahl an Zeilen wie die Eingabematrix und eine Spalte.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MeanType} = 'rows'$$

$$\rightarrow \qquad \texttt{MatrixMean} = \begin{bmatrix} 3.0 \\ 1.0 \end{bmatrix}$$

'full': Die Mittelwertbildung erfolgt über alle Elemente der Matrix `Matrix`. Die Ergebnismatrix `MatrixMean` hat eine Zeile und eine Spalte.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MeanType} = \textit{'full'}$$
$$\rightarrow \qquad \texttt{MatrixMean} = \begin{bmatrix} 2.0 \end{bmatrix}$$

Parameter

▷ **`MatrixID`** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **`MeanType`** (input_control) . . . . . string ⇝ *string*
Art der Mittelwertwertbildung.
Defaultwert: 'columns'
Werteliste: `MeanType` ∈ {'columns', 'rows', 'full'}

▷ **`MatrixMeanID`** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den Mittelwerten der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `mean_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Siehe auch

`norm_matrix`, `sum_matrix`

Modul

Foundation

`min_matrix` (: : `MatrixID`, `MinType` : `MatrixMinID`)

Gibt das elementweise Minimum einer Matrix zurück.

Der Operator `min_matrix` gibt das Minimum der Elemente der Eingabematrix `Matrix` zurück. Die Eingabematrix ist durch das Matrix Handle `MatrixID` definiert. Es wird eine neue Matrix `MatrixMin` mit dem Ergebnis generiert und das Matrix Handle `MatrixMinID` dieser Matrix zurückgegeben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator `get_full_matrix` möglich.

Die Art der Minimalwertbildung der Matrix kann mittels des Parameters `MinType` gewählt werden:

'columns': Die Minimalwertbildung erfolgt für jede Spalte der Matrix `Matrix` einzeln. Die Ergebnismatrix `MatrixMin` hat eine Zeile und die gleiche Anzahl an Spalten wie die Eingabematrix.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MinType} = \textit{'columns'}$$
$$\rightarrow \qquad \texttt{MatrixMin} = \begin{bmatrix} -6.0 & 2.0 & -3.0 \end{bmatrix}$$

'rows': Die Minimalwertbildung erfolgt für jede Zeile der Matrix `Matrix` einzeln. Die Ergebnismatrix `MatrixMin` hat die gleiche Anzahl an Zeilen wie die Eingabematrix und eine Spalte.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MinType} = \textit{'rows'}$$
$$\rightarrow \qquad \texttt{MatrixMin} = \begin{bmatrix} -3.0 \\ -6.0 \end{bmatrix}$$

'full': Die Minimalwertbildung erfolgt über alle Elemente der Matrix `Matrix`. Die Ergebnismatrix `MatrixMin` hat eine Zeile und eine Spalte.
Beispiel:

$$\texttt{MatrixA} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{MinType} = \textit{'full'}$$
$$\rightarrow \qquad \texttt{MatrixMin} = \begin{bmatrix} -6.0 \end{bmatrix}$$

Parameter

▷ **`MatrixID`** (input_control) ..... matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **`MinType`** (input_control) ..... string ⇝ *string*
Art der Minimalwertwertbildung.
Defaultwert: 'columns'
Werteliste: `MinType` ∈ {'columns', 'rows', 'full'}

▷ **`MatrixMinID`** (output_control) ..... matrix ⇝ *handle*
Matrix Handle mit den Minimalwerten der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `min_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Siehe auch

`max_matrix`

Modul

Foundation

`norm_matrix` (: : `MatrixID`, `NormType` : `Value`)

Normiert eine Matrix.

Der Operator `norm_matrix` berechnet die Norm der Elemente der Matrix `Matrix`, die durch das Matrix Handle `MatrixID` gegeben ist. Der Rückgabewert ist eine Gleitpunktzahl.

Der Typ der Normierung der Matrix kann über den Parameter `NormType` gewählt werden:

'frobenius-norm': Die Frobenius-Norm wird berechnet. Die Formel für die Berechnung des Rückgabewerts lautet:

$$\texttt{Value} = \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \texttt{Matrix}_{ij}{}^2}$$

mit m = Anzahl der Zeilen und n = Anzahl der Spalten der Matrix `Matrix`.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 \\ 4.0 & -3.0 \\ -7.0 & 4.0 \end{bmatrix} \quad \rightarrow \quad \texttt{Value} = 10.0$$

'infinity-norm': Die infinity Norm wird berechnet. Das Ergebnis ist der größte Wert der Summe der absoluten Werte der Elemente der Zeilen. Die Formel für die Berechnung des Rückgabewerts lautet:

$$\texttt{Value} \quad = \quad \max_{i=0,m-1} \sum_{j=0}^{n-1} |\texttt{Matrix}_{ij}|$$

mit m = Anzahl der Zeilen und n = Anzahl der Spalten der Matrix `Matrix`.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 \\ 4.0 & -3.0 \\ -7.0 & 4.0 \end{bmatrix} \quad \rightarrow \quad \texttt{Value} = 11.0$$

'1-norm': Die 1-Norm wird berechnet. Das Ergebnis ist der größte Wert der Summe der absoluten Werte der Elemente der Spalten. Die Formel für die Berechnung des Rückgabewerts lautet:

$$\texttt{Value} \quad = \quad \max_{j=0,n-1} \sum_{i=0}^{m-1} |\texttt{Matrix}_{ij}|$$

mit m = Anzahl der Zeilen und n = Anzahl der Spalten der Matrix `Matrix`.
Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 \\ 4.0 & -3.0 \\ -7.0 & 4.0 \end{bmatrix} \quad \rightarrow \quad \texttt{Value} = 14.0$$

'2-norm': Die 2-norm wird berechnet. Das Ergebnis ist der größte Singulärwert der `Matrix`. Die Formel für die Berechnung des Rückgabewerts lautet:

$$\texttt{Value} \quad = \quad \max\left(\text{singular values}\left(\texttt{Matrix}\right)\right)$$

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 \\ 4.0 & -3.0 \\ -7.0 & 4.0 \end{bmatrix} \quad \rightarrow \quad \texttt{Value} = 9.7006$$

Parameter

- ▷ **`MatrixID`** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **`NormType`** (input_control) . . . . . string ⇝ *string*
  Typ der Norm.
  Defaultwert: '2-norm'
  Werteliste: `NormType` ∈ {'2-norm', '1-norm', 'infinity-norm', 'frobenius-norm'}
- ▷ **`Value`** (output_control) . . . . . real ⇝ *real*
  Norm der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `norm_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Siehe auch

sum_matrix, mean_matrix

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

sum_matrix (: : MatrixID, SumType : MatrixSumID)

Gibt elementweise die Summen der Matrix zurück.

Der Operator sum_matrix berechnet die Summe der Elemente der Eingabematrix Matrix. Die Eingabematrix ist durch das Matrix Handle MatrixID definiert. Es wird eine neue Matrix MatrixSum mit dem Ergebnis generiert und das Matrix Handle MatrixSumID dieser Matrix zurückgegeben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Die Art der Summierung der Matrix kann mittels des Parameters SumType gewählt werden:

'columns': Die Summierung erfolgt für jede Spalte der Matrix Matrix einzeln. Die Ergebnismatrix MatrixSum hat eine Zeile und die gleiche Anzahl an Spalten wie die Eingabematrix.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{SumType} = 'columns'$$

$$\rightarrow \qquad \texttt{MatrixSum} = \begin{bmatrix} 2.0 & 6.0 & 4.0 \end{bmatrix}$$

'rows': Die Summierung erfolgt für jede Zeile der Matrix Matrix einzeln. Die Ergebnismatrix MatrixSum hat die gleiche Anzahl an Zeilen wie die Eingabematrix und eine Spalte.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{SumType} = 'rows'$$

$$\rightarrow \qquad \texttt{MatrixSum} = \begin{bmatrix} 9.0 \\ 3.0 \end{bmatrix}$$

'full': Die Summierung erfolgt über alle Elemente der Matrix Matrix. Die Ergebnismatrix MatrixSum hat eine Zeile und eine Spalte.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 8.0 & 4.0 & -3.0 \\ -6.0 & 2.0 & 7.0 \end{bmatrix} \qquad \texttt{SumType} = 'full'$$

$$\rightarrow \qquad \texttt{MatrixSum} = \begin{bmatrix} 12.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **SumType** (input_control) . . . . . string ⇝ *string*
Art der Summierung.
Defaultwert: 'columns'
Werteliste: SumType ∈ {'columns', 'rows', 'full'}

▷ **MatrixSumID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den Summen der Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `sum_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Siehe auch

`norm_matrix`

Modul

Foundation

18.6 Zerlegung

`decompose_matrix` (: : `MatrixID`, `MatrixType` : `Matrix1ID`, `Matrix2ID`)

Zerlegt eine Matrix.

Der Operator `decompose_matrix` zerlegt die quadratische Eingabematrix `Matrix`, die durch das Matrix Handle `MatrixID` gegeben ist. Die Ergebnisse werden in zwei generierten Matrizen `Matrix1` und `Matrix2` gespeichert und deren Matrix Handles `Matrix1ID` und `Matrix2ID` werden zurückgegeben. Zugriff auf die Elemente der Matrizen ist z.B. mit dem Operator `get_full_matrix` möglich.

Der Typ der Eingabematrix `Matrix` kann über den Parameter `MatrixType` gewählt werden. Die folgenden Werte stehen zur Verfügung: *'general'* für allgemeine, *'symmetric'* für symmetrische, *'positive_definite'* für symmetrisch positiv definite und *'tridiagonal'* für tridiagonale Matrizen.

Wenn `MatrixType` = *'general'* oder *'tridiagonal'* gewählt ist, wird eine LU Zerlegung (Lower/Upper) der Form

$$\texttt{Matrix} \quad = \quad \texttt{Matrix1} \quad \cdot \quad \texttt{Matrix2}.$$

vorgenommen. Die Ausgabematrix `Matrix1` ist eine untere Dreiecksmatrix mit Einheitsdiagonalelementen, deren Zeilen vertauscht sind. Die Ausgabematrix `Matrix2` ist eine obere Dreiecksmatrix.

Beispiel für die Zerlegung einer allgemeinen Matrix:

$$\texttt{Matrix} = \begin{bmatrix} 5.0 & -3.0 & 1.0 \\ 0.0 & 2.0 & -1.0 \\ -5.0 & -1.0 & 5.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{Matrix1} = \begin{bmatrix} 1 & 0 & 0 \\ 0.0 & -0.5 & 1 \\ -1.0 & 1 & 0 \end{bmatrix} \quad \texttt{Matrix2} = \begin{bmatrix} 5.0 & -3.0 & 1.0 \\ 0 & -4.0 & 6.0 \\ 0 & 0 & 2.0 \end{bmatrix}$$

Beispiel für die Zerlegung einer tridiagonalen Matrix:

$$\texttt{Matrix} = \begin{bmatrix} -8.0 & -8.0 & 0.0 & 0.0 \\ -8.0 & 6.0 & -4.0 & 0.0 \\ 0.0 & 7.0 & -2.0 & 7.0 \\ 0.0 & 0.0 & 5.0 & -3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{Matrix1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1.0 & 1 & 0 & 0 \\ 0.0 & 0.5 & 0.0 & 1 \\ 0.0 & 0.0 & 1 & 0 \end{bmatrix} \qquad \texttt{Matrix2} = \begin{bmatrix} -8.0 & -8.0 & 0.0 & 0.0 \\ 0 & 14.0 & -4.0 & 0.0 \\ 0 & 0 & 5.0 & -3.0 \\ 0 & 0 & 0 & 7.0 \end{bmatrix}$$

Ist MatrixType = *'symmetric'* gesetzt, wird eine UDU^T Zerlegung (Upper/Diagonal/Upper) verwendet. Sie hat die Form

$$\texttt{Matrix} \quad = \quad \texttt{Matrix1} \quad \cdot \quad \texttt{Matrix2} \quad \cdot \quad \texttt{Matrix1}^T.$$

Hierbei ist die Ausgabematrix Matrix1 eine obere Dreiecksmatrix, deren Zeilen vertauscht sind. Die Ausgabematrix Matrix2 ist eine symmetrische Blockdiagonalmatrix mit 1×1 und 2×2 Blöcken auf der Hauptdiagonalen.

Beispiel für die Zerlegung einer symmetrischen Matrix:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & -2.0 & 7.0 & -1.0 \\ -2.0 & -2.0 & 4.0 & 0.0 \\ 7.0 & 4.0 & 8.0 & 1.0 \\ -1.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{Matrix1} = \begin{bmatrix} 0 & 0 & 1 & 0.0 \\ 0 & 1 & 0.0 & 2.0 \\ 1 & -1.0 & -1.0 & -10.0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \texttt{Matrix2} = \begin{bmatrix} 27.0 & 0 & 0 & 0 \\ 0 & -2.0 & 0 & 0 \\ 0 & 0 & 3.0 & -1.0 \\ 0 & 0 & -1.0 & 0.0 \end{bmatrix}$$

Eine Cholesky Zerlegung wird berechnet, wenn MatrixType = *'positive_definite'* gesetzt ist. Die Zerlegung hat die Form

$$\texttt{Matrix} \quad = \quad \texttt{Matrix1} \quad \cdot \quad \texttt{Matrix2}.$$

wobei die Ausgabematrix Matrix1 eine untere Dreiecksmatrix und Ausgabematrix Matrix2 eine obere Dreiecksmatrix ist. Außerdem ist die Matrix Matrix2 die Transponierte der Matrix Matrix1.

Beispiel für die Zerlegung einer positiv definiten Matrix:

$$\texttt{Matrix} = \begin{bmatrix} 9.0 & 12.0 & -6.0 \\ 12.0 & 17.0 & -7.0 \\ -6.0 & -7.0 & 14.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{Matrix1} = \begin{bmatrix} 3.0 & 0 & 0 \\ 4.0 & 1.0 & 0 \\ -2.0 & 1.0 & 3.0 \end{bmatrix} \qquad \texttt{Matrix2} = \begin{bmatrix} 3.0 & 4.0 & -2.0 \\ 0 & 1.0 & 1.0 \\ 0 & 0 & 3.0 \end{bmatrix}$$

Es ist zu beachten, dass in den Beispielen Unterschiede in der Bedeutung der Werte in den Ausgabematrizen zu finden sind: Wenn ein Wert als ganze Zahl dargestellt ist, z.B. 0 oder 1, ist der Wert dieses Elements per Definition dieser bestimmte Wert. Wenn der Wert als Gleitpunktzahl dargestellt ist, z.B. 0.0 oder 1.0, ist der Wert von dem Operator berechnet worden.

Achtung

Wird MatrixType = *'symmetric'* oder *'positive_definite'* gewählt, so muss der obere Dreieckteil der Eingabematrix Matrix die relevante Information der Matrix beinhalten. Der exakt untere Teil der Matrix wird nicht verwendet. Wird MatrixType = *'tridiagonal'* gesetzt, wird nur die Hauptdiagonale und die beiden Nebendiagonalen der Eingabematrix Matrix genutzt. Die anderen Teile der Matrix werden nicht verwendet. Wenn der verwendete Teil der Eingabematrix Matrix nicht vom spezifizierten Typ ist, wird eine Fehlerbehandlung durchgeführt.

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **MatrixType** (input_control) . . . . . string ⇝ *string*
  Matrixtyp der Eingabematrix.
  Defaultwert: 'general'
  Werteliste: MatrixType ∈ {'general', 'symmetric', 'positive_definite', 'tridiagonal'}

▷ **Matrix1ID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit der Ausgabematrix 1.
▷ **Matrix2ID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit der Ausgabematrix 2.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `decompose_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Alternativen

`orthogonal_decompose_matrix`, `solve_matrix`

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

orthogonal_decompose_matrix (: : MatrixID, DecompositionType, OutputMatricesType, ComputeOrthogonal : MatrixOrthogonalID, MatrixTriangularID)

Führt eine orthogonale Zerlegung einer Matrix durch.

Der Operator `orthogonal_decompose_matrix` zerlegt die Matrix `Matrix`, die durch das Matrix Handle `MatrixID` gegeben ist. Es werden die Matrizen `MatrixOrthogonal` und `MatrixTriangular` generiert und die Ergebnisse in diesen gespeichert. Der Operator gibt die Matrix Handles `MatrixOrthogonalID` und `MatrixTriangularID` dieser zwei Matrizen zurück. Zugriff auf die Elemente der Matrizen ist z.B. mit dem Operator `get_full_matrix` möglich.

Alle Ergebnisse der Zerlegung werden in den Matrizen `MatrixOrthogonal` und `MatrixTriangular` gespeichert, wenn der Parameter `OutputMatricesType` = *'full'* ist. Ist `OutputMatricesType` = *'reduced'*, wird nur ein Teil der Ergebniselemente in den Matrizen `MatrixOrthogonal` und `MatrixTriangular` gespeichert. Die Dimension dieser Matrizen ist kleiner als für `OutputMatricesType` = *'full'*.

Beide Ausgabematrizen werden mit dem Parameter `ComputeOrthogonal` = *'true'* berechnet. Für `ComputeOrthogonal` = *'false'* wird nur die Matrix `MatrixTriangular` berechnet. Die Laufzeit ist daher kürzer.

Der Zerlegungstyp der `Matrix` wird über den Parameter `DecompositionType` gewählt. Für `DecompositionType` = *'qr'* wird eine QR Zerlegung (Quadratic/Right) oder für `DecompositionType` = *'ql'* wird eine QL Zerlegung (Quadratic/Left) berechnet. Die Zerlegung wird beschrieben als

$$\texttt{Matrix} = \texttt{MatrixOrthogonal} \cdot \texttt{MatrixTriangular}.$$

Beispiel für `DecompositionType` = *'qr'* und `OutputMatricesType` = *'full'*:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & -3.0 & 4.0 \\ 5.0 & 5.0 & -2.0 \\ -3.0 & 12.0 & 5.0 \\ 3.0 & 4.0 & 7.0 \\ 7.0 & 5.0 & 3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixOrthogonal} = \begin{bmatrix} -0.5303 & 0.2613 & -0.4277 & -0.5566 & -0.3972 \\ -0.4419 & -0.2920 & -0.6383 & 0.1727 & -0.5312 \\ 0.2652 & -0.8443 & -0.2059 & -0.3961 & -0.1326 \\ -0.2652 & -0.2432 & -0.6008 & 0.7000 & -0.1400 \\ -0.6187 & -0.2729 & 0.0799 & -0.1160 & 0.7231 \end{bmatrix}$$

$$\texttt{MatrixTriangular} = \begin{bmatrix} -11.3137 & -1.5910 & -3.6239 \\ 0 & -14.7129 & -5.1135 \\ 0 & 0 & -7.9824 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Beispiel für DecompositionType = *'qr'* und OutputMatricesType = *'reduced'*:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & -3.0 & 4.0 \\ 5.0 & 5.0 & -2.0 \\ -3.0 & 12.0 & 5.0 \\ 3.0 & 4.0 & 7.0 \\ 7.0 & 5.0 & 3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixOrthogonal} = \begin{bmatrix} -0.5303 & 0.2613 & -0.4277 \\ -0.4419 & -0.2920 & -0.6383 \\ 0.2652 & -0.8443 & -0.2059 \\ -0.2652 & -0.2432 & -0.6008 \\ -0.6187 & -0.2729 & 0.0799 \end{bmatrix}$$

$$\texttt{MatrixTriangular} = \begin{bmatrix} -11.3137 & -1.5910 & -3.6239 \\ 0 & -14.7129 & -5.1135 \\ 0 & 0 & -7.9824 \end{bmatrix}$$

Beispiel für DecompositionType = *'ql'* und OutputMatricesType = *'full'*:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & -3.0 & 4.0 \\ 5.0 & 5.0 & -2.0 \\ -3.0 & 12.0 & 5.0 \\ 3.0 & 4.0 & 7.0 \\ 7.0 & 5.0 & 3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixOrthogonal} = \begin{bmatrix} 0.6806 & 0.0657 & 0.3659 & -0.4932 & -0.3941 \\ 0.1161 & 0.5464 & 0.6091 & 0.5274 & 0.1971 \\ 0.4093 & -0.0832 & -0.4046 & 0.6474 & -0.4927 \\ -0.5362 & 0.4713 & 0.0072 & -0.1208 & -0.6897 \\ -0.2611 & -0.6842 & 0.5757 & 0.2119 & -0.2956 \end{bmatrix}$$

$$\texttt{MatrixTriangular} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 10.5059 & 0 & 0 \\ -1.1429 & 12.4620 & 0 \\ -4.0399 & -7.9812 & -10.1489 \end{bmatrix}$$

Beispiel für DecompositionType = *'ql'* und OutputMatricesType = *'reduced'*:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & -3.0 & 4.0 \\ 5.0 & 5.0 & -2.0 \\ -3.0 & 12.0 & 5.0 \\ 3.0 & 4.0 & 7.0 \\ 7.0 & 5.0 & 3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixOrthogonal} = \begin{bmatrix} 0.3659 & -0.4932 & -0.3941 \\ 0.6091 & 0.5274 & 0.1971 \\ -0.4046 & 0.6474 & -0.4927 \\ 0.0072 & -0.1208 & -0.6897 \\ 0.5757 & 0.2119 & -0.2956 \end{bmatrix}$$

$$\texttt{MatrixTriangular} = \begin{bmatrix} 10.5059 & 0 & 0 \\ -1.1429 & 12.4620 & 0 \\ -4.0399 & -7.9812 & -10.1489 \end{bmatrix}$$

Für DecompositionType = *'rq'* wird eine RQ Zerlegung (Right/Quadratic) oder für DecompositionType = *'lq'* wird eine LQ Zerlegung (Left/Quadratic) berechnet. Die Zerlegung wird beschrieben als

$$\texttt{Matrix} = \texttt{MatrixTriangular} \cdot \texttt{MatrixOrthogonal}.$$

Beispiel für DecompositionType = *'rq'* und OutputMatricesType = *'full'*:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & 5.0 & -3.0 & 3.0 & 7.0 \\ -3.0 & 5.0 & 12.0 & 4.0 & 5.0 \\ 4.0 & -2.0 & 5.0 & 7.0 & 3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixOrthogonal} = \begin{bmatrix} 0.6806 & 0.1161 & 0.4093 & -0.5362 & -0.2611 \\ 0.0657 & 0.5464 & -0.0832 & 0.4713 & -0.6842 \\ 0.3659 & 0.6091 & -0.4046 & 0.0072 & 0.5757 \\ -0.4932 & 0.5274 & 0.6474 & -0.1208 & 0.2119 \\ -0.3941 & 0.1971 & -0.4927 & -0.6897 & -0.2956 \end{bmatrix}$$

$$\texttt{MatrixTriangular} = \begin{bmatrix} 0 & 0 & 10.5059 & -1.1429 & -4.0399 \\ 0 & 0 & 0 & 12.4620 & -7.9812 \\ 0 & 0 & 0 & 0 & -10.1489 \end{bmatrix}$$

Beispiel für DecompositionType = *'rq'* und OutputMatricesType = *'reduced'*:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & 5.0 & -3.0 & 3.0 & 7.0 \\ -3.0 & 5.0 & 12.0 & 4.0 & 5.0 \\ 4.0 & -2.0 & 5.0 & 7.0 & 3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixOrthogonal} = \begin{bmatrix} 0.3659 & 0.6091 & -0.4046 & 0.0072 & 0.5757 \\ -0.4932 & 0.5274 & 0.6474 & -0.1208 & 0.2119 \\ -0.3941 & 0.1971 & -0.4927 & -0.6897 & -0.2956 \end{bmatrix}$$

$$\texttt{MatrixTriangular} = \begin{bmatrix} 10.5059 & -1.1429 & -4.0399 \\ 0 & 12.4620 & -7.9812 \\ 0 & 0 & -10.1489 \end{bmatrix}$$

Beispiel für DecompositionType = *'lq'* und OutputMatricesType = *'full'*:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & 5.0 & -3.0 & 3.0 & 7.0 \\ -3.0 & 5.0 & 12.0 & 4.0 & 5.0 \\ 4.0 & -2.0 & 5.0 & 7.0 & 3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixOrthogonal} = \begin{bmatrix} -0.5303 & -0.4419 & 0.2652 & -0.2652 & -0.6187 \\ 0.2613 & -0.2920 & -0.8443 & -0.2432 & -0.2729 \\ -0.4277 & 0.6383 & -0.2059 & -0.6008 & 0.0799 \\ -0.5566 & 0.1727 & -0.3961 & 0.7000 & -0.1160 \\ -0.3972 & -0.5312 & -0.1326 & -0.1400 & 0.7231 \end{bmatrix}$$

$$\texttt{MatrixTriangular} = \begin{bmatrix} -11.3137 & 0 & 0 & 0 & 0 \\ -1.5910 & -14.7129 & 0 & 0 & 0 \\ -3.6239 & -5.1135 & -7.9824 & 0 & 0 \end{bmatrix}$$

Beispiel für DecompositionType = *'lq'* und OutputMatricesType = *'reduced'*:

$$\texttt{Matrix} = \begin{bmatrix} 6.0 & 5.0 & -3.0 & 3.0 & 7.0 \\ -3.0 & 5.0 & 12.0 & 4.0 & 5.0 \\ 4.0 & -2.0 & 5.0 & 7.0 & 3.0 \end{bmatrix}$$

$$\rightarrow \quad \texttt{MatrixOrthogonal} = \begin{bmatrix} -0.5303 & -0.4419 & 0.2652 & -0.2652 & -0.6187 \\ 0.2613 & -0.2920 & -0.8443 & -0.2432 & -0.2729 \\ -0.4277 & 0.6383 & -0.2059 & -0.6008 & 0.0799 \end{bmatrix}$$

$$\texttt{MatrixTriangular} = \begin{bmatrix} -11.3137 & 0 & 0 \\ -1.5910 & -14.7129 & 0 \\ -3.6239 & -5.1135 & -7.9824 \end{bmatrix}$$

Es ist zu beachten, dass in den Beispielen Unterschiede in der Bedeutung der Werte in den Ausgabematrizen zu finden sind: Die Ergebnisse der Elemente sind per Definition ein bestimmter Wert, wenn dieser als ganze Zahl dargestellt ist, z.B. 0 oder 1. Der Wert ist berechnet, wenn er als Gleitpunktzahl dargestellt ist, z.B. 0.0 oder 1.0.

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **DecompositionType** (input_control) . . . . . string ⇝ *string*
  Zerlegungsmethode.
  Defaultwert: 'qr'
  Werteliste: DecompositionType ∈ {'qr', 'rq', 'ql', 'lq'}
- ▷ **OutputMatricesType** (input_control) . . . . . string ⇝ *string*
  Typ der Ausgabematrizen.
  Defaultwert: 'full'
  Werteliste: OutputMatricesType ∈ {'full', 'reduced'}
- ▷ **ComputeOrthogonal** (input_control) . . . . . string ⇝ *string*
  Berechnung der orthogonalen Matrix.
  Defaultwert: 'true'
  Werteliste: ComputeOrthogonal ∈ {'true', 'false'}
- ▷ **MatrixOrthogonalID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit dem orthogonalen Teil der zerlegten Eingabematrix.
- ▷ **MatrixTriangularID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit dem triangulären Teil der zerlegten Eingabematrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert orthogonal_decompose_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Siehe auch

decompose_matrix, solve_matrix

Literatur

David Poole: „Linear Algebra: A Modern Introduction“; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations“; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

svd_matrix (: : MatrixID, SVDType,
ComputeSingularVectors : MatrixUID, MatrixSID, MatrixVID)

Berechnet die Singulärwertzerlegung einer Matrix.

Der Operator svd_matrix berechnet die volle oder reduzierte Singulärwertzerlegung (SVD) der Matrix Matrix, die durch das Matrix Handle MatrixID gegeben ist. Der Operator gibt das Matrix Handle MatrixSID der Matrix MatrixS mit den Singulärwerten in absteigender Reihenfolge zurück. Optional werden die Matrizen MatrixU mit den linken und MatrixV mit den rechten Singulärvektoren ebenfalls berechnet und deren Matrix Handles MatrixUID und MatrixVID zurückgegeben. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich. Die SVD hat die Form

$$\mathtt{Matrix} = \mathtt{MatrixU} \cdot \mathtt{MatrixS} \cdot \mathtt{MatrixV}^T.$$

Ist SVDType = *'full'* gesetzt, wird eine volle SVD berechnet.

Beispiel:

$$\mathtt{Matrix} = \begin{bmatrix} 6.0 & -5.0 \\ 10.0 & 4.0 \\ -3.0 & 5.0 \end{bmatrix} \qquad \mathtt{SVDType} = 'full' \qquad \mathtt{ComputeSingularVectors} = 'both'$$

$$\rightarrow \qquad \mathtt{MatrixU} = \begin{bmatrix} -0.5228 & 0.5691 & 0.6346 \\ -0.8070 & -0.5702 & -0.1535 \\ 0.2745 & -0.5924 & 0.7574 \end{bmatrix} \qquad \mathtt{MatrixS} = \begin{bmatrix} 12.0547 & 0 \\ 0 & 8.1046 \\ 0 & 0 \end{bmatrix}$$

$$\mathtt{MatrixV} = \begin{bmatrix} -0.9980 & -0.0629 \\ 0.0629 & -0.9980 \end{bmatrix}$$

Ist SVDType = *'reduced'* gesetzt, wird eine reduzierte SVD berechnet.

Beispiel:

$$\mathtt{Matrix} = \begin{bmatrix} 6.0 & -5.0 \\ 10.0 & 4.0 \\ -3.0 & 5.0 \end{bmatrix} \qquad \mathtt{SVDType} = 'reduced' \qquad \mathtt{ComputeSingularVectors} = 'both'$$

$$\rightarrow \qquad \mathtt{MatrixU} = \begin{bmatrix} -0.5228 & 0.5691 \\ -0.8070 & -0.5702 \\ 0.2745 & -0.5924 \end{bmatrix} \qquad \mathtt{MatrixS} = \begin{bmatrix} 12.0547 & 0 \\ 0 & 8.1046 \end{bmatrix}$$

$$\mathtt{MatrixV} = \begin{bmatrix} -0.9980 & -0.0629 \\ 0.0629 & -0.9980 \end{bmatrix}$$

Die Matrix MatrixU mit den linken Singulärvektoren wird berechnet, wenn ComputeSingularVectors = *'left'* gesetzt ist. Die Matrix MatrixV mit den rechten Singulärvektoren wird berechnet, wenn ComputeSingularVectors = *'right'* gesetzt ist. Die Matrizen mit den linken und rechten Singulärvektoren werden berechnet, wenn ComputeSingularVectors = *'both'* gesetzt ist.

Wenn ComputeSingularVectors = *'none'* gesetzt ist, werden keine Matrizen mit den Singulärvektoren berechnet. Die Matrix MatrixS ist eine Matrix mit n Zeilen und einer Spalte, wobei n = min(Anzahl der Zeilen der Eingabematrix Matrix, Anzahl der Spalten der Eingabematrix Matrix).

Beispiel:

$$\mathtt{Matrix} = \begin{bmatrix} 6.0 & -5.0 \\ 10.0 & 4.0 \\ -3.0 & 5.0 \end{bmatrix}$$

$$\mathtt{SVDType} = 'reduced' oder 'full' \qquad \mathtt{ComputeSingularVectors} = 'none'$$

$$\rightarrow \qquad \mathtt{MatrixS} = \begin{bmatrix} 12.0547 \\ 8.1046 \end{bmatrix}$$

Es ist zu beachten, dass in den Beispielen Unterschiede in der Bedeutung der Werte in den Ausgabematrizen zu finden sind: Wenn ein Wert als ganze Zahl dargestellt ist, z.B. 0 oder 1, ist der Wert dieses Elements per Definition dieser bestimmte Wert. Wenn der Wert als Gleitpunktzahl dargestellt ist, z.B. 0.0 oder 1.0, ist der Wert von dem Operator berechnet worden.

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **SVDType** (input_control) . . . . . string ⇝ *string*
Berechnungstyp.
Defaultwert: 'full'
Werteliste: SVDType ∈ {'full', 'reduced'}

▷ **ComputeSingularVectors** (input_control) . . . . . string ⇝ *string*
Berechnung der Singulärwerte.
Defaultwert: 'both'
Werteliste: ComputeSingularVectors ∈ {'none', 'left', 'right', 'both'}

▷ **MatrixUID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den linken Singulärvektoren.

▷ **MatrixSID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den Singulärwerten.

▷ **MatrixVID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle mit den rechten Singulärvektoren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert svd_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Literatur

David Poole: „Linear Algebra: A Modern Introduction"; Thomson; Belmont; 2006.

Gene H. Golub, Charles F. van Loan: „Matrix Computations"; The Johns Hopkins University Press; Baltimore and London; 1996.

Modul

Foundation

18.7 Zugriff

get_diagonal_matrix (: : MatrixID, Diagonal : VectorID)

Gibt die Diagonalelemente einer Matrix zurück.

Der Operator get_diagonal_matrix generiert eine neue Matrix Vector und kopiert die Diagonalelemente der Matrix Matrix in diese neue Matrix. Gegeben ist die Matrix Matrix durch das Matrix Handle MatrixID. Die Matrix Vector hat eine Spalte und n Zeilen mit n = Anzahl der Diagonalelemente. Der Operator gibt das Matrix Handle VectorID der Matrix Vector zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Wenn Diagonal = 0, wird in den Vector die Hauptdiagonale der Matrix kopiert.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \qquad \texttt{Diagonal} = 0$$

$$\rightarrow \texttt{Vector} = \begin{bmatrix} 3.0 \\ 7.0 \\ 1.0 \end{bmatrix}$$

Die obere Diagonal-te Nebendiagonale der Eingabematrix Matrix wird kopiert, wenn Diagonal eine positive Zahl ist.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 & 1.0 \\ -5.0 & 7.0 & 2.0 & 6.0 \\ -9.0 & -4.0 & 1.0 & -1.0 \end{bmatrix} \qquad \texttt{Diagonal} = 2$$

$$\rightarrow \qquad \texttt{Vector} = \begin{bmatrix} -2.0 \\ 6.0 \end{bmatrix}$$

Die untere Diagonal-te Nebendiagonale der Eingabematrix Matrix wird kopiert, wenn Diagonal eine negative Zahl ist.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 & 1.0 \\ -5.0 & 7.0 & 2.0 & 6.0 \\ -9.0 & -4.0 & 1.0 & -1.0 \end{bmatrix} \qquad \texttt{Diagonal} = -1$$

$$\rightarrow \qquad \texttt{Vector} = \begin{bmatrix} -5.0 \\ -4.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **Diagonal** (input_control) . . . . . integer ⇝ *integer*
  Nummer der gewünschten Diagonale.
  Defaultwert: 0
  Wertevorschläge: Diagonal ∈ {-20, -10, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}
- ▷ **VectorID** (output_control) . . . . . matrix ⇝ *handle*
  Matrix Handle mit den Diagonalelementen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_diagonal_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Siehe auch

set_diagonal_matrix

Modul

Foundation

get_full_matrix (: : MatrixID : Values)

Gibt alle Werte einer Matrix zurück.

Der Operator get_full_matrix gibt die Werte aller Elemente der Matrix, die durch das Matrix Handle MatrixID gegeben ist, zurück. Der Ausgabeparameter Values ist ein Tupel von Gleitpunktzahlen und beinhaltet alle Werte zeilenweise, d.h. Zeile für Zeile.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix} \quad \rightarrow \quad \texttt{Values} = [3.0, 1.0, -2.0, -5.0, 7.0, 2.0, -9.0, -4.0, 1.0]$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **Values** (output_control) . . . . . real(-array) ⇝ *real*
Werte der Matrixelemente.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_full_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

clear_matrix

Siehe auch

set_full_matrix

Modul

Foundation

get_sub_matrix (: : MatrixID, Row, Column, RowsSub, ColumnsSub : MatrixSubID)

Gibt einen Teil einer Matrix zurück.

Der Operator get_sub_matrix generiert eine neue Matrix MatrixSub und kopiert in diese einen Teil der Eingabematrix Matrix, die durch das Matrix Handle MatrixID gegeben ist. Der Teil der Matrix Matrix wird durch die obere linke Ecke (Row,Column) und die Dimension der Teilmatrix (RowsSub,ColumnsSub) festgelegt. Der Operator gibt das Matrix Handle MatrixSubID der Matrix MatrixSub zurück. Zugriff auf die Elemente der Matrix ist z.B. mit dem Operator get_full_matrix möglich.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 & 1.0 \\ -5.0 & 7.0 & 2.0 & 6.0 \\ -9.0 & -4.0 & 1.0 & -1.0 \end{bmatrix}$$

$$\texttt{Row} = 0 \quad \texttt{Column} = 1 \quad \texttt{RowsSub} = 3 \quad \texttt{ColumnsSub} = 2$$

$$\rightarrow \quad \texttt{MatrixSub} = \begin{bmatrix} 1.0 & -2.0 \\ 7.0 & 2.0 \\ -4.0 & 1.0 \end{bmatrix}$$

Achtung

Es müssen die Bedingungen $0 \leq$ Row $<$ Größe der Matrix Matrix in Zeilenrichtung, Row + RowsSub $\leq$ Größe der Matrix Matrix in Zeilenrichtung, $0 \leq$ Column $<$ Größe der Matrix Matrix in Spaltenrichtung und Column + ColumnsSub $\leq$ Größe der Matrix Matrix in Spaltenrichtung erfüllt sein.

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **Row** (input_control) . . . . . integer ⇝ *integer*
Obere Zeilenposition der Teilmatrix in der Eingabematrix.
Defaultwert: 0
Wertevorschläge: Row ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
Restriktion: Row >= 0

▷ **Column** (input_control) . . . . . integer ⇝ *integer*
Linke Spaltenposition der Teilmatrix in der Eingabematrix.
Defaultwert: 0
Wertevorschläge: Column ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
Restriktion: Column >= 0

▷ **RowsSub** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Zeilen der Teilmatrix.
Defaultwert: 1
Wertevorschläge: RowsSub ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
Restriktion: RowsSub >= 1

▷ **ColumnsSub** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Spalten der Teilmatrix.
Defaultwert: 1
Wertevorschläge: ColumnsSub ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
Restriktion: ColumnsSub >= 1

▷ **MatrixSubID** (output_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Teilmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sub_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Siehe auch

set_sub_matrix

Modul

Foundation

get_value_matrix (: : MatrixID, Row, Column : Value)

Gibt eines oder mehrere Elemente einer Matrix zurück.

Der Operator get_value_matrix gibt die Werte eines oder mehrerer Elemente der Matrix Matrix in einem Tupel von Gleitpunktzahlen zurück. Die Matrix Matrix ist durch das Matrix Handle MatrixID gegeben. Die Zeilenkoordinaten der Elemente der Matrix Matrix sind im Tupel Row und die Spaltenkoordinaten im Tupel Column festgelegt.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 & 1.0 \\ -5.0 & 7.0 & 2.0 & 6.0 \\ -9.0 & -4.0 & 1.0 & -1.0 \end{bmatrix} \qquad \texttt{Row} = [0, 2, 1] \qquad \texttt{Column} = [1, 0, 3]$$

$\rightarrow$ `Value` $= [1.0, -9.0, 6.0]$

Parameter

▷ **`MatrixID`** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **`Row`** (input_control) . . . . . integer(-array) ⇝ *integer*
Zeilennummern der zurückzugebenden Elemente der Matrix.
Defaultwert: 0
Wertevorschläge: `Row` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
Restriktion: Row >= 0

▷ **`Column`** (input_control) . . . . . integer(-array) ⇝ *integer*
Spaltennummern der zurückzugebenden Elemente der Matrix.
Defaultwert: 0
Wertevorschläge: `Column` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
Restriktion: Column >= 0

▷ **`Value`** (output_control) . . . . . real(-array) ⇝ *real*
Werte der angegebenen Matrixelemente.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_value_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_matrix`

Nachfolger

`clear_matrix`

Siehe auch

`set_value_matrix`

Modul

Foundation

`set_diagonal_matrix` (: : `MatrixID`, `VectorID`, `Diagonal` :)

Setzt die Diagonalelemente einer Matrix.

Der Operator `set_diagonal_matrix` *überschreibt* die Diagonalelemente der Matrix `Matrix` mit den Elementen der Matrix `Vector`. Die Matrizen sind durch ihre Matrix Handles `MatrixID` und `VectorID` gegeben. Die Matrix `Vector` muss eine Spalte und eine Zeile, n Spalten und eine Zeile oder eine Spalte und n Zeilen haben. n ist die Anzahl der Elemente, die in der Matrix `Matrix` gesetzt werden (siehe unten). Wenn die Matrix `Vector` eine Spalte und eine Zeile hat, d.h. die Matrix nur einen Wert hat, so wird jedes Diagonalelement der Matrix `Matrix` mit diesem Wert überschrieben. Andernfalls wird die Diagonale mit den Elementen der Matrix `Vector` überschrieben.

Wenn `Diagonal` = 0, wird die Hauptdiagonale der Matrix `Matrix` überschrieben. Die Anzahl n = min(Anzahl der Zeilen der Matrix `Matrix`, Anzahl der Spalten der Matrix `Matrix`).

Beispiel 1:

$$\texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \qquad \texttt{Vector} = \begin{bmatrix} 3.0 & 7.0 & 1.0 \end{bmatrix} \qquad \texttt{Diagonal} = 0$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 3.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 7.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}$$

Beispiel 2:

$$\texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \texttt{Vector} = \begin{bmatrix} 3.0 \end{bmatrix} \quad \texttt{Diagonal} = 0$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 3.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 3.0 & 0.0 \end{bmatrix}$$

Die obere Diagonal-te Nebendiagonale der Matrix wird überschrieben, wenn Diagonal eine positive Zahl beinhaltet. Für das Beispiel 1 ist die Anzahl n = min(Anzahl der Zeilen der Matrix Matrix, Parameter Diagonal). Für das Beispiel 2 ist die Anzahl n = min(Anzahl der Zeilen der Matrix Matrix, Anzahl der Spalten der Matrix Matrix).

Beispiel 1:

$$\texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \texttt{Vector} = \begin{bmatrix} -2.0 & 6.0 \end{bmatrix} \quad \texttt{Diagonal} = 2$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & -2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 6.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Beispiel 2:

$$\texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \texttt{Vector} = \begin{bmatrix} -2.0 \end{bmatrix} \quad \texttt{Diagonal} = 2$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & -2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -2.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Die untere Diagonal-te Nebendiagonale der Matrix wird überschrieben, wenn Diagonal eine negative Zahl beinhaltet. Für das Beispiel 1 ist die Anzahl n = min(Anzahl der Spalten der Matrix Matrix, Parameter Diagonal). Für das Beispiel 2 ist die Anzahl n = min(Anzahl der Zeilen der Matrix Matrix, Anzahl der Spalten der Matrix Matrix).

Beispiel 1:

$$\texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \texttt{Vector} = \begin{bmatrix} -5.0 & -4.0 \end{bmatrix} \quad \texttt{Diagonal} = -1$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ -5.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -4.0 & 0.0 & 0.0 \end{bmatrix}$$

Beispiel 2:

$$\texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \texttt{Vector} = \begin{bmatrix} -5.0 \end{bmatrix} \quad \texttt{Diagonal} = -1$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ -5.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -5.0 & 0.0 & 0.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **VectorID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle, welches die Diagonalelemente enthält, die gesetzt werden sollen.

▷ **Diagonal** (input_control) . . . . . integer ⇝ *integer*
Position der Diagonale.
Defaultwert: 0
Wertevorschläge: `Diagonal` ∈ {-20, -10, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_diagonal_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`get_full_matrix`, `get_value_matrix`

Siehe auch

`get_diagonal_matrix`

Modul

Foundation

set_full_matrix (: : MatrixID, Values :)

Setzt alle Werte einer Matrix.

Der Operator `set_full_matrix` *setzt* alle Werte der Eingangsmatrix `Matrix`, die durch das Matrix Handle `MatrixID` gegeben ist. Der Parameter `Values` ist hierbei ein Tupel von Gleitpunkt- bzw. ganzen Zahlen, welches die Werte enthält, die den Elementen der Matrix `Matrix` zugewiesen werden. Ganzen Zahlen werden automatisch in Gleitpunktzahlen konvertiert. In dem Parameter `Values` müssen alle Werte zeilenweise vorliegen, d.h. Zeile für Zeile gespeichert sein. Außerdem muss die Anzahl der in `Values` übergebenen Werte 1 oder mit der Anzahl aller Elemente der Matrix identisch sein.

Bemerkung: Mit dem Operator `create_matrix` kann das gleiche Ergebnis erzielt werden. Der Vorteil bei der Nutzung des Operators `set_full_matrix` liegt darin, dass eine nicht länger benötigte Matrix wiederverwendet werden kann. Dadurch verkürzt sich die Laufzeit der Operation.

Beispiel 1:

$$\texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \qquad \texttt{Values} = [3, 1, -2, -5, 7, 2, -9, -4, 1]$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 \\ -5.0 & 7.0 & 2.0 \\ -9.0 & -4.0 & 1.0 \end{bmatrix}$$

Beispiel 2:

$$\texttt{Matrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \qquad \texttt{Values} = [7]$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 7.0 & 7.0 & 7.0 \\ 7.0 & 7.0 & 7.0 \\ 7.0 & 7.0 & 7.0 \end{bmatrix}$$

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **Values** (input_control) . . . . . number(-array) ⇝ *real* / integer
Zu setzende Werte der Matrixelemente.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_full_matrix` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_matrix`

Nachfolger

`clear_matrix`

Siehe auch

`get_full_matrix`

Modul

Foundation

set_sub_matrix (: : MatrixID, MatrixSubID, Row, Column :)

Setzt eine Teilmatrix in einer Matrix.

Der Operator `set_sub_matrix` *überschreibt* einen Teil der Matrix `Matrix` mit der Matrix `MatrixSub`. Die Eingabematrizen sind durch die Matrix Handles `MatrixID` und `MatrixSubID` gegeben. Die Parameter `Row` und `Column` legen die Position der oberen linken Ecke der Teilmatrix `MatrixSub` in der Matrix `Matrix` fest.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 0.0 & 9.0 & 1.0 \\ -5.0 & 3.0 & 1.0 & 6.0 \\ -9.0 & 0.0 & 1.0 & -1.0 \end{bmatrix} \qquad \texttt{MatrixSub} = \begin{bmatrix} 1.0 & -2.0 \\ 7.0 & 2.0 \\ -4.0 & 1.0 \end{bmatrix} \qquad \texttt{Row} = 0 \qquad \texttt{Column} = 1$$

$$\rightarrow \qquad \texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 & 1.0 \\ -5.0 & 7.0 & 2.0 & 6.0 \\ -9.0 & -4.0 & 1.0 & -1.0 \end{bmatrix}$$

Achtung

Es müssen die Bedingungen $0 \leq$ `Row` $<$ Größe der Matrix `Matrix` in Zeilenrichtung, `Row` $+$ Größe der Matrix

MatrixSub in Zeilenrichtung $\leq$ Größe der Matrix Matrix in Zeilenrichtung, $0 \leq$ Column $<$ Größe der in Spaltenrichtung und Column + Größe der Matrix MatrixSub in Spaltenrichtung $\leq$ Größe der Matrix Matrix in Spaltenrichtung erfüllt sein.

Parameter

▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabematrix.

▷ **MatrixSubID** (input_control) . . . . . matrix ⇝ *handle*
Matrix Handle der Eingabeteilmatrix.

▷ **Row** (input_control) . . . . . integer ⇝ *integer*
Obere Zeilenposition der Teilmatrix in der Matrix.
Defaultwert: 0
Wertevorschläge: Row ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
Restriktion: Row >= 0

▷ **Column** (input_control) . . . . . integer ⇝ *integer*
Linke Spaltenposition der Teilmatrix in der Matrix.
Defaultwert: 0
Wertevorschläge: Column ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
Restriktion: Column >= 0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_sub_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_matrix

Nachfolger

get_full_matrix, get_value_matrix

Siehe auch

get_sub_matrix

Modul

Foundation

set_value_matrix (: : MatrixID, Row, Column, Value :)

Setzt eines oder mehrere Elemente einer Matrix.

Der Operator set_value_matrix *setzt* die Werte der Elemente der Eingangsmatrix Matrix an den Positionen (Row,Column). Die Werte sind in dem Tupel Value gegeben und können Gleitpunkt- bzw. ganze Zahlen sein. Ganze Zahlen werden automatisch in Gleitpunktzahlen konvertiert. Die Anzahl der Werte in Value muss mit der Anzahl der Elemente von Row und Column übereinstimmen. Außerdem muss die Bedingung $0 \leq$ Row $<$ Größe der Matrix in Zeilenrichtung und $0 \leq$ Column $<$ Größe der Matrix in Spaltenrichtung erfüllt sein. Die Matrix Matrix ist durch das Matrix Handle MatrixID definiert.

Beispiel:

$$\texttt{Matrix} = \begin{bmatrix} 3.0 & 0.0 & -2.0 & 1.0 \\ -5.0 & 7.0 & 2.0 & -2.0 \\ 8.0 & -4.0 & 1.0 & -1.0 \end{bmatrix} \qquad \texttt{Row} = [0,2,1] \qquad \texttt{Column} = [1,0,3] \qquad \texttt{Value} = [1,-9,6]$$

$$\rightarrow \quad \texttt{Matrix} = \begin{bmatrix} 3.0 & 1.0 & -2.0 & 1.0 \\ -5.0 & 7.0 & 2.0 & 6.0 \\ -9.0 & -4.0 & 1.0 & -1.0 \end{bmatrix}$$

Parameter

- ▷ **MatrixID** (input_control) . . . . . matrix ⇝ *handle*
  Matrix Handle der Eingabematrix.
- ▷ **Row** (input_control) . . . . . integer(-array) ⇝ *integer*
  Zeilennummern der zu modifizierenden Elemente der Matrix.
  Defaultwert: 0
  Wertevorschläge: Row ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
  Restriktion: Row >= 0
- ▷ **Column** (input_control) . . . . . integer(-array) ⇝ *integer*
  Spaltennummern der zu modifizierenden Elemente der Matrix.
  Defaultwert: 0
  Wertevorschläge: Column ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100}
  Restriktion: Column >= 0
- ▷ **Value** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Zu setzende Werte an den angegebenen Matrix Elementen.
  Defaultwert: 0
  Wertevorschläge: Value ∈ {0, 1, -1}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_value_matrix den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MatrixID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_matrix

Nachfolger

clear_matrix

Siehe auch

get_value_matrix

Modul

Foundation

Kapitel 19

Morphologie

19.1 Grauwerte

Dieses Kapitel beschreibt Operatoren der Grauwertmorphologie.

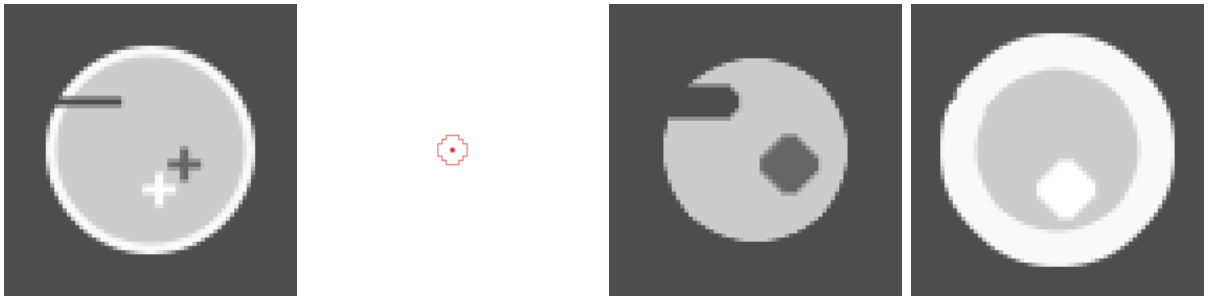
Konzept der Grauwertmorphologie

Grauwertmorpholgie stellt eine Reihe an Operatoren zur Verfügung, welche nicht-lineare Bildtransformationen ermöglichen, indem sie die Pixelwerte der Eingabebilder in Abhängigkeit ihrer Nachbarschaft verändern. Morphologische Operatoren können beispielsweise dazu verwendet werden bestimmte Bildmerkmale zu betonen oder abzuschwächen.

Im Gegensatz zu den Operatoren im Kapitel Morphologie / Region sind morphologische Grauwertoperatoren nicht auf Binärbilder beschränkt. Daher kann die Grauwertmorphologie als eine Generalisierung der Regionenmorphologie bezeichnet werden. Die morphologischen Operatoren werden in den folgenden Abschnitten näher erläutert.

Dilatation und Erosion von Grauwertbildern

Wird eine Dilatation oder Erosion durchgeführt, so wird jedem Pixel des Eingangsbildes ein Wert zugeordnet, der aus den Grauwerten der benachbarten Pixel abgeleitet wird. Welche Bildpunkte zur Nachbarschaft zählen wird durch die Form und Fläche des verwendeten Strukturelements definiert. Der Referenzpunkt des Strukturelements liegt dabei auf dem aktuell betrachteten Pixel. Bei einer Dilatation wird jedem Pixel der maximale, bei einer Erosion der minimale Grauwert seiner Nachbarschaft zugeordnet. Folglich werden helle Bereiche des Eingangsbildes durch eine Dilatation vergrößert, während eine Erosion dunklere Bildbereiche betont.



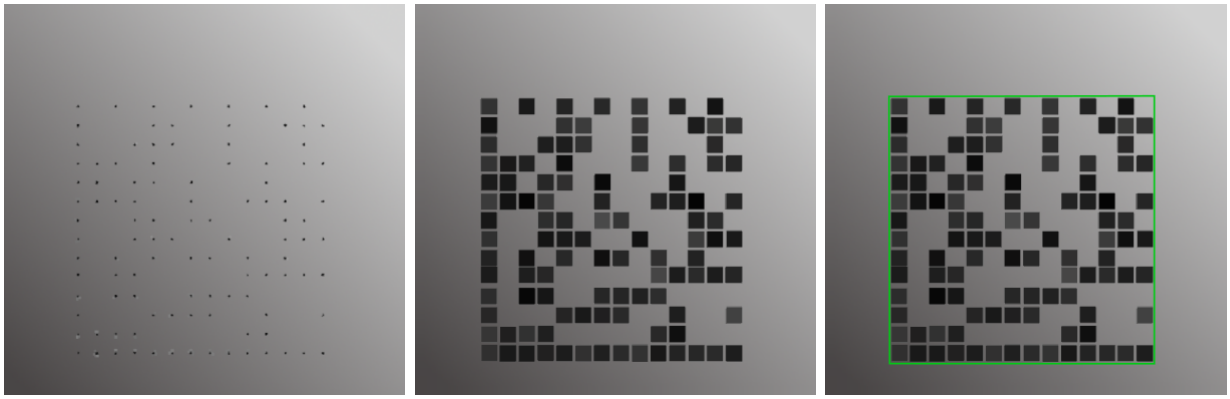
(1) (2) (3) (4)

(1) Eingangsbild, (2) Strukturelement mit Referenzpunkt im Zentrum, (3) Ergebnis einer Erosion, (4) Ergebnis einer Dilatation.

Folgende Operatoren können zur Dilatation oder Erosion eines Bildes genutzt werden:

Morphologischer Operator		Strukturelement
gray_dilation	gray_erosion	beliebig
gray_dilation_rect	gray_erosion_rect	Rechteck
gray_dilation_shape	gray_erosion_shape	Raute/Rechteck/Oktagon

Morphologische Grauwertoperatoren sind häufig Teil der Bildvorverarbeitung und werden genutzt, um die Extraktion von Informationen zu erleichtern oder erst zu ermöglichen. Im folgenden Beispiel wird ein Fall aufgezeigt, in dem eine Grauwert- Erosion notwendig ist um einen Datacode automatisch einlesbar zu machen. Die Code-Module im Eingangsbild weisen zu große Abstände zueinander auf, um für das zur Entschlüsselung gewählte Datenmodell verwendet werden zu können. Daher müssen die lokalen Minima im Bild in geeigneter Form vergrößert werden. Dazu wird eine Grauwert-Erosion mit einem quadratischen Strukturelement durchgeführt. Wie stark die Abstände zwischen den Modulen verringert werden müssen hängt von dem mit `create_data_code_2d_model` erstellten Datenmodell ab, in dem die maximal zulässige Lücke zwischen den Code-Elementen definiert ist.

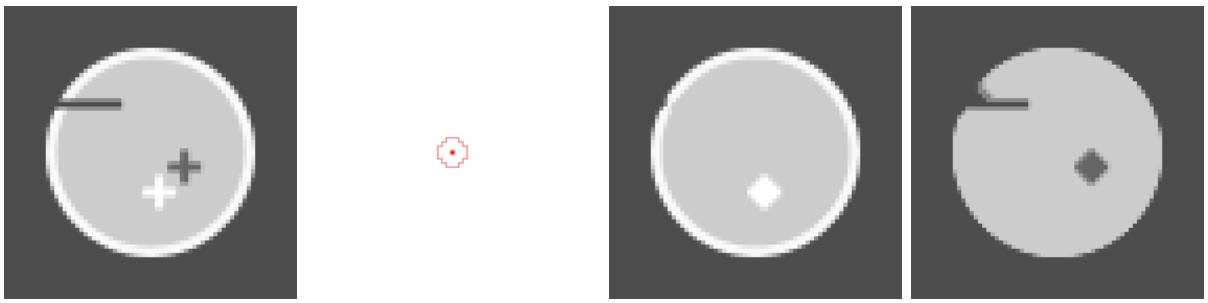


(1) (2) (3)

(1) Eingangsbild des Datacodes, (2) Erosion mit quadratischem Strukturelement, (3) Entschlüsselung des Datacodes mit Hilfe des erzeugten Datenmodells. Die Bilder stammen aus dem Beispielprogramm `2d_data_codes_minimize_module_gaps.hdev`.

Opening und Closing

Bei einem Grauwert-Opening bzw. Grauwert-Closing handelt es sich jeweils um eine Kombination der oben erläuterten Operatoren. Als Opening bezeichnet man dabei die aufeinanderfolgende Durchführung einer Erosion und einer Dilatation, während das Bild bei einem Closing zunächst dilatiert und danach erodiert wird. Wie auf den Beispielbildern zu erkennen ist, werden Bildbereiche, die im Vergleich zu ihrer Nachbarschaft dunkler sind, durch `gray_closing` reduziert oder sogar entfernt, während `gray_opening` hellere Bereiche reduziert. Zudem können durch die Verwendung des passenden Strukturelements Formen erhalten bleiben während ungewollte Artefakte entfernt werden.



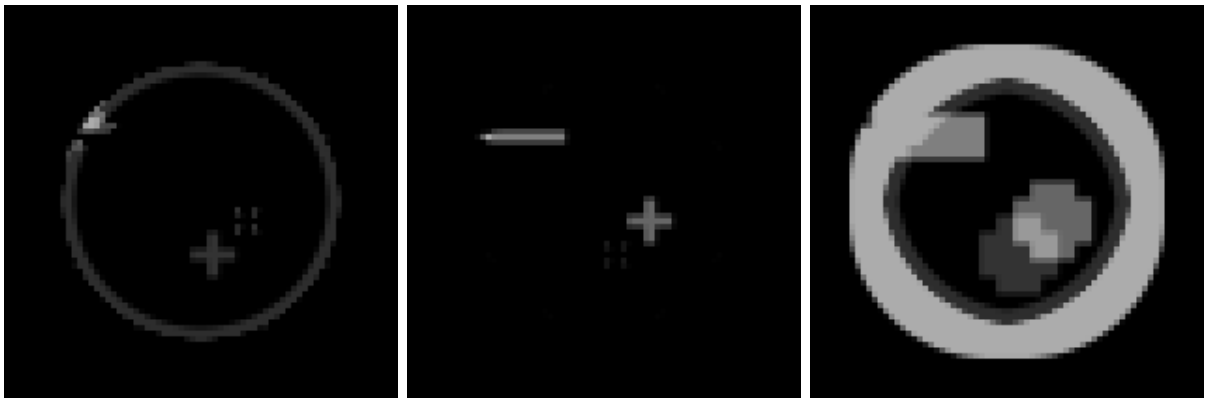
(1) (2) (3) (4)

(1) Eingangsbild, (2) Strukturelement mit Referenzpunkt im Zentrum, (3) Closing des Eingangsbildes, (4) Opening des Eingangsbildes.

Weitere Operatoren

Um die Bildbereiche, die von einem Opening bzw. Closing betroffen sind, genauer zu betrachten, kann `gray_tophat` bzw. `gray_bothat` genutzt werden. Diese Operatoren bilden die Differenz zwischen dem Originalbild und dem durch ein Opening bzw. Closing transformierten Bild. Außerdem können Bildmerkmale, die dem Strukturelement ähneln, detektiert werden.

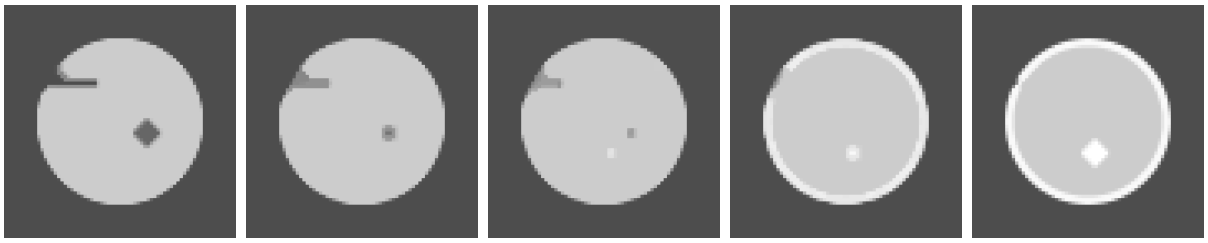
Der Operator `gray_range_rect` ermöglicht es feine Strukturen auf gleichmäßigen Oberflächen ausfindig zu machen, indem er die lokale Variation der Grauwerte visualisiert.



(1) (2) (3)

(1) Ergebnis der Top-Hat-Transformation, (2) Ergebnis der Bottom-Hat-Transformation, (3) Lokale Grauwert-Variation.

Mit Hilfe des `dual_rank`-Operators kann eine abgeschwächte Form eines Openings oder Closings ausgeführt werden. Die Steuerung dieser Transformation erfolgt über den Parameter `ModePercent`.



(1) (2) (3) (4) (5)

Dual Rank Operator: (1) `ModePercent` = 0 (entspricht Opening), (2) `ModePercent` = 25, (3) `ModePercent` = 50 (entspricht Median), (4) `ModePercent` = 75, (5) `ModePercent` = 100 (entspricht Closing).

Glossar

Im Folgenden werden die wichtigsten Begriffe, die im Zusammenhang mit Grauwertmorphologie verwendet werden, beschrieben.

nicht-linearer Operator Operator, der die Struktur des Eingangsbildes nicht notwendigerweise erhält.

Strukturelement Region, die zum Abtasten des Eingangsbildes verwendet wird.

```
dual_rank ( Image : ImageRank : MaskType, Radius, ModePercent,
    Margin : )
```

Opening, Median und Closing mit Kreis- oder Rechteckmaske.

`dual_rank` führt eine nichtlineare Transformation der Grauwerte aller Eingabebilder (`Image`) durch. Als strukturierende Elemente können Kreise oder Quadrate verwendet werden. `dual_rank` ist die Hintereinanderschaltung von zwei Aufrufen von `rank_image`. Bei dem ersten Aufruf wird der Ranggrauwert mit dem angegeben Rang (`ModePercent`) berechnet. Das Ergebnis dieser Berechnung ist die Eingabe zu einem erneuten Aufruf von `rank_image`, wobei diesmal der Rangwert 100−`ModePercent` verwendet wird.

Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (`Margin`) gewählt werden:

Grauwert	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
'continued'	Fortsetzung der Randpunkte.
'cyclic'	zyklische Fortsetzung der Bildränder.
'mirrored'	Spiegelung der Bildpunkte an den Bildrändern.

Die Berechnung einer Rangfilterung erfolgt nach folgendem Schema: Die angegebene Maske wird so über das zu filterndes Bild geschoben, dass der Schwerpunkt der Maske alle Bildpunkte einmal berührt. Für jeden solchen Bildpunkt werden alle Nachbarpunkte, die von der Maske überdeckt werden, bzgl. ihrer Grauwerte aufsteigend sortiert. Jede solche sortierte Sequenz von Grauwerten enthält somit genau so viele Grauwerte, wie die Maske Bildpunkte hat. Aus diesen Sequenzen wird das n-größte Element (= `ModePercent`, Rangwert von 0...100 in Prozent) ausgewählt und als Ergebnisgrauwert bei dem jeweiligen Ausgabebild eingetragen.

Hat `ModePercent` den Wert 0, so erhält man ein Grauwert-Opening (`gray_opening`). Setzt man `ModePercent` auf 50, so erhält man den bekannten Medianfilter (`median_image`), zweimal hintereinander angewandt. Bei dem Wert 100 berechnet `dual_rank` das Grauwert-Closing (`gray_closing`). Mit den übrigen Parametern kann fließend zwischen zwischen diesen Operationen „überblendet“ werden.

Das Konzept der Glättungsfilter ist in der Einleitung zum Kapitel Filter / Glättung beschrieben.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Zu filterndes Bild.
- ▷ **`ImageRank`** (output_object) . . . . . multichannel-image(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Gefiltertes Bild.
- ▷ **`MaskType`** (input_control) . . . . . string ⇝ *string*
  Form der Maske.
  Defaultwert: ’circle’
  Werteliste: `MaskType` ∈ {’circle’, ’square’}
- ▷ **`Radius`** (input_control) . . . . . integer ⇝ *integer*
  Radius der Filtermaske.
  Defaultwert: 1
  Wertevorschläge: `Radius` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15, 19, 25, 31, 39, 47, 59}
  Typischer Wertebereich: 1 ≤ `Radius` ≤ 101
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 2
- ▷ **`ModePercent`** (input_control) . . . . . integer ⇝ *integer*
  Modus des Filters: Bei 0 erhält man ein Grauwert-Opening, bei 50 einen Median und bei 100 ein Grauwert-Closing.
  Defaultwert: 10
  Wertevorschläge: `ModePercent` ∈ {0, 2, 5, 10, 15, 20, 40, 50, 60, 80, 85, 90, 95, 98, 100}
  Typischer Wertebereich: 0 ≤ `ModePercent` ≤ 100
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 2
- ▷ **`Margin`** (input_control) . . . . . string ⇝ *string* / integer / real
  Randbehandlung.
  Defaultwert: ’mirrored’
  Wertevorschläge: `Margin` ∈ {’mirrored’, ’cyclic’, ’continued’, 0, 30, 60, 90, 120, 150, 180, 210, 240, 255}

Beispiel

```
read_image(Image,'fabrik')
dual_rank(Image,ImageOpening,'circle',10,10,'mirrored')
dev_display(ImageOpening)
```

Komplexität

Pro Bildpunkt: O($\sqrt{F} * 10$) mit F = Fläche des strukturierenden Elementes.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dual_rank` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

`read_image`

Nachfolger

`threshold`, `dyn_threshold`, `sub_image`, `regiongrowing`

Alternativen

`rank_image`, `gray_closing`, `gray_opening`, `median_image`

Siehe auch

`gen_circle`, `gen_rectangle1`, `gray_erosion_rect`, `gray_dilation_rect`, `sigma_image`

Literatur

W. Eckstein, O. Munkelt „Extracting Objects from Digital Terrain Model“ Remote Sensing and Reconstruction for Threedimensional Objects and Scenes, SPIE Symposium on Optical Science, Engeneering, and Instrumentation, July 1995, San Diego

Modul

Foundation

gen_disc_se (: SE : Type, Width, Height, Smax :)

Erzeugen eines ellipsoidförmigen strukturierenden Elements für die Grauwertmorphologie.

`gen_disc_se` erzeugt ein ellipsoidförmiges strukturierendes Element (`SE`) für die Grauwertmorphologie von Bildern. Der Parameter `Type` gibt den Bildtyp des strukturierenden Elements an. Dieser muss dem Typ des Bildes entsprechen, das in nachfolgenden Operatoraufrufen mit diesem strukturierenden Element bearbeitet werden soll. Die Parameter `Width` und `Height` geben die Größe der beiden Achsen der Ellipse an. Der Wert `Smax` gibt den maximalen Grauwert an, den das strukturierende Element annimmt. Zur Erzeugung von strukturierenden Elementen siehe `read_gray_se`. Das strukturierende Element kann mit Hilfe des Operators `write_image` gespeichert werden. Dafür muss jedoch ein Bildformat verwendet werden, das Alpha-Kanäle unterstützt, also ’tiff’, ’jp2’ oder ’png’. Diese Dateien können dann wieder mit dem Operator `read_image` eingelesen werden.

Parameter

▷ **SE** (output_object) ..... image ⇝ *object :* byte / uint2 / real
Erzeugtes strukturierendes Element.

▷ **Type** (input_control) ..... string ⇝ *string*
Pixeltyp.
Defaultwert: ’byte’
Werteliste: `Type` ∈ {’byte’, ’uint2’, ’real’}

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des erzeugten strukturierenden Elements.
Defaultwert: 5
Wertevorschläge: `Width` ∈ {0, 1, 2, 3, 4, 5, 10, 15, 20}
Typischer Wertebereich: 0 ≤ `Width` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des erzeugten strukturierenden Elements.
Defaultwert: 5
Wertevorschläge: `Height` ∈ {0, 1, 2, 3, 4, 5, 10, 15, 20}
Typischer Wertebereich: 0 ≤ `Height` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Smax** (input_control) . . . . . number ⇝ *real* / integer
Maximaler Grauwert des erzeugten strukturierenden Elements.
Defaultwert: 0
Wertevorschläge: `Smax` ∈ {0, 1, 2, 5, 10, 20, 30, 40}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_disc_se` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`gray_erosion`, `gray_dilation`, `gray_opening`, `gray_closing`, `gray_tophat`, `gray_bothat`, `write_image`

Alternativen

`read_gray_se`, `read_image`

Siehe auch

`paint_region`, `paint_gray`, `crop_part`

Modul

Foundation

gray_bothat (Image, SE : ImageBotHat : :)

Ausführen einer Bottom-Hat-Transformation auf Bildern.

`gray_bothat` führt eine Bottom-Hat-Transformation auf dem Bild `Image` mit dem strukturierenden Element `SE` durch. Der Bildtyp des strukturierenden Elements `SE` muss zu dem Bildtyp des Eingabebildes `Image` passen. Die Bottom-Hat-Transformation eines Bildes i mit einem strukturierenden Element s ist definiert als

$$\mathit{bothat}(i, s) = (i \bullet s) - i,$$

d.h. die Differenz des Closings des Bildes mit s und des Bildes (siehe `gray_closing`). Zur Erzeugung von strukturierenden Elementen siehe `read_gray_se`.

Die Bottom-Hat-Transformation wird besonders schnell für flache strukturierende Elemente durchgeführt, d.h. strukturierende Elemente mit konstantem Grauwert innerhalb der definierenden Region.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **SE** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2 / real
Strukturierendes Element.

▷ **ImageBotHat** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
Bottom Hat.

Ergebnis

`gray_bothat` liefert genau dann den Wert TRUE, wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

read_gray_se, gen_disc_se, read_image

Nachfolger

threshold

Alternativen

gray_closing

Siehe auch

gray_tophat, top_hat, gray_erosion_rect, sub_image

Modul

Foundation

gray_closing (Image, SE : ImageClosing : :)

Ausführen eines Grauwert-Closings auf Bildern.

gray_closing führt ein Grauwert-Closing auf dem Bild Image mit dem strukturierenden Element SE durch. Der Bildtyp des strukturierenden Elements SE muss zu dem Bildtyp des Eingabebildes Image passen. Das Grauwert-Closing eines Bildes i mit einem strukturierenden Element s ist definiert als

$$i \bullet s = (i \oplus s) \ominus \check{s} \ ,$$

d.h. eine Dilatation des Bildes mit s gefolgt von einer Erosion mit dem transponierten strukturierenden Element (siehe gray_dilation und gray_erosion). Zur Erzeugung von strukturierenden Elementen siehe read_gray_se.

Das Grauwert-Closing wird besonders schnell für flache strukturierende Elemente durchgeführt, d.h. strukturierende Elemente mit konstantem Grauwert innerhalb der definierenden Region.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **SE** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Strukturierendes Element.
- ▷ **ImageClosing** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ergebnisbild.

Ergebnis

gray_closing liefert genau dann den Wert TRUE, wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

read_gray_se, gen_disc_se, read_image

Alternativen

dual_rank, gray_closing_rect, gray_closing_shape

Siehe auch

closing, gray_dilation, gray_erosion

Modul

Foundation

gray_closing_rect (Image : ImageClosing : MaskHeight, MaskWidth :)

Grauwert-Closing mit einer rechteckigen Filtermaske.

gray_closing_rect führt ein Grauwert-Closing auf dem Bild Image mit Hilfe einer rechteckigen Filtermaske (MaskHeight, MaskWidth) durch. Das Ergebnis wird in ImageClosing zurückgeliefert. Die Steuerparameter MaskHeight,MaskWidth werden, wenn sie einen geraden Wert haben, auf den nächstgrößeren ungeraden Wert transformiert. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Das Grauwert-Closing eines Bildes i mit dem rechteckigen strukturierenden Element s ist definiert als

$$i \bullet s = (i \oplus s) \ominus s \ ,$$

d.h. eine Dilatation des Bildes mit s gefolgt von einer Erosion mit s (siehe gray_dilation_rect und gray_erosion_rect).

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / uint2 / int2 / int4 / real

Eingabebild.

▷ **ImageClosing** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / uint2 / int2 / int4 / real

Ergebnisbild.

▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*

Höhe der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskHeight ≤ 511
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskHeight) && MaskHeight < height(Image) * 2

▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*

Breite der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskWidth ≤ 511
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth) && MaskWidth < width(Image) * 2

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gray_closing_rect den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

gray_closing, gray_closing_shape

Siehe auch

closing_rectangle1, gray_dilation_rect, gray_erosion_rect

Modul

Foundation

gray_closing_shape (Image : ImageClosing : MaskHeight, MaskWidth, MaskShape :)

Grauwert-Closing mit einer ausgewählten Maske.

gray_closing_shape führt ein Grauwert-Closing auf dem Bild Image mit einem strukturierenden Element der Form MaskShape mit der vertikalen Ausdehnung MaskHeight und der horizontalen Ausdehnung MaskWidth durch. Die Offsetwerte der Maske sind auf 0 festgelegt. Das Ergebnis wird in ImageClosing zurückgeliefert.

Die Steuerparameter MaskHeight und MaskWidth werden, wenn sie vom Typ Integer sind und einen geraden Wert haben, auf den nächstgrößeren ungeraden Wert transformiert. Ist dagegen mindestens einer der beiden Parameter vom Typ Float, dann wird das Eingabebild Image zunächst mit der jeweils nächstgrößeren und nächstkleineren ungeraden Maskengröße gefiltert und das Ergebnisbild ImageClosing aus den beiden Zwischenergebnissen interpoliert. Es ist daher zu beachten, dass gray_closing_shape für Maskengrößen von z.B. *4* und *4.0* unterschiedliche Ergebnisse liefert!

Im Falle der Maskenformen 'rhombus' und 'octagon' für Steuerparameter MaskShape müssen die Werte für MaskHeight und MaskWidth gleich groß sein. Der Parameterwert 'octagon' für MaskShape beschreibt eine gleichseitige, achteckige Maske, welche sich als Näherung von kreisförmigen Maskenstrukturen eignet. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Das Grauwert-Closing eines Bildes i mit einem strukturierenden Element s ist definiert als

$$i \bullet s = (i \oplus s) \ominus s \ ,$$

d.h. eine Dilatation des Bildes mit s gefolgt von einer Erosion mit s (siehe gray_dilation_shape und gray_erosion_shape).

Achtung

Es ist zu beachten, dass gray_closing_shape für Maskengrößen des Typs Float spürbar mehr Rechenzeit benötigt als für Maskengrößen des Typs Integer. Dies gilt insbesondere für rechteckige Masken mit unterschiedlicher Breite und Höhe!

gray_closing_shape kann auf OpenCL-Geräten ausgeführt werden. Wird eine Maskengröße des Typs Float verwendet, kann das Ergebnis geringfügig von dem der CPU abweichen, da die Interpolation auf dem OpenCL-Gerät in einfacher Genauigkeit durchgeführt wird.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Bilder, für deren Grauwerte die Minima berechnet werden sollen.

▷ **ImageClosing** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Bilder, die die Minima enthalten.

▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *real* / integer
Höhe der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 1.0 ≤ MaskHeight ≤ 511.0

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *real* / integer
Breite der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 1.0 ≤ MaskWidth ≤ 511.0

▷ **MaskShape** (input_control) . . . . . string ⇝ *string*
Form der Maske.
Defaultwert: 'octagon'
Werteliste: MaskShape ∈ {'rectangle', 'rhombus', 'octagon'}

Ergebnis

gray_closing_shape liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

gray_closing

Siehe auch

gray_dilation_shape, gray_erosion_shape, closing

Modul

Foundation

gray_dilation (Image, SE : ImageDilation : :)

Ausführen einer Grauwert-Dilatation auf Bildern.

gray_dilation führt eine Grauwertdilatation auf dem Bild Image mit dem strukturierenden Element SE durch. Der Bildtyp des strukturierenden Elements SE muss zu dem Bildtyp des Eingabebildes Image passen. Die Grauwertdilatation eines Bildes i mit einem strukturierenden Element s an der Pixel-Position x ist definiert durch:

$$(i \oplus s)(x) = \max\{f(x - z) + s(z) | z \in S\}$$

Hierbei ist S der Definitionsbereich des strukturierenden Elements s, d.h. dessen Region (siehe read_gray_se).

Die Grauwertdilatation wird besonders schnell für flache strukturierende Elemente durchgeführt, d.h. strukturierende Elemente mit konstantem Grauwert innerhalb der definierenden Region.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **SE** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2 / real
Strukturierendes Element.

▷ **ImageDilation** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
Ergebnisbild.

Ergebnis

`gray_dilation` liefert genau dann den Wert TRUE, wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_gray_se`, `gen_disc_se`, `read_image`

Nachfolger

`sub_image`, `gray_erosion`

Alternativen

`gray_dilation_rect`, `gray_dilation_shape`

Siehe auch

`gray_opening`, `gray_closing`, `dilation1`, `gray_skeleton`

Modul

Foundation

gray_dilation_rect (Image : ImageMax : MaskHeight, MaskWidth :)

Maximum der Grauwerte in einem Rechteck.

`gray_dilation_rect` transformiert die Grauwerte der Eingabebilder aus `Image` mit Hilfe einer Filtermaske (`MaskHeight`, `MaskWidth`), in der das Maximum der Grauwerte berechnet wird. Das Ergebnis wird in `ImageMax` zurückgeliefert. Die Steuerparameter `MaskHeight`,`MaskWidth` werden, wenn sie einen geraden Wert haben, auf den nächstgrößeren ungeraden Wert transformiert. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / uint2 / int2 / int4 / real
Bilder, für deren Grauwerte die Maxima berechnet werden sollen.

▷ **ImageMax** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / uint2 / int2 / int4 / real
Bilder, die die Maxima enthalten.

▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
Höhe der Filtermaske.
Defaultwert: 11
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ `MaskHeight` ≤ 511
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskHeight) && MaskHeight < height(Image) * 2

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
Breite der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskWidth ≤ 511
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth) && MaskWidth < width(Image) * 2

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gray_dilation_rect den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

gray_skeleton

Modul

Foundation

gray_dilation_shape (Image : ImageMax : MaskHeight, MaskWidth, MaskShape :)

Maximum der Grauwerte in einer ausgewählten Maske.

gray_dilation_shape transformiert die Grauwerte der Eingabebilder aus Image mit Hilfe einer Filtermaske der Größe MaskHeight und MaskWidth der Form MaskShape, in der das Maximum der Grauwerte berechnet wird. Das Ergebnis wird in ImageMax zurückgeliefert.

Die Steuerparameter MaskHeight und MaskWidth werden, wenn sie vom Typ Integer sind und einen geraden Wert haben, auf den nächstgrößeren ungeraden Wert transformiert. Ist dagegen mindestens einer der beiden Parameter vom Typ Float, dann wird das Eingabebild Image zunächst mit der jeweils nächstgrößeren und nächstkleineren ungeraden Maskengröße gefiltert und das Ergebnisbild ImageMax aus den beiden Zwischenergebnissen interpoliert. Es ist daher zu beachten, dass gray_dilation_shape für Maskengrößen von z.B. *4* und *4.0* unterschiedliche Ergebnisse liefert!

Im Falle der Maskenformen 'rhombus' und 'octagon' für Steuerparameter MaskShape müssen die Werte für MaskHeight und MaskWidth gleich groß sein. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Achtung

Es ist zu beachten, dass gray_dilation_shape für Maskengrößen des Typs Float spürbar mehr Rechenzeit benötigt als für Maskengrößen des Typs Integer. Dies gilt insbesondere für rechteckige Masken mit unterschiedlicher Breite und Höhe!

gray_dilation_shape kann auf OpenCL-Geräten ausgeführt werden. Wird eine Maskengröße des Typs Float verwendet, kann das Ergebnis geringfügig von dem der CPU abweichen, da die Interpolation auf dem OpenCL-Gerät in einfacher Genauigkeit durchgeführt wird.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Bilder, für deren Grauwerte die Maxima berechnet werden sollen.

▷ **ImageMax** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
Bilder, die die Maxima enthalten.

▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *real* / integer
Höhe der Filtermaske.
Defaultwert: 11
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 1.0 ≤ MaskHeight

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *real* / integer
Breite der Filtermaske.
Defaultwert: 11
Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 1.0 ≤ MaskWidth

▷ **MaskShape** (input_control) . . . . . string ⇝ *string*
Form der Maske.
Defaultwert: 'octagon'
Werteliste: `MaskShape` ∈ {'rectangle', 'rhombus', 'octagon'}

Ergebnis

`gray_dilation_shape` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`gray_dilation`, `gray_dilation_rect`

Siehe auch

`gray_opening_shape`, `gray_closing_shape`, `gray_skeleton`

Modul

Foundation

gray_erosion (Image, SE : ImageErosion : :)

Ausführen einer Grauwert-Erosion auf Bildern.

`gray_erosion` führt eine Grauwerterosion auf dem Bild `Image` mit dem strukturierenden Element `SE` durch. Der Bildtyp des strukturierenden Elements `SE` muss zu dem Bildtyp des Eingabebildes `Image` passen. Die Grauwerterosion eines Bildes i mit einem strukturierenden Element s an der Pixel-Position x ist definiert durch:

$$(i \ominus s)(x) = \min\{f(x+z) - s(z) | z \in S\}$$

Hierbei ist S der Definitionsbereich des strukturierenden Elements s, d.h. dessen Region (siehe `read_gray_se`).

Die Grauwerterosion wird besonders schnell für flache strukturierende Elemente durchgeführt, d.h. strukturierende Elemente mit konstantem Grauwert innerhalb der definierenden Region.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **SE** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2 / real
Strukturierendes Element.

▷ **ImageErosion** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
Ergebnisbild.

Ergebnis

`gray_erosion` liefert genau dann den Wert TRUE, wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_gray_se`, `gen_disc_se`, `read_image`

Nachfolger

`gray_dilation`, `sub_image`

Alternativen

`gray_erosion_rect`, `gray_erosion_shape`

Siehe auch

`gray_opening`, `gray_closing`, `erosion1`, `gray_skeleton`

Modul

Foundation

gray_erosion_rect (Image : ImageMin : MaskHeight, MaskWidth :)

Minimum der Grauwerte in einem Rechteck.

`gray_erosion_rect` transformiert die Grauwerte der Eingabebilder aus `Image` mit Hilfe einer Filtermaske (`MaskHeight`, `MaskWidth`), in der das Minimum der Grauwerte berechnet wird. Das Ergebnis wird im Parameter `ImageMin` übergeben. Die Steuerparameter `MaskHeight`, `MaskWidth` werden, wenn sie einen geraden Wert haben, in den nächstgrößeren ungeraden Wert transformiert. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / uint2 / int2 / int4 / real
Bilder, für deren Grauwerte die Minima berechnet werden sollen.

▷ **ImageMin** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / uint2 / int2 / int4 / real
Bilder, die die Minima enthalten.

▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
Höhe der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskHeight ≤ 511 (lin)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskHeight) && MaskHeight < height(Image) * 2

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
Breite der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskWidth ≤ 511 (lin)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth) && MaskWidth < width(Image) * 2

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gray_erosion_rect den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Siehe auch

gray_dilation_rect

Modul

Foundation

gray_erosion_shape (Image : ImageMin : MaskHeight, MaskWidth, MaskShape :)

Minimum der Grauwerte in einer ausgewählten Maske.

gray_erosion_shape transformiert die Grauwerte der Eingabebilder aus Image mit Hilfe einer Filtermaske der Größe MaskHeight und MaskWidth der Form MaskShape, in der das Minimum der Grauwerte berechnet wird. Das Ergebnis wird in ImageMin zurückgeliefert.

Die Steuerparameter MaskHeight und MaskWidth werden, wenn sie vom Typ Integer sind und einen geraden Wert haben, auf den nächstgrößeren ungeraden Wert transformiert. Ist dagegen mindestens einer der beiden Parameter vom Typ Float, dann wird das Eingabebild Image zunächst mit der jeweils nächstgrößeren und nächstkleineren ungeraden Maskengröße gefiltert und das Ergebnisbild ImageMin aus den beiden Zwischenergebnissen interpoliert. Es ist daher zu beachten, dass gray_erosion_shape für Maskengrößen von z.B. *4* und *4.0* unterschiedliche Ergebnisse liefert!

Im Falle der Maskenformen 'rhombus' und 'octagon' für Steuerparameter MaskShape müssen die Werte für MaskHeight und MaskWidth gleich groß sein. Der Parameterwert 'octagon' für MaskShape beschreibt eine gleichseitige, achteckige Maske, welche sich als Näherung von kreisförmigen Maskenstrukturen eignet. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Achtung

Es ist zu beachten, dass `gray_erosion_shape` für Maskengrößen des Typs Float spürbar mehr Rechenzeit benötigt als für Maskengrößen des Typs Integer. Dies gilt insbesondere für rechteckige Masken mit unterschiedlicher Breite und Höhe!

`gray_erosion_shape` kann auf OpenCL-Geräten ausgeführt werden. Wird eine Maskengröße des Typs Float verwendet, kann das Ergebnis geringfügig von dem der CPU abweichen, da die Interpolation auf dem OpenCL-Gerät in einfacher Genauigkeit durchgeführt wird.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Bilder, für deren Grauwerte die Minima berechnet werden sollen.
- ▷ **`ImageMin`** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Bilder, die die Minima enthalten.
- ▷ **`MaskHeight`** (input_control) . . . . . extent.y ⇝ *real* / integer
  Höhe der Filtermaske.
  Defaultwert: 11
  Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11, 13, 15}
  Typischer Wertebereich: 1.0 ≤ MaskHeight
- ▷ **`MaskWidth`** (input_control) . . . . . extent.x ⇝ *real* / integer
  Breite der Filtermaske.
  Defaultwert: 11
  Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9, 11, 13, 15}
  Typischer Wertebereich: 1.0 ≤ MaskWidth
- ▷ **`MaskShape`** (input_control) . . . . . string ⇝ *string*
  Form der Maske.
  Defaultwert: 'octagon'
  Werteliste: `MaskShape` ∈ {'rectangle', 'rhombus', 'octagon'}

Ergebnis

`gray_erosion_shape` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`gray_erosion`, `gray_erosion_rect`

Siehe auch

`gray_opening_shape`, `gray_closing_shape`, `gray_skeleton`

Modul

Foundation

`gray_opening` (Image, SE : ImageOpening : :)

Ausführen eines Grauwert-Openings auf Bildern.

`gray_opening` führt ein Grauwert-Opening auf dem Bild `Image` mit dem strukturierenden Element `SE` durch. Der Bildtyp des strukturierenden Elements `SE` muss zu dem Bildtyp des Eingabebildes `Image` passen. Das Grauwert-Opening eines Bildes i mit einem strukturierenden Element s ist definiert als

$$i \circ s = (i \ominus s) \oplus \check{s} \ ,$$

d.h. eine Erosion des Bildes mit s gefolgt von einer Dilatation mit dem transponierten strukturierenden Element (siehe gray_erosion und gray_dilation). Zur Erzeugung von strukturierenden Elementen siehe read_gray_se.

Das Grauwert-Opening wird besonders schnell für flache strukturierende Elemente durchgeführt, d.h. strukturierende Elemente mit konstantem Grauwert innerhalb der definierenden Region.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **SE** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Strukturierendes Element.
- ▷ **ImageOpening** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ergebnisbild.

Ergebnis

gray_opening liefert genau dann den Wert TRUE, wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

read_gray_se, gen_disc_se, read_image

Alternativen

dual_rank, gray_opening_rect, gray_opening_shape

Siehe auch

opening, gray_dilation, gray_erosion

Modul

Foundation

gray_opening_rect (Image : ImageOpening : MaskHeight, MaskWidth :)

Grauwert-Opening mit einer rechteckigen Filtermaske.

gray_opening_rect führt ein Grauwert-Opening auf dem Bild Image mit Hilfe einer rechteckigen Filtermaske (MaskHeight, MaskWidth) durch. Das Ergebnis wird in ImageOpening zurückgeliefert. Die Steuerparameter MaskHeight,MaskWidth werden, wenn sie einen geraden Wert haben, auf den nächstgrößeren ungeraden Wert transformiert. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Das Grauwert-Opening eines Bildes i mit dem rechteckigen strukturierenden Element s ist definiert als

$$i \circ s = (i \ominus s) \oplus s \ ,$$

d.h. eine Erosion des Bildes mit s gefolgt von einer Dilatation mit s (siehe gray_erosion_rect und gray_dilation_rect).

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / uint2 / int2 / int4 / real
Eingabebild.

▷ **ImageOpening** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / uint2 / int2 / int4 / real
Ergebnisbild.

▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
Höhe der Filtermaske.
Defaultwert: 11
Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ `MaskHeight` ≤ 511
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskHeight) && MaskHeight < height(Image) * 2

▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
Breite der Filtermaske.
Defaultwert: 11
Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ `MaskWidth` ≤ 511
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth) && MaskWidth < width(Image) * 2

Ergebnis

Sind die Parameterwerte korrekt, liefert `gray_opening_rect` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`gray_opening`, `gray_opening_shape`

Siehe auch

`opening_rectangle1`, `gray_dilation_rect`, `gray_erosion_rect`

Modul

Foundation

gray_opening_shape (Image : ImageOpening : MaskHeight, MaskWidth, MaskShape :)

Grauwert-Opening mit einer ausgewählten Filtermaske.

`gray_opening_shape` führt ein Grauwert-Opening auf dem Bild `Image` mit einem strukturierenden Element der Form `MaskShape`mit der vertikalen Ausdehnung `MaskHeight` und der horizontalen Ausdehnung `MaskWidth` durch. Die Offsetwerte der Maske sind auf 0 festgelegt. Das Ergebnis wird in `ImageOpening` zurückgeliefert.

Die Steuerparameter `MaskHeight` und `MaskWidth` werden, wenn sie vom Typ Integer sind und einen geraden Wert haben, auf den nächstgrößeren ungeraden Wert transformiert. Ist dagegen mindestens einer der beiden

Parameter vom Typ Float, dann wird das Eingabebild `Image` zunächst mit der jeweils nächstgrößeren und nächstkleineren ungeraden Maskengröße gefiltert und das Ergebnisbild `ImageOpening` aus den beiden Zwischenergebnissen interpoliert. Es ist daher zu beachten, dass `gray_opening_shape` für Maskengrößen von z.B. *4* und *4.0* unterschiedliche Ergebnisse liefert!

Im Falle der Maskenformen 'rhombus' und 'octagon' für Steuerparameter `MaskShape` müssen die Werte für `MaskHeight` und `MaskWidth` gleich groß sein. Der Parameterwert 'octagon' für `MaskShape` beschreibt eine gleichseitige, achteckige Maske, welche sich als Näherung von kreisförmigen Maskenstrukturen eignet. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Das Grauwert-Opening eines Bildes i mit einem strukturierenden Element s ist definiert als

$$i \circ s = (i \ominus s) \oplus s \ ,$$

d.h. eine Dilatation des Bildes mit s gefolgt von einer Erosion mit s (siehe `gray_dilation_shape` und `gray_erosion_shape`).

Achtung

Es ist zu beachten, dass `gray_opening_shape` für Maskengrößen des Typs Float spürbar mehr Rechenzeit benötigt als für Maskengrößen des Typs Integer. Dies gilt insbesondere für rechteckige Masken mit unterschiedlicher Breite und Höhe!

`gray_opening_shape` kann auf OpenCL-Geräten ausgeführt werden. Wird eine Maskengröße des Typs Float verwendet, kann das Ergebnis geringfügig von dem der CPU abweichen, da die Interpolation auf dem OpenCL-Gerät in einfacher Genauigkeit durchgeführt wird.

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Bilder, für deren Grauwerte die Minima berechnet werden sollen.
- ▷ **`ImageOpening`** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2
  Bilder, die die Minima enthalten.
- ▷ **`MaskHeight`** (input_control) ..... extent.y ⇝ *real* / integer
  Höhe der Filtermaske.
  Defaultwert: 11
  Wertevorschläge: `MaskHeight` ∈ {3, 5, 7, 9, 11, 13, 15}
  Typischer Wertebereich: 1.0 ≤ `MaskHeight` ≤ 511.0
- ▷ **`MaskWidth`** (input_control) ..... extent.x ⇝ *real* / integer
  Breite der Filtermaske.
  Defaultwert: 11
  Wertevorschläge: `MaskWidth` ∈ {3, 5, 7, 9, 11, 13, 15}
  Typischer Wertebereich: 1.0 ≤ `MaskWidth` ≤ 511.0
- ▷ **`MaskShape`** (input_control) ..... string ⇝ *string*
  Form der Maske.
  Defaultwert: 'octagon'
  Werteliste: `MaskShape` ∈ {'rectangle', 'rhombus', 'octagon'}

Ergebnis

`gray_opening_shape` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

gray_opening

Siehe auch

gray_dilation_shape, gray_erosion_shape, opening

Modul

Foundation

gray_range_rect (Image : ImageResult : MaskHeight, MaskWidth :)

Maximale Amplitude der Grauwerte in einem Rechteck.

gray_range_rect transformiert die Grauwerte der Eingabebilder aus Image mit Hilfe einer Filtermaske (MaskHeight, MaskWidth), in der die Spanne $(\max - \min)$ der Grauwerte berechnet wird. Das Ergebnis wird in die Ausgabebilder ImageResult eingetragen. Die Steuerparameter MaskHeight, MaskWidth werden, wenn sie einen geraden Wert haben, in den nächstkleineren ungeraden Wert transformiert. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / cyclic / uint2 / int2 / int4 / real
Bilder, für deren Grauwerte die Amplitude berechnet werden sollen.

▷ **ImageResult** (output_object) . . . . . (multichannel-)image(-array) ⇝ *object :* byte / cyclic / uint2 / int2 / int4 / real
Bilder, die das Ergebnis enthalten.

▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
Höhe der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskHeight ≤ 511 (lin)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskHeight) && MaskHeight < height(Image) * 2

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
Breite der Filtermaske.
Defaultwert: 11
Wertevorschläge: MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich: 3 ≤ MaskWidth ≤ 511 (lin)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: odd(MaskWidth) && MaskWidth < width(Image) * 2

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gray_range_rect den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt OpenCL Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

gray_dilation_rect, gray_erosion_rect, sub_image

Modul

Foundation

gray_tophat (Image, SE : ImageTopHat : :)

Ausführen einer Top-Hat-Transformation auf Bildern.

gray_tophat führt eine Top-Hat-Transformation auf dem Bild Image mit dem strukturierenden Element SE durch. Der Bildtyp des strukturierenden Elements SE muss zu dem Bildtyp des Eingabebildes Image passen. Die Top-Hat-Transformation eines Bildes i mit einem strukturierenden Element s ist definiert als

$$tophat(i, s) = i - (i \circ s),$$

d.h. die Differenz des Bildes und seines Openings mit s (siehe gray_opening). Zur Erzeugung von strukturierenden Elementen siehe read_gray_se und gen_disc_se.

Die Top-Hat-Transformation wird besonders schnell für flache strukturierende Elemente durchgeführt, d.h. strukturierende Elemente mit konstantem Grauwert innerhalb der definierenden Region.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **SE** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Strukturierendes Element.
- ▷ **ImageTopHat** (output_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / uint2 / real
  Ergebnisbild.

Ergebnis

gray_tophat liefert genau dann den Wert TRUE, wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Kanalebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

read_gray_se, gen_disc_se, read_image

Nachfolger

threshold

Alternativen

gray_opening

Siehe auch

gray_bothat, top_hat, gray_erosion_rect, sub_image

Modul

Foundation

read_gray_se (: SE : FileName :)

Laden von strukturierenden Elementen für die Grauwertmorphologie.

read_gray_se lädt ein strukturierendes Element für die Grauwertmorphologie von einer Datei ein. Die Dateinamen für strukturierende Elemente müssen mit '.gse' (für grayscale structuring element) enden. Dieses Suffix wird von read_gray_se automatisch an den übergebenen Dateinamen angehängt, darf also nicht mit übergeben werden. Die Daten müssen im ASCII-Format vorliegen und wie folgt aufgebaut sein: Die erste Zeile gibt den Datentyp des strukturierenden Elements an. Dieser muss 'byte', 'uint2' oder 'real' (ohne umschließende Anführungszeichen) sein und muss dem Typ des Bildes entsprechen das in nachfolgenden Operatoraufrufen mit diesem strukturierenden Element bearbeitet werden soll. Die nachfolgenden zwei Zahlen der Datei geben die Breite und Höhe des strukturierenden Elements an. Diese stellen ein Rechteck dar, das das strukturierende Element vollständig umschließt. Beide Werte müssen größer als 0 sein. Danach folgt die Anzahl der Sehnen der zu definierenden Region und die Sehnen selbst: eine Sehne besteht dabei aus Zeile der Sehne (L), Spaltenbeginn der Sehne (CB) und Spaltenende der Sehne (CE), jeweils durch Leerzeichen getrennt. Danach folgen Zahlen, deren Anzahl durch Breite und Höhe des strukturierenden Elements festgelegt wurden, die durch Leerzeichen getrennt sein müssen und als entsprechende Grauwerte für die Morphologie interpretiert werden. Zahlen, die nicht Teil der Region des strukturierenden Elements sein werden (weil für diese Punkte keine Sehne definiert wurde), dürfen nicht weggelassen werden. Sie werden jedoch im resultierenden strukturierenden Element ignoriert. Die Darstellung von strukturierenden Elementen erfolgt als Bild des o.a. Bildtyps. Es können auch gewöhnliche Bilder als strukturierende Elemente verwendet werden. Die Region des Bildes definiert dabei die Form des strukturierenden Elements. Dabei sollte allerdings darauf geachtet werden, dass die Bildabmessungen nicht zu groß sind, da die Laufzeit proportional zur Fläche des Bildes multipliziert mit der Fläche des strukturierenden Elements ist. Flache strukturierende Elemente (d.h. Elemente mit konstanten Grauwerten) werden jedoch in nachfolgenden Operatoren besonders schnell ausgeführt. Der Koordinatenursprung des strukturierenden Elements liegt bei der Hälfte der definierten Breite bzw. Höhe. Bei geraden Abmessungen wird der Ursprung in Richtung linke obere Ecke abgerundet. Das strukturierende Element kann mit Hilfe des Operators write_image gespeichert werden. Dafür muss jedoch ein Bildformat verwendet werden das Alpha-Kanäle unterstützt, also 'tiff', 'jp2' oder 'png'. Diese Dateien können dann wieder mit dem Operator read_image eingelesen werden.

Parameter

▷ **SE** (output_object) ..... image ⇝ *object :* byte / uint2 / real
Erzeugtes strukturierendes Element.

▷ **FileName** (input_control) ..... filename.read ⇝ *string*
Name der einzulesenden Datei.
Dateiendung: .gse

Beispiel

```
* Load a byte structuring element of width 3 and height 3 that
* consists of 3 chords at lines 0 (from x=1 to x=1), 1 (from x=0
* to x=2) and 2 (from x=1 to x=1). The (used) grayvalues are
* 0, 0, 1, 0, 0 .
* contents of file 'isod4.gse':
* byte
*  3 3
*
* 3
* 0 1 1
* 1 0 2
* 2 1 1
*
*  -1  0 -1
*   0  1  0
*  -1  0 -1
read_gray_se (SE, environment('HALCONROOT')+'/filter/isod4')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_gray_se den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

gray_erosion, gray_dilation, gray_opening, gray_closing, gray_tophat, gray_bothat, write_image

Alternativen

gen_disc_se, read_image, change_domain

Siehe auch

paint_region, paint_gray, crop_part

Modul

Foundation

19.2 Region

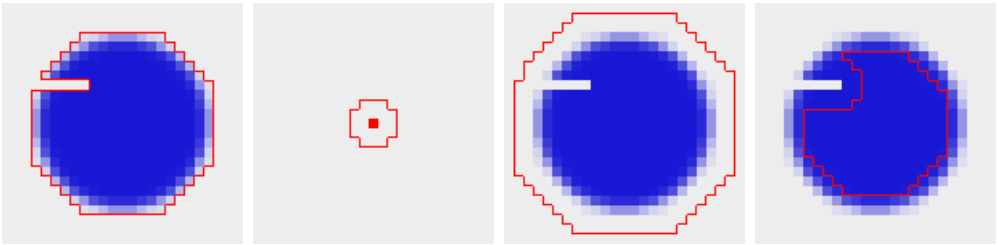
Dieses Kapitel beschreibt Operatoren der Regionenmorphologie.

Konzept der Regionenmorphologie

Die Regionenmorphologie bietet eine Reihe von morphologischen Operatoren, mit denen die Form einer Region verändert oder beschrieben werden kann. Morphologische Operatoren können beispielsweise verwendet werden, um benachbarte Regionen zu verschmelzen oder zu trennen. Eine weitere Anwendung ist das Glätten der Ränder einer Region. Im Folgenden werfen wir einen genaueren Blick auf die morphologischen Operatoren.

Dilatation und Erosion

Zum Dilatieren oder Erodieren einer Eingaberegion wird ein strukturierendes Element auf die Eingaberegion angewendet. Dieses strukturierende Element tastet das Bild zeilenweise ab. Während der Dilatation wird der Referenzpunkt des strukturierenden Elementes zu der resultierenden Region hinzugefügt, sobald das strukturierende Element und die Eingaberegion mindestens einen Pixel gemeinsam haben. Erosion verkleinert die Fläche der Eingaberegion, da der Referenzpunkt des strukturierenden Elementes nur zu der resultierenden Region hinzugefügt wird, wenn das strukturierende Element vollständig innerhalb der Eingaberegion liegt. Infolgedessen kann Erosion alternativ verwendet werden, um Objekte zu finden.



(1) (2) (3) (4)

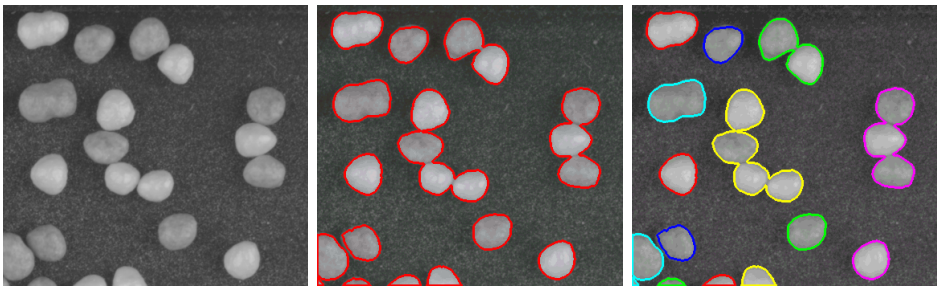
(1) Bild mit Eingaberegion, (2) strukturierendes Element mit Referenzpunkt im Ursprung der Region, (3) Ergebnis der Dilatation, (4) Ergebnis der Erosion.

Diese Operatoren können zum Dilatieren oder Erodieren einer Region verwendet werden:

Morphologische Operatoren		Strukturierendes Element	Referenzpunkt
dilation1	erosion1	beliebig	Ursprung
minkowski_add1	minkowski_sub1	beliebig, transponiert	Ursprung
dilation2	erosion2	beliebig	beliebig
minkowski_add2	minkowski_sub2	beliebig, transponiert	beliebig
dilation_circle	erosion_circle	kreisförmig	Ursprung
dilation_rectangle1	erosion_rectangle1	rechteckig	Ursprung

Die Minkowski-Addition und die Dilatation sind identisch, wenn das strukturierende Element symmetrisch ist. Das Gleiche gilt für die Minkowski-Subtraktion und die Erosion. Mit Erosion lassen sich miteinander verschmolzene Objekte separieren. Im Folgenden werden die dazu notwendigen Schritte kurz beschrieben.

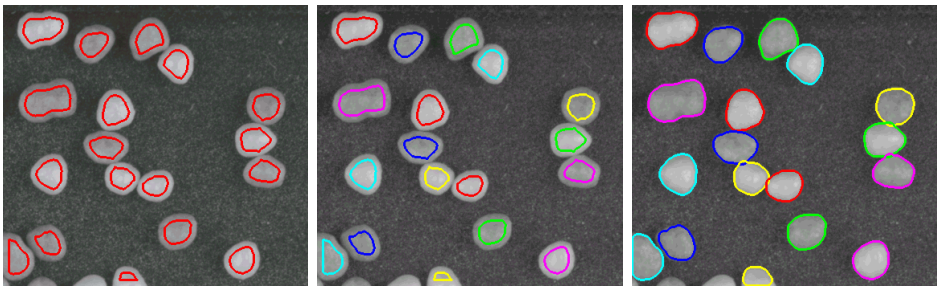
Zuerst müssen die Objekte des Bildes segmentiert werden, zum Beispiel durch die Verwendung des Operators `threshold`. Als Nächstes wird der Operator `connection` verwendet, um mehrere Regionen anstelle einer einzigen Region zu erhalten. Wie folgendes Bild (3) zeigt, ist das Ergebnis unbefriedigend, da mehrere Regionen miteinander verschmolzen sind.



(1) (2) (3)

(1) Bild mit kugelförmigen Objekten, (2) segmentierte Regionen, (3) Ergebnis der Connection-Operation.

Dieses Problem kann mit Erosion gelöst werden. Wie oben erwähnt, reduziert eine Erosion die Fläche der Eingaberegion. Folglich werden die Regionen zerlegt wie gewünscht, wenn der Operator `connection` nach der Erosion angewendet wird. Zuletzt werden die zerlegten Regionen dilatiert, um in etwa die ursprüngliche Form der Regionen zu erhalten.

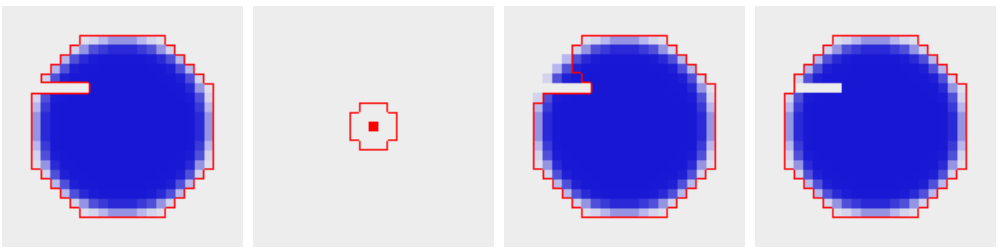


(1) (2) (3)

(1) Segmentierte Regionen nach Anwendung der Erosion, (2) Ergebnis der Connection-Operation, (3) Ergebnis der Dilatation.

Opening und Closing

Beide Operatoren erzeugen die resultierende Region durch Kombination von Dilatation und Erosion. Opening führt eine Erosion, gefolgt von einer Dilatation, aus. Opening ist nützlich, um kleine unerwünschte Strukturen zu entfernen. Closing ist das Gegenteil von Opening, d.h. die Erosion wird nach der Dilatation ausgeführt. Der Closing-Operator ist in der Lage kleine Lücken zu schließen, wie in der folgenden Abbildung zu sehen ist.



(1) (2) (3) (4)

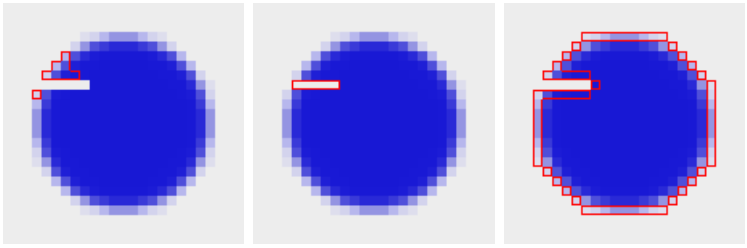
(1) Bild mit Eingaberegion, (2) strukturierendes Element mit Referenzpunkt, (3) Ergebnis der Opening-Operation, (4) Ergebnis der Closing-Operation.

Diese Operatoren können zum Öffnen oder Schließen einer Region verwendet werden:

Morphologische Operatoren		Strukturierendes Element	Referenzpunkt
opening	closing	beliebig	Ursprung
opening_circle	closing_circle	kreisförmig	Ursprung
opening_rectangle1	closing_rectangle1	rechteckig	Ursprung

Weitere Operatoren

Zusätzlich zu den oben genannten Operatoren, kann top_hat benutzt werden, um die Differenz zwischen der Eingaberegion und dem Ergebnis der Opening-Operation zu ermitteln. bottom_hat liefert die Differenz zwischen der Eingaberegion und dem Ergebnis der Closing-Operation. Darüber hinaus kann die Kontur einer Region mit dem Operator boundary ermittelt werden.



(1) (2) (3)

(1) Ergebnis der Top-Hat-Transformation, (2) Ergebnis der Bottom-Hat-Transformation, (3) Kontur der Region.

Der Operator hit_or_miss kann verwendet werden, um Objekte zu finden, wobei der Vordergrund und der Hintergrund des Bildes berücksichtigt werden. Zur Entfernung unerwünschter Zweige aus einem Skelett ist der Operator pruning geeignet.

Glossar

Im Folgenden werden die wichtigsten Begriffe, die im Zusammenhang mit der Morphologie verwendet werden, beschrieben.

Eingaberegion Region, die durch morphologische Operatoren modifiziert wird.

Stukturierendes Element Region, die zum Abtasten der Eingaberegion verwendet wird.

bottom_hat (Region, StructElement : RegionBottomHat : :)

Ermitteln von Lücken zwischen Regionen.

bottom_hat berechnet die Bottom-Hat-Operation, indem zunächst eine Closing-Operation mit StructElement (siehe closing) durchgeführt wird. Von diesem Zwischenergebnis wird die Originalregion abgezogen. Im Gegensatz zu closing, das aus Einzelregionen bei entsprechender Voraussetzung mehrere Regionen zu einer größeren zusammenschließt, bestimmt bottom_hat die Region, welche für diesen Zusammenschluss notwendig ist.

Die Position von StructElement ist ohne Bedeutung, da die Closing-Operation translationsinvariant bzgl. StructElement ist.

Das strukturierende Element (StructElement) kann mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **StructElement** (input_object) ..... region ⇝ *object*
Strukturierendes Element (lageinvariant).

▷ **RegionBottomHat** (output_object) ..... region(-array) ⇝ *object*
Ergebnis des Bottom-Hat-Operators.

Beispiel

```
read_image (Image, 'brake_disk/brake_disk_bike_01')
threshold (Image, Regions, 128, 255)
gen_circle (Circle, 40, 40, 20)
bottom_hat (Regions, Circle, RegionBottomHat)
```

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `bottom_hat` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_region_polygon_filled`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`closing`, `difference`

Siehe auch

`top_hat`, `gray_bothat`, `opening`

Modul

Foundation

boundary (Region : RegionBorder : BoundaryType :)

Reduzieren einer Region auf deren Rand.

`boundary` berechnet für jede Eingaberegion in `Region` die Kontur und gibt diese in `RegionBorder` zurück. Der Parameter `BoundaryType` bestimmt die Art der Randerzeugung.

Die Berechnung der Kontur wird mit Hilfe morphologischer Operationen durchgeführt. Die Ergebnisregionen bestehen nur aus einem minimalen Rand der Eingaberegionen. Dabei hängt die Position des Randes vom Parameter `BoundaryType` ab, für den die folgenden Werte möglich sind:

- *'inner'*: Die Kontur liegt innerhalb der Eingaberegionen.
- *'inner_filled'*: Die Kontur liegt innerhalb der Eingaberegionen, Hohlflächen der Eingaberegionen werden unterdrückt. Aufgrund der Algorithmen-Optimierung kann es leichte Abweichung geben zwischen den so erhaltenen Konturen und den entsprechenden Konturen die mit *'inner_filled'* erhalten wurden.
- *'outer'*: Die Kontur liegt ein Pixel außerhalb der Eingaberegionen.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die auf den Rand reduziert werden.

▷ **RegionBorder** (output_object) ..... region(-array) ⇝ *object*
Ergebnisregionen.

▷ **BoundaryType** (input_control) ..... string ⇝ *string*
Art der Regionenrandes.
Defaultwert: 'inner'
Werteliste: `BoundaryType` ∈ {'inner', 'outer', 'inner_filled'}

Beispiel

```
#include "HalconCpp.h"
using namespace Halcon;

main()
{
  HWindow  w;
  HRegion  circ1 = HRegion::GenCircle (20, 10, 10.5);

  circ1.Display (w);
  w.Click ();

  HRegion marg1 = circ1.Boundary ("inner");
  w.SetColor ("red");
  marg1.Display (w);
  w.Click ();

  return(0);
}
```

Komplexität

Sei A die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(3\sqrt{A}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `boundary` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

dilation_circle, erosion_circle, difference

Siehe auch

fill_up

Modul

Foundation

closing (Region, StructElement : RegionClosing : :)

Schließen von Lücken.

Die Hintereinanderschaltung von Dilatation und Minkowski-Subtraktion definiert die morphologische Grundfunktion closing. closing dient zum Glätten von Regionenrändern. Löcher in Regionen die kleiner als StructElement sind, werden geschlossen. Bei allen closing-Varianten verschmelzen die einzelnen Regionen nicht. closing verwendet intern als Bezugspunkt des strukturierenden Elementes StructElement den Schwerpunkt. Die Position von StructElement ist ohne Bedeutung, da die Closing-Operation translationsinvariant bzgl. StructElement ist.

StructElement kann mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points etc. erzeugt werden.

Achtung

closing wird für jede Region einzeln angewandt. Sollen Lücken zwischen den Regionen geschlossen werden, dann muss vorher ein union1 oder union2 ausgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen auf die die Closing-Operation angewandt wird.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element (lageinvariant).
- ▷ **RegionClosing** (output_object) ..... region(-array) ⇝ *object*
  Regionen mit aufgefüllten Lücken.

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"

main()
{
  cout << "Reproduction of 'closing ()' using " << endl;
  cout << "'dilation()' and 'minkowski_sub1()'" << endl;

  HByteImage img("monkey");
  HWindow    w;

  HRegion      circ = HRegion::GenCircle (10, 10, 1.5);
  HRegionArray regs = (img >= 128).Connection();
```

```
  HRegionArray dilreg = regs.Dilation1 (circ, 1);
  HRegionArray minsub = dilreg.MinkowskiSub1 (circ, 1);

                        img.Display (w);      w.Click ();
  w.SetColor ("red");   regs.Display (w);     w.Click ();
  w.SetColor ("green"); dilreg.Display (w);   w.Click ();
  w.SetColor ("blue");  minsub.Display (w);   w.Click ();

  return(0);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \sqrt{F1} \cdot \sqrt{F2}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `closing` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_region_polygon_filled`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`closing_circle`

Siehe auch

`dilation1`, `erosion1`, `opening`, `minkowski_sub1`

Modul

Foundation

`closing_circle` (Region : RegionClosing : Radius :)

Ausführen eines Closings mit einer Kreismaske.

Das Funktionsverhalten von `closing_circle` ist analog zu `closing` d.h. es werden die Ränder einer Eingaberegion geglättet und Löcher innerhalb einer Region deren Größe kleiner ist als das kreisförmige, strukturierende Element mit dem `Radius`, geschlossen. Die `closing_circle`-Operation ist als die Hintereinanderschaltung von Dilatation und Minkowski-Subtraktion mit einer Kreismaske definiert.

Achtung

`closing_circle` wird für jede Region einzeln angewandt. Sollen Lücken zwischen den Regionen geschlossen werden, dann muss vorher ein `union1` oder `union2` ausgeführt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Regionen auf die der Closing-Operator angewandt wird.
- ▷ **`RegionClosing`** (output_object) ..... region(-array) ⇝ *object*
  Regionen mit aufgefüllten Lücken.
- ▷ **`Radius`** (input_control) ..... real ⇝ *real* / integer
  Radius der Kreismaske.
  Defaultwert: 3.5
  Wertevorschläge: `Radius` ∈ {1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5}
  Typischer Wertebereich: 0.5 ≤ `Radius` ≤ 511.5 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0

Beispiel

```
my_closing_circle(Hobject In, double Radius, Hobject *Out)
{
  Hobject  tmp, StructElement;
  gen_circle(StructElement,100.0,100.0,Radius);
  dilation1(In,StructElement,&tmp,1);
  minkowski_sub1(tmp,StructElement,Out,1);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(4 \cdot \sqrt{F1} \cdot \texttt{Radius}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `closing_circle` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`rank_region`, `fill_up`, `closing`

Siehe auch

`dilation1`, `minkowski_sub1`, `erosion1`, `opening`

Modul

Foundation

closing_rectangle1 (Region : RegionClosing : Width, Height :)

Ausführen eines Closings mit einer Rechtecksmaske.

closing_rectangle1 setzt sich aus der Abfolge der Funktionsaufrufe dilation_rectangle1 und erosion_rectangle1 zusammen. Die Größe des rechteckigen, strukturierenden Elementes wird durch die Parameter Width und Height bestimmt. Wie bei allen closing-Varianten werden die Ränder einer Eingaberegion geglättet und Löcher innerhalb einer Region deren Größe kleiner ist als das rechteckige strukturierende Element geschlossen.

Achtung

closing_rectangle1 wird für jede Region einzeln angewandt. Eine Verschmelzung von Regionen muss explizit mit Routinen wie union1 durchgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen die verarbeitet werden sollen.
- ▷ **RegionClosing** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis des Closing-Operators.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Rechtecks.
  Defaultwert: 10
  Wertevorschläge: Width ∈ {1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200}
  Typischer Wertebereich: 1 ≤ Width ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Rechtecks.
  Defaultwert: 10
  Wertevorschläge: Height ∈ {1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200}
  Typischer Wertebereich: 1 ≤ Height ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und H die Höhe des Rechtecks, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \sqrt{F1} \cdot log_2(H)) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion closing_rectangle1 den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

closing

Siehe auch

dilation_rectangle1, erosion_rectangle1, opening_rectangle1, gen_rectangle1

Modul

Foundation

dilation1 (Region, StructElement : RegionDilation : Iterations :)

Ausdehnen von Regionen.

dilation1 berechnet die Dilatation der Eingaberegionen mit einem strukturierenden Element. Die Anwendung von dilation1 glättet die Ränder der Regionen. Gleichzeitig vergrößert sich die Fläche dieser Regionen. Darüberhinaus kann es vorkommen, dass vorher nicht zusammenhängende Regionen verschmolzen werden. Dennoch bleiben solche Regionen logisch verschiedene Region. Die Dilatation ist eine mengentheoretische Regionenoperation. Sie verwendet die Operation Vereinigung.

Seien M (StructElement) und R (Region) Regionen, wobei M das „strukturierende Element“ und R die zu verarbeitende Region darstellt. Sei weiterhin m ein Punkt aus M, dann wird der Verschiebungsvektor $\vec{v}_m = (dx, dy)$ definiert als die Differenz des Schwerpunktvektors von M mit dem Vektor $\vec{m}$. Die Translation einer Region R um einen Vektor $\vec{v}$ sei mit $t_{\vec{v}_m}(R)$ bezeichnet. Dann ist

$$\texttt{dilation1}(R, M) := \bigcup_{m \in M} t_{-\vec{v}_m}(R)$$

Es wird für jeden Punkt m in M eine Translation mit der Region R durchgeführt. Die Vereinigung über all diese Verschiebungen ist die Dilatation der Region R mit M. dilation1 entspricht dem Operator minkowski_add1 mit dem Unterschied, dass bei dilation1 das strukturierende Element punktgespiegelt wird. Die Position von StructElement ist ohne Bedeutung, da die Verschiebungsvektoren bzgl. des Schwerpunktes bestimmt werden.

Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Elemente ausgeführt werden sollen. Als Eingaberegion für die Iteration n wird die Ergebnisregion der Iteration $(n-1)$ verwendet. Aus der obigen Definition ergibt sich, dass bei einem leeren strukturierenden Element eine leere Region erzeugt wird.

Strukturierende Elemente (StructElement) können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points etc. erzeugt werden.

Achtung

Eine Dilatation führt grundsätzlich zu einer Vergrößerung der Regionen. Eng benachbarte Regionen, die nach Ausführung des Operators zusammenstoßen oder sich überlappen, werden weiterhin als zwei getrennte Regionen behandelt. Um eine Vereinigung zweier Regionen zu erreichen muss zuerst ein union1 durchgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen,die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element.
- ▷ **RegionDilation** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Dilation-Operation.

▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert: 1
Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich: 1 ≤ Iterations (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion dilation1 den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_region_polygon_filled

Nachfolger

reduce_domain, add_channels, select_shape, area_center, connection

Alternativen

minkowski_add1, minkowski_add2, dilation2

Siehe auch

erosion1, erosion2, opening, closing

Modul

Foundation

dilation2 (Region, StructElement : RegionDilation : Row, Column, Iterations :)

Ausdehnen von Regionen (mit Bezugspunkt).

dilation2 berechnet die Dilatation der Eingaberegionen mit einem strukturierenden Element (StructElement) und dem Bezugspunkt, der durch Row und Column charakterisiert wird. dilation2 entspricht dem Operator dilation1 mit dem Unterschied, dass der Bezugspunkt des strukturierenden Elementes frei gewählt werden kann. Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Elemente ausgeführt werden sollen. Als Eingaberegion für die Iteration n wird die Ergebnisregion der Iteration $(n-1)$ verwendet.

Bei Verwendung des leeren strukturierenden Elementes wird eine leere Region erzeugt.

Strukturierende Elemente (StructElement) können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Achtung

Eine Dilatation führt grundsätzlich zu einer Vergrößerung der Regionen. Eng benachbarte Regionen, die nach Ausführung des Operators zusammenstoßen oder sich überlappen werden weiterhin als zwei getrennte Regionen behandelt. Um eine Vereinigung zweier Regionen zu erreichen muss zuerst ein `union1` durchgeführt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Regionen die verarbeitet werden sollen.
- ▷ **`StructElement`** (input_object) ..... region ⇝ *object*
  Strukturierendes Element für Dilation-Operation.
- ▷ **`RegionDilation`** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Dilation-Operation.
- ▷ **`Row`** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunktes.
  Defaultwert: 0
- ▷ **`Column`** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunktes.
  Defaultwert: 0
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert: 1
  Wertevorschläge: `Iterations` ∈ {1, 2, 3, 4, 5, 7, 11, 17, 25, 32, 64, 128}
  Typischer Wertebereich: 1 ≤ Iterations (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation2` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_region_polygon_filled`

Nachfolger

`reduce_domain`, `add_channels`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add1`, `minkowski_add2`, `dilation1`

Siehe auch

erosion1, erosion2, opening, closing

Modul

Foundation

dilation_circle (Region : RegionDilation : Radius :)

Ausführen einer Dilatation mit einer Kreismaske.

dilation_circle führt eine Minkowski-Addition mit einer Kreismaske aus. Da die Kreismaske symmetrisch ist, ist dies identisch mit einer Dilatation. Die Größe des Kreises, der als strukturierendes Element verwendet wird, ist durch Radius angegeben.

Die Wirkung dieser Funktion ist das Vergrößern der Region, die Glättung der Ränder, sowie das Verschließen von Löchern innerhalb der Region, die kleiner sind als die Kreismaske. Sinnvollerweise wird der Radius auf Werte wie 3.5, 5.5 etc. gesetzt, um so Translationen der Region zu vermeiden. Bei ganzzahligen Radien hat der Kreis nämlich keinen ganzzahligen Schwerpunkt, der aber gerundet werden muss.

Achtung

dilation_circle wird für jede Region einzeln angewandt. Sollen Lücken zwischen den Regionen geschlossen werden, dann muss vorher ein union1 oder union2 ausgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen auf die der Dilation-Operator angewandt wird.
- ▷ **RegionDilation** (output_object) ..... region(-array) ⇝ *object*
  Regionen mit aufgefüllten Lücken.
- ▷ **Radius** (input_control) ..... real ⇝ *real* / integer
  Radius der Kreismaske.
  Defaultwert: 3.5
  Wertevorschläge: Radius ∈ {1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5}
  Typischer Wertebereich: 0.5 ≤ Radius ≤ 511.5 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 1.0

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"

main()
{
  cout << "Reproduction of 'dilation_circle ()'" << endl;
  cout << "First = original image " << endl;
  cout << "Blue  = after dilation " << endl;
  cout << "Red   = before dilation " << endl;

  HByteImage img("monkey");
  HWindow    w;

  HRegion      circ   = HRegion::GenCircle (10, 10, 1.5);
  HRegionArray regs   = (img >= 128).Connection();
  HRegionArray minadd = regs.MinkowskiAdd1 (circ, 1);

                        img.Display (w);      w.Click ();
  w.SetColor ("blue");  minadd.Display (w);   w.Click ();
```

```
  w.SetColor ("red");    regs.Display (w);      w.Click ();

  return(0);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \texttt{Radius} \cdot \sqrt{F1}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation_circle` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add1`, `minkowski_add2`, `expand_region`, `dilation1`, `dilation2`, `dilation_rectangle1`

Siehe auch

`gen_circle`, `erosion_circle`, `closing_circle`, `opening_circle`

Modul

Foundation

dilation_rectangle1 (Region : RegionDilation : Width, Height :)

Ausführen einer Dilatation mit einem Rechteck.

`dilation_rectangle1` führt eine Dilatation auf der/den angegebenen Region(en) `Region` durch. Das strukturierende Element für diese Operation bildet ein Rechteck mit der Größe `Width` × `Height`. Die Funktion führt zu einer Expansion der Region und zum Schließen von Löchern, die kleiner als das angegebene Rechteck sind, innerhalb der Regionen.

`dilation_rectangle1` ist eine sehr schnelle Operation, da die Höhe des Rechtecks nur logarithmisch, die Breite gar nicht in die Komplexität eingeht. Dies führt auch bei sehr großen Rechtecken (Kante > 100) zu einem sehr gutem Laufzeitverhalten.

Achtung

`dilation_rectangle1` wird für jede Region einzeln angewandt. Sollen Lücken zwischen den Regionen geschlossen werden, dann muss vorher ein `union1` oder `union2` ausgeführt werden.

Damit eine Region in alle Richtungen gleichmäßig vergrößert wird, müssen `Width` und `Height` ungerade sein. Falls das nicht der Fall ist, wird die Region rechts bzw. unten um ein Pixel weiter dilatiert als nach links bzw. oben.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen die verarbeitet werden sollen.

▷ **RegionDilation** (output_object) ..... region(-array) ⇝ *object*
Ergebnis der Dilation-Operation.

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Rechtecks.
Defaultwert: 11
Wertevorschläge: `Width` ∈ {1, 2, 3, 4, 5, 11, 15, 21, 31, 51, 71, 101, 151, 201}
Typischer Wertebereich: 1 ≤ `Width` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Rechtecks.
Defaultwert: 11
Wertevorschläge: `Height` ∈ {1, 2, 3, 4, 5, 11, 15, 21, 31, 51, 71, 101, 151, 201}
Typischer Wertebereich: 1 ≤ `Height` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"

main()
{
  cout << "Reproduction of 'dilation_rectangle ()'" << endl;
  cout << "First = original image " << endl;
  cout << "Blue  = after dilation " << endl;
  cout << "Red   = after segmentation " << endl;

  HByteImage img("monkey");
  HWindow    w;

  HRegionArray regs   = (img >= 220).Connection();
  HRegionArray dilreg = regs.DilationRectangle1 (2, 4);

                        img.Display (w);      w.Click ();
  w.SetColor ("blue");  dilreg.Display (w);   w.Click ();
  w.SetColor ("red");   regs.Display (w);     w.Click ();

  return(0);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und H die Höhe des Rechtecks, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot ld(H)) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation_rectangle1` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

minkowski_add1, minkowski_add2, expand_region, dilation1, dilation2, dilation_circle

Siehe auch

gen_rectangle1, gen_region_polygon_filled

Modul

Foundation

erosion1 (Region, StructElement : RegionErosion : Iterations :)

Verdünnen von Regionen.

erosion1 berechnet die Erosion der Eingaberegionen mit einem strukturierenden Element. Die Anwendung von erosion1 glättet die Ränder der Regionen. Gleichzeitig verkleinert sich die Fläche dieser Regionen. Darüberhinaus kann es vorkommen, dass vorher zusammenhängende Regionen getrennt werden. Dennoch bleiben solche Regionen logisch eine Region. Die Erosion ist eine mengentheoretische Regionenoperation. Sie verwendet die Operation Durchschnitt.

Seien M (StructElement) und R (Region) Regionen, wobei M das „strukturierende Element“ und R die zu verarbeitende Region darstellt. Sei weiterhin m ein Punkt aus M, dann wird der Verschiebungsvektor $\vec{v}_m = (dx, dy)$ definiert als die Differenz des Schwerpunktvektors von M mit dem Vektor $\vec{m}$. Die Translation einer Region R um einen Vektor $\vec{v}$ sei mit $t_{\vec{v}_m}(R)$ bezeichnet. Dann ist

$$\texttt{erosion1}(R, M) := \bigcap_{m \in M} t_{-\vec{v}_m}(R).$$

Es wird für jeden Punkt aus M eine Translation mit der Region R durchgeführt. Der Durchschnitt über all diesen Verschiebungen ist die Erosion der Region R mit M. erosion1 entspricht dem Operator minkowski_sub1 mit dem Unterschied, dass bei erosion1 das strukturierende Element punktgespiegelt wird. Die Position von StructElement ist ohne Bedeutung, da die Verschiebungsvektoren bzgl. des Schwerpunktes bestimmt werden.

Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet. Bei Verwendung eines leeren strukturierenden Elementes wird die maximale Region erzeugt.

Strukturierende Elemente (StructElement) können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element für die Erosion.
- ▷ **RegionErosion** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Erosion.
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert: 1
  Wertevorschläge: `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich: 1 ≤ Iterations (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `erosion1` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_region_polygon_filled`

Nachfolger

`connection`, `reduce_domain`, `select_shape`, `area_center`

Alternativen

`minkowski_sub1`, `minkowski_sub2`, `erosion2`

Siehe auch

`transpose_region`

Modul

Foundation

```
erosion2 ( Region, StructElement : RegionErosion : Row, Column,
    Iterations : )
```

Verdünnen von Regionen (mit Bezugspunkt).

erosion2 berechnet die Erosion der Eingaberegionen mit einem strukturierenden Element StructElement und einem Bezugspunkt, der durch Row und Column bestimmt ist. Der Operator führt eine Regionentransformation durch. erosion2 entspricht dem Operator erosion1 mit dem Unterschied, dass bei erosion2 der Bezugspunkt frei gewählt werden kann. Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet.

Bei Verwendung des leeren strukturierenden Elementes wird die maximale Region erzeugt.

Strukturierende Elemente (StructElement) können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element für die Erosion
- ▷ **RegionErosion** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Erosion.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunkts.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ Row ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunkts.
  Defaultwert: 0
  Typischer Wertebereich: $0 \leq$ Column ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert: 1
  Wertevorschläge: Iterations $\in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}$
  Typischer Wertebereich: $1 \leq$ Iterations (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion erosion2 den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_region_polygon_filled

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

minkowski_sub2, minkowski_sub1, erosion1

Siehe auch

transpose_region, gen_circle, gen_rectangle2, gen_region_polygon

Modul

Foundation

erosion_circle (Region : RegionErosion : Radius :)

Ausführen einer Erosion mit einer Kreismaske.

erosion_circle führt eine Minkowski-Subtraktion mit einer Kreismaske aus. Die Größe des Kreises, der als strukturierendes Element verwendet wird, ist durch Radius angegeben.

Die erkennbare Wirkung dieser Funktion ist die Glättung der Regionenränder. Zusätzlich werden Regionen, die flächenmäßig kleiner sind als die Kreismaske eliminiert. Sinnvollerweise wird der Radius auf Werte wie 3.5, 5.5 etc. gesetzt, um so Translationen der Region zu vermeiden. Bei ganzzahligen Radien hat der Kreis nämlich keinen ganzzahligen Schwerpunkt, der aber gerundet werden muss.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen auf die der Erosion-Operator angewandt wird.

▷ **RegionErosion** (output_object) ..... region(-array) ⇝ *object*
Verkleinerte Regionen.

▷ **Radius** (input_control) ..... real ⇝ *real* / integer
Radius der Kreismaske.
Defaultwert: 3.5
Wertevorschläge: Radius ∈ {1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5}
Typischer Wertebereich: 0.5 ≤ Radius ≤ 511.5 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 1.0

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"

main()
{
  cout << "Simulation of 'erosion_circle ()'" << endl;
  cout << "First = original image " << endl;
  cout << "Red   = after segmentation " << endl;
  cout << "Blue  = after erosion " << endl;

  HByteImage img("monkey");
```

```
  HWindow    w;

  HRegion      circ   = HRegion::GenCircle (10, 10, 1.5);
  HRegionArray regs   = (img >= 128).Connection();
  HRegionArray minsub = regs.MinkowskiSub1 (circ, 1);

                         img.Display (w);      w.Click ();
  w.SetColor ("red");    regs.Display (w);     w.Click ();
  w.SetColor ("blue");   minsub.Display (w);   w.Click ();

  return(0);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \texttt{Radius} \cdot \sqrt{F1}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `erosion_circle` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`

Nachfolger

`connection`, `reduce_domain`, `select_shape`, `area_center`

Alternativen

`minkowski_sub1`

Siehe auch

`gen_circle`, `dilation_circle`, `closing_circle`, `opening_circle`

Modul

Foundation

erosion_rectangle1 (Region : RegionErosion : Width, Height :)

Ausführen einer Erosion mit einem Rechteck.

`erosion_rectangle1` führt eine Erosion auf der/den angegebenen Region(en) `Region` durch. Das strukturierende Element für diese Operation bildet ein Rechteck mit den Größenparametern `Width` und `Height`. Die Funktion führt zu einer Verkleinerung der Region.

`erosion_rectangle1` ist eine sehr schnelle Operation, da die Höhe des Rechtecks nur logarithmisch, die Breite gar nicht in die Komplexität eingeht. Dies führt auch bei sehr großen Rechtecken (Kante > 100) zu einem sehr gutem Laufzeitverhalten.

Regionen, die schmale Verbindungsstege zwischen größeren Flächen besitzen werden zwar optisch getrennt, jedoch bleibt die Region logisch als eine Region erhalten.

Achtung

Damit eine Region in alle Richtungen gleichmäßig verkleinert wird, müssen `Width` und `Height` ungerade sein. Falls das nicht der Fall ist, wird die Region rechts bzw. unten um ein Pixel weiter dilatiert als nach bzw. oben.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Regionen die verarbeitet werden sollen.
- ▷ **`RegionErosion`** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Erosions-Operation.
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Breite des Rechtecks.
  Defaultwert: 11
  Wertevorschläge: `Width` ∈ {1, 2, 3, 4, 5, 11, 15, 21, 31, 51, 71, 101, 151, 201}
  Typischer Wertebereich: 1 ≤ `Width` ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Rechtecks.
  Defaultwert: 11
  Wertevorschläge: `Height` ∈ {1, 2, 3, 4, 5, 11, 15, 21, 31, 51, 71, 101, 151, 201}
  Typischer Wertebereich: 1 ≤ `Height` ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und H die Höhe des Rechtecks, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot ld(H)) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `erosion_rectangle1` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`erosion1`, `minkowski_sub1`

Siehe auch

`gen_rectangle1`

Modul

Foundation

hit_or_miss (Region, StructElement1, StructElement2 : RegionHitMiss : Row, Column :)

Ausführen der Hit-or-Miss-Operation für Regionen.

hit_or_miss berechnet die Hit-or-Miss-Transformation. Der Operator führt dabei zuerst eine Erosion der Region Region mit dem strukturierenden Element StructElement1 durch. Im zweiten Schritt wird eine Erosion der komplementären Region mit dem strukturierenden Element StructElement2 durchgeführt. Aus den beiden Zwischenergebnissen wird die Schnittmenge gebildet.

Die Hit-or-Miss-Transformation wählt genau die Punkte aus, deren Umgebung die Bedingungen der strukturierenden Elemente StructElement1 und StructElement2 erfüllen. StructElement1 bezeichnet dabei die Punkte des Vordergrundes, StructElement2 die Punkte des Hintergrundes. Für ein sinnvolles Ergebnis müssen StructElement1 und StructElement2 wie Schlüssel und Schloss zusammenpassen. StructElement1 und StructElement2 sind in jedem Fall disjunkt. Row und Column bezeichnen Zeilen- und Spaltennummer des Bezugspunktes.

Strukturierende Elemente (StructElement1, StructElement2) können mit Operatoren wie gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement1** (input_object) ..... region ⇝ *object*
  Erosionsmaske für die Eingaberegionen.
- ▷ **StructElement2** (input_object) ..... region ⇝ *object*
  Erosionsmaske für die Komplemente der Eingaberegionen.
- ▷ **RegionHitMiss** (output_object) ..... region(-array) ⇝ *object*
  Das Ergebnis der Hit-or-Miss-Operation.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunktes.
  Defaultwert: 16
  Wertevorschläge: Row ∈ {0, 16, 32, 128, 256}
  Typischer Wertebereich: 0 ≤ Row ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunktes.
  Defaultwert: 16
  Wertevorschläge: Column ∈ {0, 16, 32, 128, 256}
  Typischer Wertebereich: 0 ≤ Column ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, $F1$ die Fläche des strukturierenden Elementes 1 und $F2$ die Fläche der des strukturierenden Elementes 2, dann ist die Laufzeitkomplexität für ein Objekt:

$$O\left(\sqrt{F} \cdot \left(\sqrt{F1} + \sqrt{F2}\right)\right) \quad .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion hit_or_miss den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Nachfolger

difference, reduce_domain, select_shape, area_center, connection

Alternativen

erosion2, dilation2

Siehe auch

gen_region_points, gen_region_polygon_filled

Modul

Foundation

minkowski_add1 (Region, StructElement : RegionMinkAdd : Iterations :)

Ausdehnen von Regionen.

minkowski_add1 berechnet die Minkowski-Addition der Eingaberegionen mit einem strukturierenden Element. Die Anwendung von minkowski_add1 glättet die Ränder der Regionen. Gleichzeitig vergrößert sich die Fläche dieser Regionen. Darüberhinaus kann es vorkommen, dass vorher nicht zusammenhängende Regionen verschmolzen werden. Dennoch bleiben solche Regionen logisch verschiedene Region. Die Minkowski-Addition ist eine mengentheoretische Regionenoperation. Sie basiert auf Translation und Vereinigung.

Seien M StructElement und R (Region) Regionen, wobei M das „strukturierende Element“ und R die zu verarbeitende Region darstellt. Sei weiterhin m ein Punkt aus M, dann wird der Verschiebungsvektor $\vec{v}_m = (dx, dy)$ definiert als die Differenz des Schwerpunktvektors von M mit dem Vektor $\vec{m}$. Die Translation einer Region R um einen Vektor $\vec{v}$ sei mit $t_{\vec{v}_m}(R)$ bezeichnet. Dann ist

$$\texttt{minkowski_add1}(R, M) := \bigcup_{m \in M} t_{\vec{v}_m}(R)$$

Es wird für jeden Punkt m in M eine Translation mit der Region R durchgeführt. Die Vereinigung über all diese Verschiebungen ist die Dilatation der Region R mit M. Der Unterschied zwischen minkowski_add1 und dilation1 ist die Punktspiegelung des strukturierenden Elements (Negation des Verschiebungsvektors) bei dilation1. Die Position von StructElement ist ohne Bedeutung, da die Verschiebungsvektoren bzgl. des Schwerpunktes bestimmt werden.

Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet. Aus der obigen Definition ergibt sich, dass bei einem leeren strukturierenden Element eine leere Region erzeugt wird.

Strukturierende Elemente (StructElement) können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points etc. erzeugt werden.

Achtung

Eine Minkowski-Addition führt grundsätzlich zu einer Vergrößerung der Regionen. Eng benachbarte Regionen, die nach Ausführung des Operators zusammenstoßen oder sich überlappen werden weiterhin als zwei getrennte Regionen behandelt. Um eine Vereinigung zweier Regionen zu erreichen muss zuerst ein union1 durchgeführt werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen die verarbeitet werden sollen.

▷ **StructElement** (input_object) ..... region ⇝ *object*
Strukturierendes Element.

▷ **RegionMinkAdd** (output_object) ..... region(-array) ⇝ *object*
Ergebnis der Minkowski-Addition.

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert: 1
Wertevorschläge: `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich: 1 ≤ Iterations (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `minkowski_add1` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_region_polygon_filled`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add2`, `dilation1`

Siehe auch

`transpose_region`, `minkowski_sub1`

Modul

Foundation

minkowski_add2 (Region, StructElement : RegionMinkAdd : Row, Column, Iterations :)

Ausdehnen von Regionen (mit Bezugspunkt).

minkowski_add2 berechnet die Minkowski-Addition der Eingaberegionen mit einem strukturierenden Element (StructElement) und dem Bezugspunkt, der durch Row und Column charakterisiert wird. minkowski_add2 entspricht dem Operator minkowski_add1 mit dem Unterschied, dass der Bezugspunkt des strukturierenden Elementes frei gewählt werden kann. Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet.

Bei Verwendung des leeren strukturierenden Elementes wird ein leere Region erzeugt.

Strukturierende Elemente können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points etc. erzeugt werden.

Achtung

Eine Minkowski-Addition führt grundsätzlich zu einer Vergrößerung der Regionen. Eng benachbarte Regionen, die nach Ausführung des Operators zusammenstoßen oder sich überlappen werden weiterhin als zwei getrennte Regionen behandelt. Um eine Vereinigung zweier Regionen zu erreichen muss zuerst ein union1 durchgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element.
- ▷ **RegionMinkAdd** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Minkowski-Addition.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunktes.
  Typischer Wertebereich: $1 \leq$ Row ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunktes.
  Typischer Wertebereich: $1 \leq$ Column ≤ 511 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert: 1
  Wertevorschläge: Iterations $\in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}$
  Typischer Wertebereich: $1 \leq$ Iterations (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion minkowski_add2 den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_region_polygon_filled

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

minkowski_add1, dilation1

Siehe auch

transpose_region

Modul

Foundation

minkowski_sub1 (Region, StructElement : RegionMinkSub : Iterations :)

Verdünnen von Regionen.

minkowski_sub1 berechnet die Minkowski-Subtraktion der Eingaberegionen mit einem Strukturierenden Element. Die Anwendung von minkowski_sub1 glättet die Ränder der Regionen. Gleichzeitig verkleinert sich die Fläche dieser Regionen. Darüberhinaus kann es vorkommen, dass vorher zusammenhängende Regionen getrennt werden. Dennoch bleiben solche Regionen logisch eine Region. Die Minkowski-Subtraktion ist eine mengentheoretische Regionenoperation. Sie verwendet die Operation Durchschnitt.

Seien M (StructElement) und R (Region) Regionen, wobei M das „strukturierende Element“ und R die zu verarbeitende Region darstellt. Sei weiterhin m ein Punkt aus M, dann wird der Verschiebungsvektor $\vec{v}_m = (dx, dy)$ definiert als die Differenz des Schwerpunktvektors von M mit dem Vektor $\vec{m}$. Die Translation einer Region R um einen Vektor $\vec{v}$ sei mit $t_{\vec{v}_m}(R)$ bezeichnet. Dann ist

$$\texttt{minkowski_sub1}(R, M) := \bigcap_{m \in M} t_{\vec{v}_m}(R)$$

Es wird für jeden Punkt aus M eine Translation mit der Region R durchgeführt. Der Durchschnitt über all diesen Verschiebungen ist die Minkowski-Differenz der Region R mit M. Der Unterschied zwischen minkowski_sub1 und erosion1 ist die Punktspiegelung des strukturierenden Elemente (Negation des Verschiebungsvektors) bei erosion1. Die Position von StructElement ist ohne Bedeutung, da die Verschiebungsvektoren bzgl. des Schwerpunktes bestimmt werden.

Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet. Bei Verwendung eines leeren strukturierenden Elementes wird die maximale Region erzeugt.

Strukturierende Elemente (StructElement) können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element.
- ▷ **RegionMinkSub** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Minkowski-Subtraktion.

▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert: 1
Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich: 1 ≤ Iterations (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion minkowski_sub1 den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_region_polygon_filled

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

minkowski_sub2, erosion1

Siehe auch

transpose_region

Modul

Foundation

minkowski_sub2 (Region, StructElement : RegionMinkSub : Row, Column, Iterations :)

Verdünnen von Regionen (mit Bezugspunkt).

minkowski_sub2 berechnet die Minkowski-Subtraktion der Eingaberegionen mit einem Strukturierenden Element StructElement und dem Bezugspunkt in Row und Column. minkowski_sub2 entspricht dem Operator minkowski_sub1 mit dem Unterschied, dass bei minkowski_sub2 der Bezugspunkt frei gewählt werden kann. Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet.

Bei Verwendung des leeren strukturierenden Elementes wird die maximale Region erzeugt.

Strukturierende Elemente (StructElement) können mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen die verarbeitet werden sollen.

▷ **StructElement** (input_object) ..... region ⇝ *object*
Strukturierendes Element.

▷ **RegionMinkSub** (output_object) ..... region(-array) ⇝ *object*
Ergebnis der Erosion.

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeile des Bezugspunkts.
Defaultwert: 0
Wertevorschläge: Row ∈ {0, 10, 16, 32, 64, 100, 128}
Typischer Wertebereich: 0 ≤ Row ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spalte des Bezugspunkts.
Defaultwert: 0
Wertevorschläge: Column ∈ {0, 10, 16, 32, 64, 100, 128}
Typischer Wertebereich: 0 ≤ Column ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert: 1
Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich: 1 ≤ Iterations (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion minkowski_sub2 den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, watersheds, class_ndim_norm, gen_circle, gen_ellipse,

gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_region_polygon_filled

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

minkowski_sub1, erosion1, erosion2

Siehe auch

gen_circle, gen_rectangle2, gen_region_polygon

Modul

Foundation

opening (Region, StructElement : RegionOpening : :)

Auftrennen von Lücken.

Die opening-Operation ist als die Hintereinanderschaltung von Erosion und Minkowski-Addition definiert. Durch Anwendung von opening bleiben größere Strukturen weitgehend erhalten, kleine Regionen wie Linien und Punkte sowie feine Strukturen werden gelöscht. Im Gegensatz dazu bewirkt eine closing-Operation, dass kleine Unterbrechungen, Lücken und Risse erhalten bleiben bzw. ausgefüllt werden (siehe hierzu closing).

opening dient zur Elimination von kleinen Regionen (kleiner als StructElement) und zum Glätten der Ränder. opening verwendet intern als Bezugspunkt den Schwerpunkt des strukturierenden Elementes. Die Position von StructElement ist aber ohne Bedeutung, da Opening translationsinvariant gegenüber dem strukturierende Element ist.

StructElement kann mit Operatoren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element (lageinvariant).
- ▷ **RegionOpening** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis des Opening-Operators.

Beispiel

```
* Large regions in an aerial picture (beech trees or meadows):
read_image(Image,'forest_road')
threshold(Image,Light,160,255)
gen_circle(StructElement,100,100,10)
* selecting the large regions
opening(Light,StructElement,Large)
* Selecting of edges with certain orientation:
read_image(Image,'fabrik')
sobel_amp(Image,Sobel,'sum_abs',3)
threshold(Sobel,Edges,10,255)
gen_rectangle2(StructElement,100,100,3.07819,20,1)
opening(Edges,StructElement,Direction)
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \sqrt{F1} \cdot \sqrt{F2}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion opening den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  set_system('no_object_result',<RegionResult>)
- leere Region:
  set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_region_polygon_filled

Nachfolger

reduce_domain, select_shape, area_center, connection

Alternativen

minkowski_add1, erosion1, opening_circle

Siehe auch

gen_circle, gen_rectangle2, gen_region_polygon

Modul

Foundation

opening_circle (Region : RegionOpening : Radius :)

Auftrennen von Lücken mit einer Kreismaske.

opening_circle ist als die Hintereinanderschaltung von Erosion und Minkowski-Addition mit einer Kreismaske definiert (siehe Beispiel). opening_circle dient zur Elimination von kleinen Regionen (kleiner als die Kreismaske) und zum Glätten der Ränder.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **RegionOpening** (output_object) ..... region(-array) ⇝ *object*
Ergebnis des Opening-Operators.

▷ **Radius** (input_control) ..... real ⇝ *real* / integer
Radius der Kreismaske.
Defaultwert: 3.5
Wertevorschläge: Radius ∈ {1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5}
Typischer Wertebereich: $0.5 \leq$ Radius ≤ 511.5 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 1.0

Beispiel

```
* Large regions in an aerial picture (beech trees or meadows):
read_image (Image, 'forest_road')
threshold (Image, Region, 120, 255)
* Close the small gaps.
closing_circle (Region, RegionClosing, 3.5)
* Select the large regions.
opening_circle (RegionClosing, RegionOpening, 19.5)
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(4 \cdot \sqrt{F1} \cdot \texttt{Radius}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `opening_circle` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`opening`, `dilation1`, `minkowski_add1`, `gen_circle`

Siehe auch

`transpose_region`

Modul

Foundation

opening_rectangle1 (Region : RegionOpening : Width, Height :)

Ausführen eines Openings mit einem Rechteck.

`opening_rectangle1` setzt sich aus der Abfolge der Funktionsaufrufe `erosion_rectangle1` und `dilation_rectangle1` zusammen. Die Größe des rechteckigen, strukturierenden Elementes wird durch die Parameter `Width` und `Height` bestimmt. Wie bei allen `opening`-Varianten bleiben größere Strukturen erhalten, während kleine Regionen wie Linien und Punkte sowie feine Strukturen gelöscht werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen die verarbeitet werden sollen.

▷ **RegionOpening** (output_object) ..... region(-array) ⇝ *object*
Ergebnis des Opening-Operators.

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Rechtecks.
Defaultwert: 10
Wertevorschläge: `Width` ∈ {1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200}
Typischer Wertebereich: 1 ≤ `Width` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Rechtecks.
Defaultwert: 10
Wertevorschläge: `Height` ∈ {1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200}
Typischer Wertebereich: 1 ≤ `Height` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und H die Höhe des Rechtecks, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \sqrt{F1} \cdot ld(H)) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `opening_rectangle1` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`opening`, `gen_rectangle1`, `dilation_rectangle1`, `erosion_rectangle1`

Modul

Foundation

pruning (Region : RegionPrune : Length :)

Beschneiden von Ästen einer Region.

`pruning` entfernt aus einem Skelett (`Region`) Äste einer vorgegebenen Länge, die durch `Length` vorgegeben ist. Die übrigen Äste bleiben unverändert.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die verarbeitet werden sollen.

▷ **RegionPrune** (output_object) ..... region(-array) ⇝ *object*
Das Ergebnis des Pruning-Operators.

▷ **Length** (input_control) ..... integer ⇝ *integer*
Länge der Äste die entfernt werden.
Defaultwert: 2
Wertevorschläge: `Length` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich: 1 ≤ `Length` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\texttt{Length} \cdot 3 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `pruning` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`skeleton`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Siehe auch

`junctions_skeleton`

Modul

Foundation

top_hat (Region, StructElement : RegionTopHat : :)

Ermitteln von Ausbuchtungen von Regionen.

`top_hat` berechnet das `opening` von `Region` mit `StructElement`. Von der Eingaberegion wird dieses Zwischenergebnis abgezogen. Im Gegensatz zu `opening`, das Einzelregionen bei entsprechender Voraussetzung in mehrere Regionen auftrennt, bestimmt `top_hat` die Region, welche bei dieser Trennung entfernt wird.

Die Position von `StructElement` ist ohne Bedeutung, da `opening` translationsinvariant bzgl. `StructElement` ist.

Das strukturierende Element (`StructElement`) kann mit Operatoren wie `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`, `draw_region`, `gen_region_polygon`, `gen_region_points`, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *object*
  Strukturierendes Element (lageinvariant).
- ▷ **RegionTopHat** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis des Top-Hat-Operators.

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `top_hat` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_region_polygon_filled`

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`opening`, `difference`

Siehe auch

`bottom_hat`, `gray_tophat`, `opening`

Modul

Foundation

Kapitel 20

OCR

20.1 Convolutional Neural Networks

clear_ocr_class_cnn (: : OCRHandle :)

Löschen eines OCR-Klassifikators.

clear_ocr_class_cnn löscht den durch OCRHandle angegebenen OCR-Klassifikator, der mit read_ocr_class_cnn gelesen wurde, und gibt sämtlichen für den Klassifikator benötigten Speicher frei. Der Klassifikator kann nach dem Aufruf von clear_ocr_class_cnn nicht mehr verwendet werden. Das Handle OCRHandle ist nach dem Aufruf ungültig.

Parameter

▷ **OCRHandle** (input_control) ..... ocr_cnn(-array) ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Falls OCRHandle korrekt ist, dann liefert clear_ocr_class_cnn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

do_ocr_single_class_cnn, do_ocr_multi_class_cnn, do_ocr_word_cnn

Siehe auch

read_ocr_class_cnn

Modul

OCR/OCV

deserialize_ocr_class_cnn (: : SerializedItemHandle : OCRHandle)

Deserialisieren eines serialisierten CNN-basierten OCR-Klassifikators.

deserialize_ocr_class_cnn deserialisiert einen CNN-basierten OCR-Klassifikator, welcher mit serialize_ocr_class_cnn serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Der serialisierte OCR-Klassifikator wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten OCR-Klassifikator mit dem Handle OCRHandle gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **OCRHandle** (output_control) ..... ocr_cnn ⇝ *handle*
  Handle des OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_ocr_class_cnn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_ocr_class_cnn

Nachfolger

do_ocr_single_class_cnn, do_ocr_multi_class_cnn

Modul

OCR/OCV

do_ocr_multi_class_cnn (Character, Image : : OCRHandle : Class, Confidence)

Klassifikation mehrerer Zeichen mit einem CNN-basiertem OCR-Klassifikator.

do_ocr_multi_class_cnn berechnet zu den durch die Regionen Character und die Grauwerte Image gegebenen Zeichen mit dem durch OCRHandle gegebenen OCR-Klassifikator die jeweils beste Klasse und gibt die Klassen in Class und die zugehörigen Konfidenzen (Wahrscheinlichkeiten) der Klassen in Confidence zurück.

Im Gegensatz zu do_ocr_single_class_cnn kann do_ocr_multi_class_cnn also mehrere Zeichen gleichzeitig klassifizieren und ist damit typischerweise schneller als eine Schleife, die do_ocr_single_class_cnn zur Klassifikation einzelner Zeichen verwendet. Allerdings kann do_ocr_multi_class_cnn nur die beste Klasse des jeweiligen Zeichens zurückliefern. Da sich die Konfidenzen als Wahrscheinlichkeiten interpretieren lassen, und damit auch leicht überprüfbar ist, ob ein Zeichen zu unsicher klassifiziert worden ist, ist dies im Normalfall kein Nachteil, außer die Klassen überlappen sich so stark, dass in vielen Fällen noch die zweitbeste Klasse untersucht werden muss, um sicher zu entscheiden, welche Klasse vorliegt. In diesem Fall sollte do_ocr_single_class_cnn verwendet werden.

Eine Zeichenkette des Zahlenwertes '$\backslash 032$' (alternativ als '$\backslash 0x1A$' dargestellt) im Resultat Class bedeutet, dass die Region als Rückweisungsklasse klassifiziert wurde.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte der Zeichen.
- ▷ **OCRHandle** (input_control) ..... ocr_cnn ⇝ *handle*
  Handle des OCR-Klassifikators.

▷ **Class** (output_control) . . . . . string(-array) ⇝ *string*
Ergebnis der Klassifikation der Zeichen durch das CNN.
Parameteranzahl: Class == Character

▷ **Confidence** (output_control) . . . . . real(-array) ⇝ *real*
Konfidenz der Klasse der Zeichen.
Parameteranzahl: Confidence == Character

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `do_ocr_multi_class_cnn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`read_ocr_class_cnn`

Alternativen

`do_ocr_word_cnn`, `do_ocr_single_class_cnn`

Modul

OCR/OCV

do_ocr_single_class_cnn (Character, Image : : OCRHandle, Num : Class, Confidence)

Klassifikation eines Zeichens mit einem CNN-basiertem OCR-Klassifikator.

`do_ocr_single_class_cnn` berechnet zu dem durch die Region `Character` und die Grauwerte `Image` gegebenen Zeichen mit dem durch `OCRHandle` gegebenen OCR-Klassifikator die `Num` besten Klassen und gibt die Klassen in `Class` und die zugehörigen Konfidenzen (Wahrscheinlichkeiten) der Klassen in `Confidence` zurück. Da potentiell mehrere Klassen zurückgeliefert werden können, darf `Character` nur eine Region (ein Zeichen) enthalten. Falls mehrere Zeichen in einem Aufruf klassifiziert werden sollen, muss `do_ocr_multi_class_cnn` verwendet werden.

Da `do_ocr_multi_class_cnn` bei der Klassifikation mehrerer Zeichen typischerweise schneller ist als eine Schleife mit `do_ocr_single_class_cnn` und da sich die Konfidenzen als Wahrscheinlichkeiten interpretieren lassen, und damit auch leicht überprüfbar ist, ob ein Zeichen zu unsicher klassifiziert worden ist, empfiehlt sich in den meisten Fällen die Verwendung von `do_ocr_multi_class_cnn`, außer die zweitbeste Klasse soll explizit untersucht werden.

Eine Zeichenkette des Zahlenwertes '$\backslash 032$' (alternativ als '$\backslash 0x1A$' dargestellt) im Resultat `Class` bedeutet, dass die Region als Rückweisungsklasse klassifiziert wurde.

Parameter

▷ **Character** (input_object) . . . . . region ⇝ *object*
Zu erkennendes Zeichen.

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Grauwerte des Zeichens.

▷ **OCRHandle** (input_control) . . . . . ocr_cnn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **Num** (input_control) . . . . . integer-array ⇝ *integer*
Anzahl der maximal zu bestimmenden besten Klassen.
Defaultwert: 1
Wertevorschläge: Num ∈ {1, 2, 3, 4, 5}

▷ **Class** (output_control) . . . . . string(-array) ⇝ *string*
Ergebnis der Klassifikation des Zeichens durch das CNN.

▷ **Confidence** (output_control) . . . . . real(-array) ⇝ *real*
Konfidenz(en) der Klasse(n).

Beispiel

```
read_image(Image, 'bottle2')
OffsetRow := 100
OffsetCol := 108
read_ocr_class_cnn('Universal_0-9_NoRej.occ', OCRHandle)
* Select each digit and use do_ocr_single_class_cnn to apply OCR
gen_rectangle1(ROI_Date, OffsetRow, OffsetCol, OffsetRow, OffsetCol)
for I := 1 to 6 by 1
  Offset := I % 2 * 10
  smallest_rectangle1(ROI_Date, Row1, Col1, Row2, Col2)
  gen_rectangle1(ROI_Date, OffsetRow, Offset + Col2, OffsetRow + 42, \
  Col2 + Offset + 31)
  do_ocr_single_class_cnn(ROI_Date, Image, OCRHandle, 1, Class, Confidence)
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert do_ocr_single_class_cnn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_ocr_class_cnn

Alternativen

do_ocr_multi_class_cnn

Modul

OCR/OCV

do_ocr_word_cnn (Character, Image : : OCRHandle, Expression, NumAlternatives, NumCorrections : Class, Confidence, Word, Score)

Klassifikation einer Zeichengruppe mit einem CNN-basiertem OCR-Klassifikator.

do_ocr_word_cnn verhält sich wie do_ocr_multi_class_cnn, indem es zu den durch die Regionen Character und die Grauwerte Image gegebenen Zeichen mit dem durch OCRHandle gegebenen OCR-Klassifikator die jeweils beste Klasse berechnet, und die Klassen in Class und die zugehörigen Konfidenzen (Wahrscheinlichkeiten) der Klassen in Confidence zurückgibt.

Im Gegensatz zu do_ocr_multi_class_cnn behandelt do_ocr_word_cnn die Zeichengruppe als eine Einheit, die durch Aneinanderhängen von Klassennamen für jedes einzelne Zeichen ein Wort Word liefert. Dies ermöglicht die Einschränkung der erlaubten Klassifikationsergebnisse auf Textebene durch Angabe eines Ausdrucks Expression, der das erwartete Wort beschreibt.

Ein Ausdruck kann das Wort auf ein festes Lexikon, welches zuvor durch create_lexicon oder import_lexicon erzeugt wurde, einschränken, indem der Name des Lexikons in spitzen Klammern wie in *'<meinlexikon>'* angegeben wird. Ist der Ausdruck nicht von dieser Form, so wird er als regulärer Ausdruck mit derselben Syntax wie für tuple_regexp_match interpretiert. Bei Verwendung von Quantifikatoren wie '*' sollte man normalerweise mit einem Ausdruck der Form *'^...'$'* dafür sorgen, dass das gesamte Wort verwendet wird. Im Gegensatz zu tuple_regexp_match erlaubt do_ocr_word_cnn **keine** Angabe von zusätzlichen Optionen für Expression in einem Tupel.

Falls das aus den jeweils besten Klassen abgeleitete Wort dem Ausdruck nicht genügt, versucht do_ocr_word_cnn eine Korrektur durch Berücksichtigung der NumAlternatives besten Klassen für jedes Zeichen, so wie sie von do_ocr_single_class_cnn für ein einzelnes Zeichen bestimmt werden. Hierzu

werden alle möglichen Korrekturen ausprobiert, bei denen die Klassifikation von höchstens NumCorrections Zeichenregionen geändert wird. Zu beachten ist hierbei, dass NumAlternatives und NumCorrections Auswirkungen auf die Komplexität des Algorithmus haben und daher in manchen Fällen intern beschränkt werden müssen. Für weitere Informationen siehe den Abschnitt 'Komplexität' weiter unten.

Falls der Ausdruck ein Lexikon beschreibt und die obige Prozedur nicht zum Erfolg führte, so wird das ähnlichste Wort aus dem Lexikon zurückgegeben, solange für die Korrektur weniger als NumCorrections Editieroperationen benötigt werden (siehe suggest_lexicon).

Das Ergebnis wird mit einem Score zwischen 0.0 (keine Korrektur gefunden) und 1.0 (ursprüngliches Wort war schon korrekt) bewertet. Verringert wird der Score durch das Anbringen einer Strafe entsprechend der Anzahl korrigierter Zeichen und einer (kleineren) Strafe für das Ignorieren von Klassen mit höherer Konfidenz um einen Ausdruck Expression zu erhalten:

$$\texttt{Score} = 1.0 - \texttt{PenaltyCorrections} - \texttt{PenaltyAlternatives}$$

$$\begin{aligned} \texttt{PenaltyCorrections} &= \alpha * \texttt{num_corr} \\ \texttt{PenaltyAlternatives} &= \alpha * \beta * \texttt{num_alt} \end{aligned}$$

wobei num_corr die Anzahl tatsächlich angebrachter Korrekturen und num_alt die Anzahl ignorierter Alternativen ist.

$$\begin{aligned} \alpha &= \frac{1}{\texttt{NumCorrections} + 1} \\ \beta &= \frac{1}{\texttt{NumCorrections} * (\texttt{NumAlternatives} - 1) + 2} \end{aligned}$$

Diese Bewertung ist rein kombinatorisch und berücksichtigt **nicht** die ursprünglichen Konfidenzen der besten Klassen.

Eine Zeichenkette des Zahlenwertes '\032' (alternativ als '\0x1A' dargestellt) im Resultat Class bedeutet, dass die Region als Rückweisungsklasse klassifiziert wurde.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte der Zeichen.
- ▷ **OCRHandle** (input_control) ..... ocr_cnn ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Expression** (input_control) ..... string ⇝ *string*
  Ausdruck, der die erlaubte Wortstruktur beschreibt
- ▷ **NumAlternatives** (input_control) ..... integer ⇝ *integer*
  Anzahl der Klassen pro Zeichen, die für die interne Wortkorrektur berücksichtigt werden.
  Defaultwert: 3
  Wertevorschläge: NumAlternatives ∈ {3, 4, 5}
  Typischer Wertebereich: 1 ≤ NumAlternatives
- ▷ **NumCorrections** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl der korrigierten Zeichen.
  Defaultwert: 2
  Wertevorschläge: NumCorrections ∈ {1, 2, 3, 4, 5}
  Typischer Wertebereich: 0 ≤ NumCorrections
- ▷ **Class** (output_control) ..... string(-array) ⇝ *string*
  Ergebnis der Klassifikation der Zeichen durch das MLP.
  Parameteranzahl: Class == Character
- ▷ **Confidence** (output_control) ..... real(-array) ⇝ *real*
  Konfidenz der Klasse der Zeichen.
  Parameteranzahl: Confidence == Character

▷ **Word** (output_control) . . . . . string ⇝ *string*
Text des Wortes nach Klassifikation und Korrektur.

▷ **Score** (output_control) . . . . . real ⇝ *real*
Ähnlichkeitsmaß von korrigiertem Wort zum Original.

Komplexität

Die Komplexität des Prüfens aller möglichen Korrekturen hat die Größenordnung $O((n * a)^{min(c,n)})$, wobei a die Anzahl der Alternativen, n die Anzahl der Zeichenregionen und c die Anzahl der erlaubten Korrekturen ist. Als Absicherung gegen quasi-unendliche Schleifen im Falle von großem n wird der Wert von c intern beschränkt auf 5, 3, oder 1, falls $a * n >= 30$, 60, bzw. 90.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `do_ocr_word_cnn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_ocr_class_cnn`

Alternativen

`do_ocr_multi_class_cnn`

Modul

OCR/OCV

get_params_ocr_class_cnn (: : OCRHandle, GenParamName : GenParamValue)

Auslesen der Parameter eine CNN-basierten OCR-Klassifikators.

`get_params_ocr_class_cnn` gibt die Parameter `GenParamName` des durch `OCRHandle` angegebenen OCR-Klassifikators in `GenParamValue` aus. Dies ist insbesondere nützlich, wenn der Klassifikator mit `read_ocr_class_cnn` eingelesen wurde. Die Ausgabedaten von `get_params_ocr_class_cnn` können z.B. verwendet werden, um zu überprüfen, ob ein zu lesendes Zeichen in dem Klassifikator vorhanden ist.

Mögliche Werte für `GenParamName` sind:

'characters': Liefert eine Liste vortrainierter Symbole, welche mit dem CNN- basierten OCR-Klassifikator gelesen werden können. Falls der Eingabeparameter *'characters'* spezifiziert wird, ist kein weiterer Parameter zulässig. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_cnn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Namen von Parametern für die Steuerung des Verhaltens des verwendeten CNNs.
Defaultwert: 'characters'
Werteliste: `GenParamName` ∈ {'characters'}

▷ **GenParamValue** (output_control) . . . . . string(-array) ⇝ *integer* / string
Zu den generischen Parametern gehörende Werte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_params_ocr_class_cnn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_ocr_class_cnn

Nachfolger

do_ocr_single_class_cnn, do_ocr_multi_class_cnn

Modul

OCR/OCV

query_params_ocr_class_cnn (: : OCRHandle : GenParamName)

Liefert für einen gegebenen CNN-basierten OCR-Klassifikator die generischen Parameternamen, die im Operator get_params_ocr_class_cnn verwendet werden können.

Mit Hilfe des Operators query_params_ocr_class_cnn lassen sich für einen gegebenen CNN-basierten OCR Klassifikator alle möglichen generischen Parameternamen abfragen. Die resultierenden Parameternamen können anschließend als Eingabeparameter für den Operator get_params_ocr_class_cnn verwendet werden.

Parameter

▷ **OCRHandle** (input_control) ..... ocr_cnn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **GenParamName** (output_control) ..... string(-array) ⇝ *string*
Namen der generischen Parameter.

Ergebnis

Ist der eingegebene Handle des OCR-Klassifikators valide, dann liefert query_params_ocr_class_cnn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

get_params_ocr_class_cnn

Modul

OCR/OCV

read_ocr_class_cnn (: : FileName : OCRHandle)

Lesen eines CNN-basierten OCR-Klassifikators aus Datei.

read_ocr_class_cnn liest einen abgespeicherten OCR-Klassifikator ein, der auf einem Convolutional Neural Network (CNN) basiert.

Da das Trainieren eines OCR-Klassifikators relativ lange dauern kann, wird der Klassifikator typischerweise in einem Offline-Prozess trainiert und gespeichert. Im Online-Prozess wird der Klassifikator mit read_ocr_class_cnn gelesen und danach zur Klassifikation mit do_ocr_single_class_cnn, do_ocr_multi_class_cnn oder do_ocr_word_cnn verwendet.

HALCON stellt eine Anzahl vortrainierter OCR-Klassifikatoren zur Verfügung (siehe „Solution Guide I", Kapitel 'OCR', Abschnitt 'Pretrained OCR Fonts'). Diese vortrainierten OCR-Klassifikatoren ermöglichen es, eine

Vielzahl an Zeichensätzen zu lesen, ohne vorher einen OCR-Klassifikator trainieren zu müssen. Das Training der vortrainierten OCR-Klassifikatoren erfolgte mit Zeichen, die dunkel auf hellem Hintergrund gedruckt waren.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Defaultwert: 'Universal_Rej.occ'
Wertevorschläge: FileName ∈ {'Universal_NoRej.occ', 'Universal_Rej.occ', 'Universal_0-9_NoRej.occ', 'Universal_0-9_Rej.occ', 'Universal_0-9+_NoRej.occ', 'Universal_0-9+_Rej.occ', 'Universal_0-9A-Z_NoRej.occ', 'Universal_0-9A-Z_Rej.occ', 'Universal_0-9A-Z+_NoRej.occ', 'Universal_0-9A-Z+_Rej.occ', 'Universal_A-Z+_NoRej.occ', 'Universal_A-Z+_Rej.occ'}
Dateiendung: .occ, .fnt

▷ **OCRHandle** (output_control) . . . . . ocr_cnn ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_ocr_class_cnn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

do_ocr_single_class_cnn, do_ocr_multi_class_cnn, do_ocr_word_cnn

Siehe auch

get_params_ocr_class_cnn

Modul

OCR/OCV

serialize_ocr_class_cnn (: : OCRHandle : SerializedItemHandle)

Serialisieren eines CNN-basierten OCR-Klassifikators.

serialize_ocr_class_cnn serialisiert einen CNN-basierten OCR-Klassifikator (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von serialize_ocr_class_cnn in eine Datei geschrieben werden. Der OCR-Klassifikator wird in dem Handle OCRHandle übergeben. Der serialisierte OCR-Klassifikator wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_ocr_class_cnn wieder deserialisiert werden.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_cnn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_ocr_class_cnn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`clear_ocr_class_cnn`, `fwrite_serialized_item`, `send_serialized_item`, `deserialize_ocr_class_cnn`

Modul

OCR/OCV

20.2 Deep OCR

`apply_deep_ocr` (`Image` : : `DeepOcrHandle`, `Mode` : `DeepOcrResult`)

Anwenden eines Deep OCR-Modells auf mehreren Bildern.

`apply_deep_ocr` wendet das durch `DeepOcrHandle` gegebene Deep OCR-Modell auf die Eingabebilder im Tupel `Image` an. Der Operator gibt `DeepOcrResult` zurück, ein Tupel mit einem Ergebnis-Dictionary `DeepOcrResult` pro Eingabebild.

Der Operator `apply_deep_ocr` stellt folgende Anforderungen an die Eingabebilder `Image`:

- Bildtyp: byte.
- Anzahl Kanäle: 1 oder 3.

Des Weiteren wird der Operator `apply_deep_ocr` die Bilder `Image` vorverarbeiten um den Modellspezifikationen zu entsprechen. Dies bedeutet, dass die Byte-Bilder normalisiert und in real-Bilder umgewandelt werden. Außerdem wird für `Mode` = *'auto'* oder *'detection'* das Bild `Image` auf die Eingabedimensionen des Modells aufgefüllt und im Falle von nur einem Kanal in ein dreikanaliges Bild umgewandelt. Für `Mode` = *'recognition'* werden dreikanalige Bilder automatisch in einkanalige Bilder umgewandelt.

Der Parameter `Mode` bestimmt den Modus und dadurch, welche Komponente ausgeführt wird. Es werden folgende Werte unterstützt:

'auto' **(A):** Es werden beide Teile ausgeführt, die Lokalisierung und Erkennung der Worte.

'detection' **(DET):** Nur die Komponente zur Lokalisierung wird ausgeführt. Daher lokalisiert das Modell lediglich die Wortbereiche innerhalb des Bildes.

'recognition' **(REC):** Nur die Komponente zum Erkennen der Worte wird ausgeführt. Daher erfordert das Modell, dass das Bild nur einen engen Zuschnitt eines einzelnen Wortes enthält.

Es gilt zu beachten, dass das Modell die entsprechenden Komponenten enthalten muss, siehe `create_deep_ocr`.

Das Ergebnis-Dictionary `DeepOcrResult` hat folgende Einträge, abhängig von `Mode` (mit der Abkürzung markiert):

`image` (A, DET, REC): Vorverarbeitetes Bild.

`score_maps` (A, DET): Bewertungen als Bild mit vier Kanälen:

- Zeichenbewertung: Bewertung der Lokalisierung von Zeichen.
- Verknüpfungsbewertung: Bewertung der Verbindung von lokalisierten Zeichenmittelpunkten zu einem zusammenhängenden Wort.
- Ausrichtung 1: Sinuskomponente der vorhergesagten Wortausrichtung.
- Ausrichtung 2: Kosinuskomponente der vorhergesagten Wortausrichtung.

`words` (A, DET): Dictionary mit den folgenden Einträgen. Dabei sind die Einträge Tupel mit einem Wert für jedes gefundene Wort.

- `word` (A): Erkanntes Wort.

- `char_candidates` (A): Ein Dictionary mit Informationen über jeden Buchstaben eines jeden erkannten Wortes. Das Dictionary enthält für jedes Wort ein Schlüssel/Wert Paar: Der Index des Wortes ist der Schlüssel und ein Tupel von Dictionaries sind die Werte. Jedes dieser Buchstaben-Dictonaries enthält folgende Schlüssel/Werte Paare:
  - `'candidate'`: Ein Tupel der besten *'recognition_num_char_candidates'* Kandidaten.
  - `'confidence'`: Softmax basierte Konfidenzwerte eines jeden besten Kandidaten. Man beachte, dass diese Werte nicht kalibriert sind und mit Vorsicht verwendet werden sollten. Sie können sich signifikant von Modell zu Modell unterscheiden.
- `word_image` (A): Vorverarbeiteter Bildteil, der das Wort enthält.
- `row` (A, DET): Lokalisiertes Wort: Mittelpunkt, Zeilenkoordinate
- `col` (A, DET): Lokalisiertes Wort: Mittelpunkt, Spaltenkoordinate
- `phi` (A, DET): Lokalisiertes Wort: Winkel phi.
- `length1` (A, DET): Lokalisiertes Wort: Halbe Kantenlänge 1.
- `length2` (A, DET): Lokalisiertes Wort: Halbe Kantenlänge 2.
- `line_index` (A, DET): Zeilen-Index der lokalisierten Worte, falls *'detection_sort_by_line'* gleich *'true'*.

Die Lokalisierung der Worte ist mittels Parameter orientierter Rechtecke gegeben, siehe `gen_rectangle2` für weitere Informationen.

`word_boxes_on_image` (A, DET): Dictionary mit der Lokalisierung der Worte im Koordinatensystem der vorverarbeiteten Bilder, die sich in `image` befinden. Dabei sind die Einträge Tupel mit einem Wert für jedes gefundene Wort.

- `row` (A, DET): Lokalisiertes Wort: Mittelpunkt, Zeilenkoordinate
- `col` (A, DET): Lokalisiertes Wort: Mittelpunkt, Spaltenkoordinate
- `phi` (A, DET): Lokalisiertes Wort: Winkel phi.
- `length1` (A, DET): Lokalisiertes Wort: Halbe Kantenlänge 1.
- `length2` (A, DET): Lokalisiertes Wort: Halbe Kantenlänge 2.

Die Lokalisierung der Worte ist mittels Parameter orientierter Rechtecke gegeben, siehe `gen_rectangle2` für weitere Informationen.

`word_boxes_on_score_maps` (A, DET): Dictionary mit der Lokalisierung der Worte im Koordinatensystem der Bewertungsbilder, die sich in `score_maps` befinden. Die Einträge sind die gleichen wie für `word_boxes_on_image` oben.

`word` (REC): Erkanntes Wort.

`char_candidates` (REC): Ein Tupel von Dictionaries mit Informationen über jeden Buchstaben des erkannten Wortes. Jedes Buchstaben-Dictonary enthält folgende Schlüssel/Werte Paare:

- `'candidate'`: Ein Tupel der besten *'recognition_num_char_candidates'* Kandidaten.
- `'confidence'`: Softmax basierte Konfidenzwerte eines jeden besten Kandidaten. Man beachte, dass diese Werte nicht kalibriert sind und mit Vorsicht verwendet werden sollten. Sie können sich signifikant von Modell zu Modell unterscheiden.

Achtung

Systemvoraussetzungen: Um diesen Operator auf der GPU auszuführen (siehe `get_deep_ocr_param`), werden cuDNN und cuBLAS benötigt. Für weitere Details wird auf den „Installation Guide“, Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods“, verwiesen. Alternativ kann dieser Operator auch auf der CPU ausgeführt werden.

Parameter

▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *object* : byte
Eingabebild.

▷ **`DeepOcrHandle`** (input_control) ..... deep_ocr ⇝ *handle*
Handle des Deep OCR-Modells.

▷ **`Mode`** (input_control) ..... string ⇝ *string*
Inferenz-Modus.
Defaultwert: []
Werteliste: `Mode` ∈ {'auto', 'recognition', 'detection'}

▷ **DeepOcrResult** (output_control) . . . . . dict(-array) ⇝ *handle*
Tupel mit Ergebnis-Dictionaries.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `apply_deep_ocr` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`get_deep_ocr_param`, `set_deep_ocr_param`, `create_deep_ocr`

Modul

OCR/OCV

create_deep_ocr (: : GenParamName, GenParamValue : DeepOcrHandle)

Erstellen eines Deep OCR-Modells.

`create_deep_ocr` erstellt aus vorab trainierten Komponenten ein Deep OCR-Modell und gibt sein Handle in `DeepOcrHandle` zurück. Das Handle beinhaltet alle zur Inferenz erforderlichen Parameter.

Ein Deep OCR-Modell besteht normalerweise aus zwei Komponenten:

- `'detection_model'`: Lokalisiert die Worte (zusammenhängende Zeichen) im Bild.
- `'recognition_model'`: Erkennt das Wort innerhalb des gefundenen Bildteils.

Mit `GenParamName` und entsprechendem `GenParamValue` sind folgende Optionen setzbar:

'mode': Bestimmt den Modus und dadurch, welche Komponente ausgeführt wird. Es werden folgende Werte unterstützt:

'detection': Nur die Komponente zur Lokalisierung wird ausgeführt. Daher lokalisiert das Modell lediglich die Wortbereiche innerhalb des Bildes.

'recognition': Nur die Komponente zum Erkennen der Worte wird ausgeführt. Daher erfordert das Modell, dass das Bild nur einen engen Zuschnitt eines einzelnen Wortes enthält.

'auto': Es werden beide Teile ausgeführt, die Lokalisierung und Erkennung der Worte.

Default: 'auto'.

Der Zeichensatz, den das Modell über seine Erkennungskomponente erkennt, kann mittels `get_deep_ocr_param` (mit *'recognition_alphabet'*) abgerufen werden.

Parameter

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'mode'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Wert des generischen Parameters.
Defaultwert: []
Werteliste: `GenParamValue` ∈ {'auto', 'recognition', 'detection'}

▷ **DeepOcrHandle** (output_control) . . . . . deep_ocr ⇝ *handle*
Handle des Deep OCR-Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_deep_ocr` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung geworfen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`write_deep_ocr`, `apply_deep_ocr`, `get_deep_ocr_param`, `set_deep_ocr_param`

Modul

OCR/OCV

get_deep_ocr_param (: : DeepOcrHandle,
GenParamName : GenParamValue)

Auslesen der Parameter des Deep OCR-Modells.

`get_deep_ocr_param` gibt die Parameterwerte von `GenParamName` für das Deep OCR-Modell `DeepOcrHandle` in `GenParamValue` aus.

Die Parameter können das ganze Modell betreffen oder nur spezifische Komponenten. Die folgende Tabelle gibt einen Überblick, welche Parameter gesetzt und welche zurückgegeben werden können sowie für welche Modell-Komponente sie zutreffen.

GenParamName	Modell	Det. Komp.	Recog. Komp.
'device'	s	s	s
'detection_device'	–	s g	–
'detection_image_dimensions'	–	g	–
'detection_image_height'	–	s g	–
'detection_image_size'	–	s g	–
'detection_image_width'	–	s g	–
'detection_min_character_score'	–	s g	–
'detection_min_link_score'	–	s g	–
'detection_min_word_area'	–	s g	–
'detection_min_word_score'	–	s g	–
'detection_model'	–	g	–
'detection_orientation'	–	s g	–
'detection_sort_by_line'	–	s g	–
'detection_tiling'	–	s g	–
'recognition_alphabet'	–	–	g
'recognition_batch_size'	–	–	s g
'recognition_device'	–	–	s g
'recognition_image_dimensions'	–	–	g
'recognition_image_height'	–	–	g
'recognition_image_width'	–	–	s g
'recognition_model'	–	–	g
'recognition_num_char_candidates'	–	–	s g

Hierbei stehen die verwendeten Symbole für folgendes:

- 's': Der Parameter kann mit `set_deep_ocr_param` gesetzt werden.
- 'g': Der Parameter kann mit `get_deep_ocr_param` zurückerhalten werden.
- '-': Der Parameter hat keine Anwendung.
- Modell: Komplettes Deep OCR-Modell mit all seinen Komponenten.
- Det. comp.: Komponente zur Lokalisierung.
- Recog. comp.: Komponente zur Erkennung.

Im Folgenden werden die Parameter `GenParamName`, deren Werte mit `get_deep_ocr_param` abgefragt werden können, aufgelistet und erläutert.

Komplettes Modell:

'device':
Handle der Hardware Einheit, auf der das Modell ausgeführt wird. Ein Tupel mit den Handles aller potentiell Deep OCR-fähigen Geräte kann über `query_available_dl_devices` erzeugt werden. Es gilt zu beachten, dass eine Einheit auch für eine einzelne Komponente neu gesetzt werden kann. In diesem Fall wird nur der möglicherweise verbliebene Teil des Modells (z.B. eine verbleibende Komponente) auf der Einheit dieses Handles ausgeführt.
Default: Handle der Standard-Einheit, d.h. die GPU mit Index *0* in der Liste, die mit `get_system` und *'cuda_devices'* erhalten wird. Falls keine Einheit vorhanden ist, ein leeres Tupel.

Lokalisierungs-Komponente:

'detection_device':
Dieser Parameter setzt die Hardware Einheit auf welcher die Lokalisierungs-Komponente des Deep OCR-Modells ausgeführt wird. Siehe *'device'* für weitere Erklärungen.
Default: Der selbe Wert wie für *'device'*.

'detection_image_dimensions':
Tupel mit den Bilddimensionen (*'detection_image_width'*, *'detection_image_height'*, Anzahl Kanäle), welche die Lokalisierungs-Komponente verarbeitet.
Default: [*1024*, *1024*, *3*]

'detection_image_height':
Höhe des Bildes, welche die Lokalisierungs-Komponente verarbeitet. Dies bedeutet, dass das Netzwerk erst das Eingabebild auf diese Höhe zoomt bevor es das Bild verarbeitet. Dadurch kann diese Größe die Ergebnisse beeinflussen.
Die Modellarchitektur verlangt, dass die Höhe ein Vielfaches von 32 ist. Ist dies nicht der Fall, wird die Höhe auf das nächstgrößere ganzzahlige Vielfache von 32 gerundet.
Typical values: *768*, *1024*, *1280*
Default: 1024

'detection_image_size':
Tupel mit der Bildgröße (*'detection_image_width'*, *'detection_image_height'*), welche die Lokalisierungs-Komponente verarbeitet.
Default: [*1024*, *1024*]

'detection_image_width':
Breite des Bildes, welche die Lokalisierungs-Komponente verarbeitet. Dies bedeutet, dass das Netzwerk erst das Eingabebild auf diese Breite zoomt bevor es das Bild verarbeitet. Dadurch kann diese Größe die Ergebnisse beeinflussen.
Die Modellarchitektur verlangt, dass die Breite ein Vielfaches von 32 ist. Ist dies nicht der Fall, wird die Breite auf das nächstgrößere ganzzahlige Vielfache von 32 gerundet.
Typical values: *768*, *1024*, *1280*
Default: 1024

'detection_min_character_score':
Der Parameter *'detection_min_character_score'* gibt den unteren Schwellenwert an, der für die Zeichenbewertung verwendet wird, um die Abmessungen der Zeichen zu schätzen. Durch Anpassen von *'detection_min_character_score'* können vorgeschlagene Instanzen aufgespalten, beziehungsweise benachbarte Instanzen vereint werden.

Wertebereich: $\in [0, 1]$.
Default: 0.5

'detection_min_link_score':
Der Parameter *'detection_min_link_score'* definiert die minimale Verknüpfungsbewertung, die zwischen zwei lokalisierten Zeichen gefordert ist, um diese Zeichen als zusammenhängendes Wort zu erkennen.
Wertebereich: $\in [0, 1]$.
Default: 0.3

'detection_min_word_area':
Der Parameter *'detection_min_word_area'* definiert die Mindestgröße, die ein lokalisiertes Wort haben muss, um vorgeschlagen zu werden. Über diesen Parameter können zu kleine Vorschläge gefiltert werden.
Wertebereich: ≥ 0.
Default: 10

'detection_min_word_score':
Der Parameter *'detection_min_word_score'* definiert die minimale Bewertung, die eine lokalisierte Instanz enthalten muss, um als Wort vorgeschlagen zu werden. Über diesen Parameter können unsichere Worte herausgefiltert werden.
Wertebereich: $\in [0, 1]$.
Default: 0.7

'detection_model':
Handle der Deep OCR-Modell-Komponente zur Lokalisierung der Worte.

'detection_orientation':
Dieser Parameter erlaubt es, die Orientierung der vorhergesagten Worte im Voraus festzulegen. Diese Festlegung kann wieder rückgängig gemacht werden, indem *'detection_orientation'* auf *'auto'* gesetzt wird. Dann wird für die Orientierung der Worte wieder die Vorhersage des Modells benutzt.
Wertebereich: $(-\pi, \pi]$.
Default: 'auto'

'detection_sort_by_line':
Auf Grundlage der geschätzten Orientierungen der lokalisierten Worte werden die gefundenen Worte zeilenweise sortiert. Wird eine Sortierung in Zeilen- und Spaltenrichtung bevorzugt, muss der Parameter *'detection_sort_by_line'* auf *'false'* gesetzt werden.
Default: 'true'

'detection_tiling':
Das Eingabebild wird automatisch in überlappende Kacheln der Größe *'detection_image_size'* aufgeteilt, welche separat von der Lokalisierungs-Komponente verarbeitet werden. Dies ermöglicht das Verarbeiten von Eingabebildern, welche wesentlich größer sind als die eigentliche Bildgröße *'detection_image_size'*, ohne dass das Eingabebild gezoomt werden muss. Entsprechend wird für *'detection_tiling'* = *'true'* das Eingabebild nicht mehr gezoomt bevor es verarbeitet wird.
Default: *'false'*

Erkennungs-Komponente:

'recognition_alphabet':
Der Zeichensatz mit welchem das Deep OCR-Modell trainiert wurde und somit die Zeichen, welche das Modell erkennen kann.

'recognition_batch_size':
Anzahl der Bilder in einem Batch, die gleichzeitig in den Gerätespeicher transferiert werden. Für mehr Informationen wird auf die Referenzdokumentation des Operator `apply_dl_model` bezüglich des Parameters *'batch_size'* verwiesen. Dieser Parameter kann dazu verwendet werden die Laufzeit von `apply_deep_ocr` auf der gegebenen Hardware-Einheit zu verkürzen. Falls die Erkennungs-Komponente mehrere Eingaben (Wörter) gleichzeitig verarbeiten muss, kann eine höhere *'recognition_batch_size'* zu schnellerer Verarbeitung führen. Es gilt zu beachten, dass ein höherer Wert für die *'recognition_batch_size'* zu höherem Speicherverbrauch auf der Hardware-Einheit führt.
Default: 1

'recognition_device':
Dieser Parameter setzt die Hardware Einheit auf welcher die Erkennungs-Komponente des Deep OCR-Modells ausgeführt wird. Siehe *'device'* für weitere Erklärungen.
Default: Der selbe Wert wie für *'device'*.

'recognition_image_dimensions':
Tupel mit den Bilddimensionen (*'recognition_image_width'*, *'recognition_image_height'*, Anzahl Kanäle), welche die Erkennungs-Komponente verarbeitet.
Default: [*120*, *32*, *1*]

'recognition_image_height':
Höhe des Bildes, welche die Erkennungs-Komponente verarbeitet. Dies bedeutet, dass das Netzwerk erst den Bildausschnitt auf diese Höhe zoomt bevor es den Bildausschnitt verarbeitet.
Default: 32

'recognition_image_width':
Breite des Bildes, welche die Erkennungs-Komponente verarbeitet. Dies bedeutet, dass das Netzwerk erst den Bildausschnitt auf diese Breite zoomt bevor es den Bildausschnitt verarbeitet.
Default: 120

'recognition_model':
Handle der Deep OCR-Modell-Komponente zur Worterkennung.

'recognition_num_char_candidates':
Kontrolliert die Anzahl der zurückgegebenen Buchstaben-Kandidaten des Operators `apply_deep_ocr`.
Default: *3*

Parameter

▷ **`DeepOcrHandle`** (input_control) . . . . . deep_ocr ⇝ *handle*
Handle des Deep OCR-Modells.

▷ **`GenParamName`** (input_control) . . . . . attribute.name ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'recognition_model'
Werteliste: `GenParamName` ∈ {'detection_device', 'detection_image_dimensions', 'detection_image_height', 'detection_image_size', 'detection_image_width', 'detection_min_character_score', 'detection_min_link_score', 'detection_min_word_area', 'detection_min_word_score', 'detection_model', 'detection_orientation', 'detection_sort_by_line', 'detection_tiling', 'recognition_alphabet', 'recognition_batch_size', 'recognition_device', 'recognition_image_dimensions', 'recognition_image_height', 'recognition_image_width', 'recognition_model', 'recognition_num_char_candidates'}

▷ **`GenParamValue`** (output_control) . . . . . attribute.value(-array) ⇝ *integer* / handle / string / real
Wert des generischen Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_deep_ocr_param` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_deep_ocr`, `set_deep_ocr_param`

Nachfolger

`set_deep_ocr_param`, `apply_deep_ocr`

Siehe auch

`set_deep_ocr_param`

Modul

OCR/OCV

`read_deep_ocr` (: : FileName : DeepOcrHandle)

Lesen eines Deep OCR-Modells aus einer Datei.

Der Operator read_deep_ocr liest ein Deep OCR-Modell. Diese Modelle müssen im HALCON Format vorliegen. Als Ergebnis wird das Handle DeepOcrHandle zurückgegeben.

Das Deep OCR-Modell wird aus der Datei FileName geladen. Die Datei wird hierbei sowohl im Verzeichnis $HALCONROOT/dl/, als auch im aktuell genutzten Verzeichnis gesucht. Die HALCON Standard-Dateiendung eines Deep OCR-Modells ist '.hdo'.

Zu beachten ist, dass die Werte laufzeitspezifischer Parameter nicht in der Datei abgespeichert werden, siehe write_deep_ocr. Folglich werden diese Parameter beim Einlesen des Modells mit ihrem Standardwert (siehe get_deep_ocr_param) initialisiert.

Parameter

▷ **FileName** (input_control) ..... filename.read ⇝ *string*
Dateiname
Dateiendung: .hdo

▷ **DeepOcrHandle** (output_control) ..... deep_ocr ⇝ *handle*
Handle des Deep OCR-Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_deep_ocr den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

create_deep_ocr

Nachfolger

set_deep_ocr_param, get_deep_ocr_param, apply_deep_ocr

Alternativen

create_deep_ocr

Modul

OCR/OCV

set_deep_ocr_param (: : DeepOcrHandle, GenParamName, GenParamValue :)

Setzen der Parameter des Deep OCR-Modells.

set_deep_ocr_param setzt die Parameter GenParamName des Deep OCR-Modells DeepOcrHandle auf die Werte in GenParamValue.

Die möglichen Parameter sind in get_deep_ocr_param gelistet und erklärt.

Achtung

Systemvoraussetzungen: Um *'gpu'* Parameter zu setzen, d.h. um den Parameter GenParamName *'device'* auf eine GPU zu setzen, sind cuDNN und cuBLAS notwendig. Für weitere Details siehe *„Installation Guide“*, Abschnitt „Requirements for Deep Learning and Deep-Learning-Based Methods“.

Parameter

▷ **DeepOcrHandle** (input_control) ..... deep_ocr ⇝ *handle*
Handle des Deep OCR-Modells.

▷ **GenParamName** (input_control) . . . . . attribute.name ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'detection_image_width'
Werteliste: GenParamName ∈ {'device', 'detection_device', 'detection_image_height', 'detection_image_size', 'detection_image_width', 'detection_min_character_score', 'detection_min_link_score', 'detection_min_word_area', 'detection_min_word_score', 'detection_orientation', 'detection_sort_by_line', 'detection_tiling', 'recognition_batch_size', 'recognition_device', 'recognition_image_width', 'recognition_num_char_candidates'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *real* / string / integer / handle
Wert des generischen Parameters.
Defaultwert: 1024
Wertevorschläge: GenParamValue ∈ {0, 10, 50, 100, 200, 512, 768, 1024, 1280, 0.3, 0.4, 0.5, 0.6, 0.7, 'true', 'false', 'auto'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_deep_ocr_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_deep_ocr, get_deep_ocr_param

Nachfolger

get_deep_ocr_param, apply_deep_ocr

Siehe auch

get_deep_ocr_param

Modul

OCR/OCV

write_deep_ocr (: : DeepOcrHandle, FileName :)

Abspeichern eines Deep OCR-Modells in eine Datei.

write_deep_ocr speichert das Deep OCR-Modell DeepOcrHandle in der Datei FileName ab. Zu beachten ist, dass die Werte der laufzeitspezifischen Parameter *'gpu'*, *'runtime'* und *'runtime_init'* nicht abgespeichert werden.

Die HALCON Standard-Dateiendung eines Deep OCR-Modells ist '.hdo'.

Das Deep OCR-Modell kann mit read_deep_ocr eingelesen werden.

Parameter

▷ **DeepOcrHandle** (input_control) . . . . . deep_ocr ⇝ *handle*
Handle des Deep OCR-Modells.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Dateiname
Dateiendung: .hdo

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_deep_ocr den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_deep_ocr, set_deep_ocr_param

Nachfolger

clear_handle

Modul

OCR/OCV

20.3 K-Nächste-Nachbarn

clear_ocr_class_knn (: : OCRHandle :)

Löschen eines OCR-Klassifikators.

clear_ocr_class_knn löscht den durch OCRHandle angegebenen OCR-Klassifikator, der mit create_ocr_class_knn oder select_feature_set_trainf_knn erzeugt wurde, und gibt sämtlichen für den Klassifikator benötigten Speicher frei. Der Klassifikator kann nach dem Aufruf von clear_ocr_class_knn nicht mehr verwendet werden. Das Handle OCRHandle ist nach dem Aufruf ungültig.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Falls OCRHandle korrekt ist, dann liefert clear_ocr_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

trainf_ocr_class_knn, read_ocr_class_knn

Siehe auch

create_class_knn

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

OCR/OCV

create_ocr_class_knn (: : WidthCharacter, HeightCharacter, Interpolation, Features, Characters, GenParamName, GenParamValue : OCRHandle)

Erzeugen eines OCR-Klassifikators unter Verwendung eines k-Nearest-Neighbor-Klassifikators.

create_ocr_class_knn erzeugt einen OCR-Klassifikator unter Verwendung eines k-Nearest-Neighbor-Klassifikators (k-NN). Das Handle des OCR-Klassifikators wird in OCRHandle zurückgegeben.

Eine Beschreibung des k-NN Klassifikators kann bei create_class_knn nachgelesen werden.

Die für die OCR zu verwendenden Merkmale werden in Features angegeben. Die Merkmale sind unten beschrieben. Die Anzahl der Klassen, die im k-NN verwendet werden, wird aus den für die OCR zu verwendenden Zeichen, die in Characters übergeben werden, festgelegt.

Die zur Klassifikation zu verwendenden Merkmale werden mit Features angegeben. Dabei kann Features ein Tupel mit mehreren Merkmalsnamen beinhalten. Jeder dieser Merkmalsnamen führt zur Berechnung von einem oder mehreren Merkmalen für den Klassifikator. Einige der Merkmalsnamen erzeugen Grauwertmerkmale (z.B.*'pixel_invar'*). Da ein Klassifikator immer eine konstante Anzahl von Merkmalen (Eingabevariablen) benötigt, wird ein zu klassifizierendes Zeichen auf eine Standardgröße transformiert, die durch WidthCharacter und HeightCharacter festgelegt wird. Die bei der Transformation anzuwendende Interpolation wird mit Interpolation festgelegt. Sie hat dieselbe Bedeutung wie bei affine_trans_image. Die Interpolation sollte so gewählt werden, dass bei der Transformation keine Aliasing-Effekte auftreten. Im Normalfall sollte Interpolation = *'constant'* verwendet werden. Bei der Wahl der Größe des transformierten Zeichens ist zu beachten, dass die Größe nicht zu groß gewählt wird, da die Generalisierungseigenschaften des Klassifikators sonst sehr schlecht werden können. Insbesondere können bei großen Größen kleine Segmentierungsfehler einen großen Einfluss auf die berechneten Merkmale haben, wenn Grauwertmerkmale verwendet werden, da sich durch die Segmentierungsfehler das umschließende Rechteck des Zeichens ändern kann, was dazu führt, dass das Zeichen anders als trainiert skaliert wird. Im Normalfall sollten Größen zwischen 6×8 und 10×14 verwendet werden.

Mit Features können die folgenden Merkmalsnamen zur Erkennung der Zeichen ausgewählt werden.

'default' *'ratio'* und *'pixel_invar'* werden ausgewählt.

'pixel' Grauwerte des Zeichens (WidthCharacter × HeightCharacter Merkmale).

'pixel_invar' Grauwerte des Zeichens mit automatischer maximaler Grauwertspreizung (WidthCharacter × HeightCharacter Merkmale).

'pixel_binary' Region des Zeichens als Binärbild auf eine Größe von WidthCharacter × HeightCharacter skaliert. (WidthCharacter × HeightCharacter Merkmale).

'gradient_8dir' Die Gradienten des Zeichens werden berechnet. Die Gradientenrichtungen werden in 8 Richtungen diskretisiert. Das Amplitudenbild wird entsprechend der diskretisierten Richtungen in 8 Kanäle aufgeteilt. In jedem der 8 Kanäle werden auf einem 5×5 Gitter Stichproben entnommen. Diese Stichproben bilden die Merkmale (200 Merkmale).

'projection_horizontal' Horizontale Projektion der Grauwerte (siehe gray_projections, HeightCharacter Merkmale).

'projection_horizontal_invar' Horizontale Projektion der Grauwerte, die maximal gespreizt sind (HeightCharacter Merkmale).

'projection_vertical' Vertikale Projektion der Grauwerte (siehe gray_projections, WidthCharacter Merkmale).

'projection_vertical_invar' Vertikale Projektion der Grauwerte, die maximal gespreizt sind (WidthCharacter Merkmale).

'ratio' Seitenverhältnis des Zeichens (siehe height_width_ratio, 1 Merkmal).

'anisometry' Formmerkmal Anisometrie (siehe eccentricity, 1 Merkmal).

'width' Breite des Zeichens vor der Normierung auf die Standardgröße (nicht skalierungsinvariant, siehe height_width_ratio, 1 Merkmal).

'height' Höhe des Zeichens vor der Normierung auf die Standardgröße (nicht skalierungsinvariant, siehe height_width_ratio, 1 Merkmal).

'zoom_factor' Größenunterschied zwischen aktuellem Zeichen und den Werten von WidthCharacter und HeightCharacter (nicht skalierungsinvariant, 1 Merkmal).

'foreground' Relativer Anteil der Vordergrund-Pixel (1 Merkmal).

'foreground_grid_9' Relativer Anteil der Vordergrund-Pixel in einem 3×3 Raster innerhalb des umschließenden Rechtecks des Zeichens (9 Merkmale).

'foreground_grid_16' Relativer Anteil der Vordergrund-Pixel in einem 4×4 Raster innerhalb des umschließenden Rechtecks des Zeichens (16 Merkmale).

'compactness' Formmerkmal Kompaktheit (siehe `compactness`, 1 Merkmal).

'convexity' Formmerkmal Konvexität (siehe `convexity`, 1 Merkmal).

'moments_region_2nd_invar' Normierte zweite geometrische Momente der Region (siehe `moments_region_2nd_invar`, 3 Merkmale).

'moments_region_2nd_rel_invar' Normierte zweite relative geometrische Momente der Region (siehe `moments_region_2nd_rel_invar`, 2 Merkmale).

'moments_region_3rd_invar' Normierte dritte geometrische Momente der Region (siehe `moments_region_3rd_invar`, 4 Merkmale).

'moments_central' Normierte zentrale geometrische Momente der Region (siehe `moments_region_central`, 4 Merkmale).

'moments_gray_plane' Normierte Grauwertmomente und die Winkel der Grauwertebene (siehe `moments_gray_plane`, 4 Merkmale)

'phi' Sinus und Kosinus der Orientierung (Winkel) des Zeichens (siehe `elliptic_axis`, 2 Merkmal).

'num_connect' Anzahl der Zusammenhangskomponenten (siehe `connect_and_holes`, 1 Merkmal).

'num_holes' Anzahl der Hohlflächen (siehe `connect_and_holes`, 1 Merkmal).

'cooc' Werte der binären Cooccurrence-Matrix (siehe `gen_cooc_matrix`, 8 Merkmale).

'num_runs' Anzahl der Sehnen der Region normiert auf die Höhe (1 Merkmal).

'chord_histo' Häufigkeit der Sehnen pro Zeile (nicht skalierungsinvariant, `HeightCharacter` Merkmale).

Nachdem der Klassifikator erzeugt wurde, wird er mit `trainf_ocr_class_knn` trainiert. Daraufhin kann der Klassifikator mit `write_ocr_class_knn` abgespeichert werden. Alternativ können auch sofort nach dem Training Zeichen mit `do_ocr_single_class_knn` oder `do_ocr_multi_class_knn` klassifiziert werden.

Ein Vergleich zwischen k-NN und Support-Vektor-Maschinen (SVM) (siehe `create_ocr_class_svm`) zeigt typischerweise, dass k-NNs generell schneller trainiert werden, insbesondere bei großen Trainingsdatensätzen, aber eine leicht schwächere Erkennungsrate haben. Es ist zu beachten, dass der Vergleich von optimal abgestimmten Parametern für die SVMs ausgeht.

Parameter

▷ **`WidthCharacter`** (input_control) . . . . . integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 8
Wertevorschläge: `WidthCharacter` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich: 4 ≤ `WidthCharacter` ≤ 20

▷ **`HeightCharacter`** (input_control) . . . . . integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 10
Wertevorschläge: `HeightCharacter` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich: 4 ≤ `HeightCharacter` ≤ 20

▷ **`Interpolation`** (input_control) . . . . . string ⇝ *string*
Interpolationsmodus bei der Skalierung der Zeichen.
Defaultwert: 'constant'
Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}

▷ **`Features`** (input_control) . . . . . string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden.
Defaultwert: 'default'
Werteliste: `Features` ∈ {'default', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'projection_horizontal', 'projection_horizontal_invar', 'projection_vertical', 'projection_vertical_invar', 'ratio', 'anisometry', 'width', 'height', 'zoom_factor', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'moments_gray_plane', 'phi', 'num_connect', 'num_holes', 'cooc', 'num_runs', 'chord_histo'}

▷ **`Characters`** (input_control) . . . . . string-array ⇝ *string*
Alle Zeichen des zu lesenden Zeichensatzes.
Defaultwert: ['0','1','2','3','4','5','6','7','8','9']

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Dieser Parameter wird noch nicht unterstützt.
Defaultwert: []
Werteliste: GenParamName ∈ {[]}

▷ **GenParamValue** (input_control) . . . . . number-array ⇝ *integer* / string
Dieser Parameter wird noch nicht unterstützt.
Defaultwert: []
Werteliste: GenParamValue ∈ {[]}

▷ **OCRHandle** (output_control) . . . . . ocr_knn ⇝ *handle*
Handle des k-NN Klassifikators.

Beispiel

```
read_image (Image, 'letters')
* Segment the image.
binary_threshold(Image,&Region, 'otsu', 'dark', &UsedThreshold);
dilation_circle (Region, RegionDilation, 3.5)
connection (RegionDilation, ConnectedRegions)
intersection (ConnectedRegions, Region, RegionIntersection)
sort_region (RegionIntersection, Characters, 'character', 'true', 'row')
* Generate the training file.
count_obj (Characters, Number)
Classes := []
for J := 0 to 25 by 1
    Classes := [Classes,gen_tuple_const(20,chr(ord('a')+J))]
endfor
Classes := [Classes,gen_tuple_const(20,'.')]
write_ocr_trainf (Characters, Image, Classes, 'letters.trf')
* Generate and train the classifier.
read_ocr_trainf_names ('letters.trf', CharacterNames, CharacterCount)
create_ocr_class_knn (8, 10, 'constant', 'default', CharacterNames, \
                      [], [], OCRHandle)
trainf_ocr_class_knn (OCRHandle, 'letters.trf', [], [])
* Re-classify the characters in the image.
do_ocr_multi_class_knn (Characters, Image, OCRHandle, Class, Confidence)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_ocr_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

trainf_ocr_class_knn

Alternativen

create_ocr_class_svm

Siehe auch

do_ocr_single_class_knn, do_ocr_multi_class_knn, clear_class_knn, create_class_knn, trainf_ocr_class_knn, classify_class_knn

Modul

OCR/OCV

deserialize_ocr_class_knn (: : SerializedItemHandle : OCRHandle)

Deserialisieren eines serialisierten k-NN-basierten OCR-Klassifikators.

deserialize_ocr_class_knn deserialisiert einen k-NN-basierten OCR-Klassifikator, welcher mit serialize_ocr_class_knn serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Der serialisierte OCR-Klassifikator wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten OCR-Klassifikator mit dem Handle OCRHandle gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **OCRHandle** (output_control) . . . . . ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_ocr_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_ocr_class_knn

Nachfolger

classify_class_knn

Siehe auch

create_ocr_class_knn, serialize_ocr_class_knn

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

OCR/OCV

do_ocr_multi_class_knn (Character, Image : : OCRHandle : Class, Confidence)

Klassifikation mehrerer Zeichen mit einem OCR-Klassifikator.

do_ocr_multi_class_knn berechnet mit dem durch OCRHandle gegebenen OCR-Klassifikator zu den durch die Regionen Character und die Grauwerte Image gegebenen Zeichen die jeweils beste Klasse und gibt die Klassen in Class und die zugehörigen Konfidenzen der Klassen in Confidence zurück. Die Konfidenzen liegen zwischen 0.0 und 1.0. Je höher der Wert, desto sicherer ist die Klassifikation der einzelnen Zeichen. Im Gegensatz zu do_ocr_single_class_knn kann do_ocr_multi_class_knn also mehrere Zeichen gleichzeitig klassifizieren und ist damit typischerweise schneller als eine Schleife, die do_ocr_single_class_knn zur Klassifikation einzelner Zeichen verwendet. Allerdings kann do_ocr_multi_class_knn nur die beste Klasse des jeweiligen Zeichens zurückliefern.

Der OCR-Klassifikator muss vor der Verwendung von do_ocr_multi_class_knn mit trainf_ocr_class_knn trainiert werden.

Parameter

- ▷ **Character** (input_object) . . . . . region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte der Zeichen.
- ▷ **OCRHandle** (input_control) . . . . . ocr_knn ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Class** (output_control) . . . . . string(-array) ⇝ *string*
  Ergebnis der Klassifikation der Zeichen durch das k-NN.
  Parameteranzahl: Class == Character
- ▷ **Confidence** (output_control) . . . . . real(-array) ⇝ *real*
  Konfidenz der Klasse der Zeichen.
  Parameteranzahl: Confidence == Character

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `do_ocr_multi_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`trainf_ocr_class_knn`, `read_ocr_class_knn`

Alternativen

`do_ocr_single_class_knn`

Siehe auch

`create_ocr_class_knn`, `classify_class_knn`

Modul

OCR/OCV

do_ocr_single_class_knn (Character, Image : : OCRHandle, NumClasses, NumNeighbors : Class, Confidence)

Klassifikation eines Zeichens mit einem OCR-Klassifikator.

`do_ocr_single_class_knn` berechnet mit dem durch `OCRHandle` gegebenen k-NN-Klassifikator zu den Eigenschaften des durch die Region `Character` und die Grauwerte `Image` gegebenen Zeichens die `NumNeighbors` besten Trainingsbeispiele und gibt deren Klassen in `Class` und die zugehörigen Konfidenzen in `Confidence` zurück. Gehören die ersten `NumNeighbors` Trainingsbeispiele zu verschiedenen Klassen, werden maximal `NumClasses` Klassen sortiert nach Häufigkeit und gewichteter Distanz zurückgegeben. Die Konfidenzen liegen zwischen 0.0 und 1.0. Je höher der Wert, desto sicherer ist die Klassifikation. Die Konfidenzen können nur robust ermittelt werden, wenn `NumNeighbors` ausreichend groß ist.

Da potentiell mehrere Klassen zurückgeliefert werden können, darf `Character` nur eine Region (ein Zeichen) enthalten. Falls mehrere Zeichen in einem Aufruf klassifiziert werden sollen, muss `do_ocr_multi_class_knn` verwendet werden. Vor dem Aufruf von `do_ocr_single_class_knn` sollte der OCR-Klassifikator bereits mit `trainf_ocr_class_knn` oder `select_feature_set_trainf_knn` trainiert worden sein. Wenn nur eine Klasse unter den nächsten `NumNeighbors` Nachbarn ist, wird auch nur eine Klasse zurückgegeben.

Parameter

- ▷ **Character** (input_object) . . . . . region ⇝ *object*
  Zu erkennendes Zeichen.
- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte des Zeichens.

▷ **OCRHandle** (input_control) . . . . . ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **NumClasses** (input_control) . . . . . integer-array ⇝ *integer*
Anzahl der maximal zu bestimmenden besten Klassen.
Defaultwert: 1
Wertevorschläge: NumClasses ∈ {1, 2, 3, 4, 5}

▷ **NumNeighbors** (input_control) . . . . . integer-array ⇝ *integer*
Anzahl der zu bestimmenden nächsten Nachbarn.
Defaultwert: 1
Wertevorschläge: NumNeighbors ∈ {1, 2, 3, 4, 5}

▷ **Class** (output_control) . . . . . string(-array) ⇝ *string*
Ergebnis der Klassifikation des Zeichens durch das k-NN.

▷ **Confidence** (output_control) . . . . . real(-array) ⇝ *real*
Konfidenz(en) der Klasse(n).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert do_ocr_single_class_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_knn, read_ocr_class_knn

Alternativen

do_ocr_multi_class_knn

Siehe auch

create_ocr_class_knn, classify_class_knn

Modul

OCR/OCV

do_ocr_word_knn (Character, Image : : OCRHandle, Expression, NumAlternatives, NumCorrections : Class, Confidence, Word, Score)

Klassifikation einer Zeichengruppe mit einem OCR-Klassifikator.

do_ocr_word_knn verhält sich wie do_ocr_multi_class_knn, indem es mit dem durch OCRHandle gegebenen OCR-Klassifikator zu den durch die Regionen Character und die Grauwerte Image gegebenen Zeichen die jeweils beste Klasse berechnet, und die Klassen in Class und die zugehörigen Konfidenzen der Klassen in Confidence zurückgibt. Die Konfidenzen liegen zwischen 0.0 und 1.0. Je höher der Wert, desto sicherer ist die Klassifikation der einzelnen Zeichen.

Im Gegensatz zu do_ocr_multi_class_knn behandelt do_ocr_word_knn die Zeichengruppe als eine Einheit, die durch Aneinanderhängen von Klassennamen für jedes einzelne Zeichen ein Wort Word liefert. Dies ermöglicht die Einschränkung der erlaubten Klassifikationsergebnisse auf Textebene durch Angabe eines Ausdrucks Expression, der das erwartete Wort beschreibt.

Ein Ausdruck kann das Wort auf ein festes Lexikon, welches zuvor durch create_lexicon oder import_lexicon erzeugt wurde, einschränken, indem der Name des Lexikons in spitzen Klammern wie in *'<meinlexikon>'* angegeben wird. Hat der Ausdruck eine andere Form, so wird er als regulärer Ausdruck mit derselben Syntax wie für tuple_regexp_match interpretiert. Bei Verwendung von Quantifikatoren wie '*' sollte man normalerweise mit einem Ausdruck der Form '^...*'$*'' dafür sorgen, dass das gesamte Wort verwendet wird.

Im Gegensatz zu tuple_regexp_match erlaubt do_ocr_word_knn **keine** Angabe von zusätzlichen Optionen für Expression in einem Tupel.

Falls das aus den jeweils besten Klassen abgeleitete Wort dem Ausdruck nicht genügt, versucht do_ocr_word_knn eine Korrektur durch Berücksichtigung der NumAlternatives besten Klassen für jedes Zeichen, so wie sie von do_ocr_single_class_knn für ein einzelnes Zeichen bestimmt werden. Hierzu werden alle möglichen Korrekturen ausprobiert, bei denen die Klassifikation von höchstens NumCorrections Zeichenregionen geändert wird. Zu beachten ist hierbei, dass NumAlternatives und NumCorrections Auswirkungen auf die Komplexität des Algorithmus haben und daher in manchen Fällen intern beschränkt werden müssen. Für weitere Informationen siehe den Abschnitt 'Komplexität' weiter unten.

Falls der Ausdruck ein Lexikon beschreibt und die obige Prozedur nicht zum Erfolg führte, so wird das ähnlichste Wort aus dem Lexikon zurückgegeben, solange für die Korrektur weniger als NumCorrections Editieroperationen benötigt werden (siehe suggest_lexicon).

Das Ergebnis wird mit einem Score zwischen 0.0 (keine Korrektur gefunden) und 1.0 (ursprüngliches Wort war schon korrekt) bewertet. Verringert wird der Score durch das Anbringen einer Strafe entsprechend der Anzahl korrigierter Zeichen und einer (kleineren) Strafe für das Ignorieren von Klassen mit höherer Konfidenz um einen Ausdruck Expression zu erhalten:

$$\texttt{Score} = 1.0 - \texttt{PenaltyCorrections} - \texttt{PenaltyAlternatives}$$

$$\begin{aligned} \texttt{PenaltyCorrections} &= \alpha * \texttt{num_corr} \\ \texttt{PenaltyAlternatives} &= \alpha * \beta * \texttt{num_alt} \end{aligned}$$

wobei num_corr die Anzahl tatsächlich angebrachter Korrekturen und num_alt die Anzahl ignorierter Alternativen ist.

$$\begin{aligned} \alpha &= \frac{1}{\texttt{NumCorrections} + 1} \\ \beta &= \frac{1}{\texttt{NumCorrections} * (\texttt{NumAlternatives} - 1) + 2} \end{aligned}$$

Diese Bewertung ist rein kombinatorisch und berücksichtigt **nicht** die ursprünglichen Konfidenzen der besten Klassen.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte der Zeichen.
- ▷ **OCRHandle** (input_control) ..... ocr_knn ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Expression** (input_control) ..... string ⇝ *string*
  Ausdruck, der die erlaubte Wortstruktur beschreibt
- ▷ **NumAlternatives** (input_control) ..... integer ⇝ *integer*
  Anzahl der Klassen pro Zeichen, die für die interne Wortkorrektur berücksichtigt werden.
  Defaultwert: 3
  Wertevorschläge: NumAlternatives ∈ {3, 4, 5}
  Typischer Wertebereich: $1 \leq$ NumAlternatives ≤ 5
- ▷ **NumCorrections** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl der korrigierten Zeichen.
  Defaultwert: 2
  Wertevorschläge: NumCorrections ∈ {1, 2, 3, 4, 5}
  Typischer Wertebereich: $0 \leq$ NumCorrections ≤ 5
- ▷ **Class** (output_control) ..... string(-array) ⇝ *string*
  Ergebnis der Klassifikation der Zeichen durch das k-NN.
  Parameteranzahl: Class == Character

▷ **Confidence** (output_control) . . . . . real(-array) ⇝ *real*
Konfidenz der Klasse der Zeichen.
Parameteranzahl: Confidence == Character

▷ **Word** (output_control) . . . . . string ⇝ *string*
Text des Wortes nach Klassifikation und Korrektur.

▷ **Score** (output_control) . . . . . real ⇝ *real*
Ähnlichkeitsmaß von korrigiertem Wort zum Original.

Komplexität

Die Komplexität des Prüfens aller möglichen Korrekturen hat die Größenordnung $O((n * a)^{min(c,n)})$, wobei a die Anzahl der Alternativen, n die Anzahl der Zeichenregionen und c die Anzahl der erlaubten Korrekturen ist. Als Absicherung gegen quasi-unendliche Schleifen im Falle von großem n wird der Wert von c intern beschränkt auf 5, 3, oder 1, falls $a * n >= 30$, 60, bzw. 90.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `do_ocr_word_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`trainf_ocr_class_knn`, `read_ocr_class_knn`

Alternativen

`do_ocr_multi_class_knn`

Siehe auch

`create_ocr_class_knn`, `classify_class_knn`

Modul

OCR/OCV

get_features_ocr_class_knn (Character : : OCRHandle, Transform : Features)

Berechnung der Merkmale eines Zeichens.

`get_features_ocr_class_knn` berechnet die Merkmale des Zeichens, das durch `Character` angegeben wird, mit dem OCR-Klassifikator `OCRHandle` und gibt sie in `Features` zurück. Im Gegensatz zu `do_ocr_single_class_knn` und `do_ocr_multi_class_knn` wird das Zeichen als ein Bildobjekt angegeben. Vor der Verwendung von `get_features_ocr_class_knn` muss also `reduce_domain` aufgerufen werden. Der Parameter `Transform` legt fest, ob die im Klassifikator durch `Preprocessing` in `create_ocr_class_knn` festgelegte Merkmalstransformation angewendet werden soll (`Transform` = *'true'*) oder ob die untransformierten Merkmale zurückgegeben werden sollen (`Transform` = *'false'*). `get_features_ocr_class_knn` kann dazu verwendet werden, die zur Klassifikation verwendeten Merkmale zu inspizieren.

Parameter

▷ **Character** (input_object) . . . . . singlechannelimage ⇝ *object* : byte / uint2
Eingabezeichen.

▷ **OCRHandle** (input_control) . . . . . ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **Transform** (input_control) . . . . . string ⇝ *string*
Soll der Merkmalsvektor mit der Vorverarbeitung transformiert werden?
Defaultwert: 'true'
Werteliste: `Transform` ∈ {'true', 'false'}

▷ **Features** (output_control) . . . . . real-array ⇝ *real*
Merkmalsvektor des Zeichens.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_features_ocr_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`trainf_ocr_class_knn`

Siehe auch

`create_ocr_class_knn`

Modul

OCR/OCV

get_params_ocr_class_knn (: : OCRHandle : WidthCharacter, HeightCharacter, Interpolation, Features, Characters, Preprocessing, NumTrees)

Auslesen der Parameter eines OCR-Klassifikators.

`get_params_ocr_class_knn` gibt die Parameter des durch `OCRHandle` angegebenen OCR-Klassifikators, die bei der Erzeugung mit `create_ocr_class_knn` verwendet wurden, zurück. Dies ist insbesondere nützlich, wenn der Klassifikator mit `read_ocr_class_knn` eingelesen wurde. Die Ausgabedaten von `get_params_ocr_class_knn` können z.B. verwendet werden, um zu überprüfen, ob ein zu lesendes Zeichen in dem Klassifikator vorhanden ist. Zur Beschreibung der Parameter siehe `create_ocr_class_knn` und `trainf_ocr_class_knn`.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **WidthCharacter** (output_control) . . . . . integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.

▷ **HeightCharacter** (output_control) . . . . . integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.

▷ **Interpolation** (output_control) . . . . . string ⇝ *string*
Interpolationsmodus bei der Skalierung der Zeichen.

▷ **Features** (output_control) . . . . . string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden.

▷ **Characters** (output_control) . . . . . string-array ⇝ *string*
Zeichen des zu lesenden Zeichensatzes.

▷ **Preprocessing** (output_control) . . . . . string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.

▷ **NumTrees** (output_control) . . . . . integer ⇝ *integer*
Anzahl an Bäumen die zur Klassifikation verwendet werden.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_params_ocr_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_ocr_class_mlp`, `read_ocr_class_mlp`

Nachfolger

`do_ocr_single_class_mlp`, `do_ocr_multi_class_mlp`

Siehe auch

`trainf_ocr_class_mlp`, `get_params_class_mlp`

Modul

OCR/OCV

read_ocr_class_knn (: : FileName : OCRHandle)

Lesen eines OCR-Klassifikators aus einer Datei.

`read_ocr_class_knn` liest einen mit `write_ocr_class_knn` abgespeicherten OCR-Klassifikator ein.

Parameter

▷ **FileName** (input_control) ..... filename.read ⇝ *string*
Name der Datei.
Dateiendung: `.onc`, `.okc`, `.fnt`

▷ **OCRHandle** (output_control) ..... ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_ocr_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`classify_class_knn`

Siehe auch

`create_ocr_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

OCR/OCV

select_feature_set_trainf_knn (: : TrainingFile, FeatureList, SelectionMethod, Width, Height, GenParamName, GenParamValue : OCRHandle, FeatureSet, Score)

Wählt die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus.

select_feature_set_trainf_knn wählt die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus. Die Daten für das OCR-Klassifikationsproblem werden mit der Trainingsdatei TrainingFile ausgewählt. Als Klassifikator wird ein k-Nearest-Neighbor-Klassifikator (kNN) verwendet.

Es wird eine Untermenge aller angegebenen OCR-Merkmale gewählt. Alle zu untersuchenden Merkmale werden in FeatureList spezifiziert. Die Liste der möglichen OCR-Merkmale wird in der Dokumentation von create_ocr_class_knn näher erläutert. Die letztendlich ausgewählte Merkmalsuntermenge wird in FeatureSet zurückgegeben.

select_feature_set_trainf_knn ist auf OCR-Klassifikationsprobleme spezialisiert und unterstützt nur Merkmale, die in der Auswahlliste stehen. Für andere Merkmale bietet sich somit die Nutzung des generellen Operators select_feature_set_knn an.

Für den Selektionsprozess können in SelectionMethod zwei verschiedene Methoden ausgewählt werden: entweder die Greedy-Variante *'greedy'* (das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt) oder die dynamisch oszillierende Suche *'greedy_oscillating'* (Das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt. Danach wird getestet ob eines der hinzugefügten Merkmale entbehrlich ist.). Während *'greedy'* schneller terminiert, kann *'greedy_oscillating'* bessere Ergebnisse erzielen wenn viele Trainingsdaten vorliegen.

Als Optimierungskriterium dient die Klassifikationsrate, die mit einem zwei-fachen Kreuzvalidierungsverfahren ermittelt wird. Die beste erreichte Klassifikationsrate wird in Score zurückgegeben. OCRHandle gibt einen Klassifikator zurück, der mit der entsprechenden Featureauswahl parametrisiert und trainiert wurde.

Mit Hilfe der Parameter GenParamName und GenParamValue können die folgenden Werte gesetzt werden:

'num_neighbors': Anzahl Nachbarn die ermittelt werden, das Erhöhen dieses Wertes führt zu besseren Ergebnissen bei längerer Laufzeit.
Possible values: '1', '2', '5', '10'
Default value: '1'

'num_trees': die Anzahl an Suchbäumen im k-NN
Possible values: '1', '4', '10'
Default Value: '4'

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lange sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator select_feature_set_trainf_knn zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

▷ **TrainingFile** (input_control) ..... filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: ''
Dateiendung: .trf, .otr

▷ **FeatureList** (input_control) ..... string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden können.
Defaultwert: ['zoom_factor','ratio','width','height','foreground','foreground_grid_9','foreground_grid_16', 'anisometry','compactness','convexity','moments_region_2nd_invar','moments_region_2nd_rel_invar', 'moments_region_3rd_invar','moments_central','phi','num_connect','num_holes','projection_horizontal', 'projection_vertical','projection_horizontal_invar','projection_vertical_invar','chord_histo','num_runs', 'pixel','pixel_invar','pixel_binary','gradient_8dir','cooc','moments_gray_plane']
Werteliste: FeatureList ∈ {'default', 'zoom_factor', 'ratio', 'width', 'height', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'anisometry', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'phi', 'num_connect', 'num_holes', 'projection_horizontal', 'projection_vertical', 'projection_horizontal_invar', 'projection_vertical_invar', 'chord_histo', 'num_runs', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'cooc', 'moments_gray_plane'}

▷ **SelectionMethod** (input_control) ..... string ⇝ *string*
Die Methode zur Auswahl der Merkmale.
Defaultwert: 'greedy'
Werteliste: SelectionMethod ∈ {'greedy', 'greedy_oscillating'}

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 15

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 16

▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
Namen von (optionalen) Parametern für die Steuerung des Verhaltens verwendeten des k-NN Klassifikators.
Defaultwert: []
Werteliste: GenParamName ∈ {'num_neighbors'}

▷ **GenParamValue** (input_control) ..... number-array ⇝ *real* / integer / string
Die zu den optionalen generischen Parametern gehörenden Werte.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {1, 2, 3}

▷ **OCRHandle** (output_control) ..... ocr_knn ⇝ *handle*
Ein trainierter k-NN Klassifikator.

▷ **FeatureSet** (output_control) ..... string-array ⇝ *string*
Die ausgewählten Merkmale.

▷ **Score** (output_control) ..... real-array ⇝ *real*
Die Klassifikationsrate die mit dem ausgewählten Merkmalssatz erreicht wurde.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert select_feature_set_trainf_knn den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Alternativen

select_feature_set_trainf_svm, select_feature_set_trainf_mlp

Siehe auch

select_feature_set_knn

Modul

OCR/OCV

serialize_ocr_class_knn (: : OCRHandle : SerializedItemHandle)

Serialisieren eines k-NN-basierten OCR-Klassifikators.

serialize_ocr_class_knn serialisiert einen k-NN-basierten OCR-Klassifikator (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_ocr_class_knn in eine Datei geschrieben werden. Der OCR-Klassifikator wird in dem Handle OCRHandle übergeben. Der serialisierte OCR-Klassifikator wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_ocr_class_knn wieder deserialisiert werden.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_ocr_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`trainf_ocr_class_knn`, `read_ocr_class_knn`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_ocr_class_knn`

Siehe auch

`create_class_knn`, `read_ocr_class_knn`, `deserialize_ocr_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

OCR/OCV

trainf_ocr_class_knn (: : OCRHandle, TrainingFile, GenParamName, GenParamValue :)

Trainieren eines OCR-Klassifikators.

`trainf_ocr_class_knn` trainiert den OCR-Klassifikator `OCRHandle` mit den Trainingszeichen, die in den durch `TrainingFile` angegebenen OCR-Trainingsdateien enthalten sind. Die Trainingsdateien müssen zuvor, z.B. mit `write_ocr_trainf`, erzeugt worden sein. Es ist zu beachten, dass Trainingszeichen, die keine entsprechende Klasse im Klassifikator `OCRHandle` haben, verworfen werden. Als generische Parameter stehen folgenden Möglichkeiten zur Verfügung: Die Anzahl an Suchbäumen kann mit *'num_trees'* (Standardwert: 4) gesteuert werden und zum zweiten kann eine Normalisierung der Daten aktiviert werden mit *'normalization'* (Standardwert: *'false'*). Eine höhere Anzahl Suchbäume erhöht die Genauigkeit der Suche, führt aber auch zu höheren Klassifikationszeiten. Die Normalisierung verändert die bereits hinzugefügten Trainingsdaten, wodurch keine weiteren Trainingsdaten hinzugefügt werden können.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **TrainingFile** (input_control) . . . . . filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: 'ocr.trf'
Dateiendung: `.trf`, `.otr`

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen von (optionalen) Parametern für die Steuerung des Verhaltens des Operators.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'num_trees', 'normalization'}

▷ **GenParamValue** (input_control) . . . . . number-array ⇝ *integer* / string / real
Die zu den optionalen generischen Parametern gehörenden Werte.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {4, 5, 'false', 'true'}

Beispiel

```
* Train an OCR classifier
read_ocr_trainf_names ('ocr.trf', CharacterNames, CharacterCount)
create_ocr_class_knn (8, 10, 'constant', 'default', CharacterNames, [],\
                      [], OCRHandle)
trainf_ocr_class_knn (OCRHandle, 'ocr.trf', [], [])
write_ocr_class_knn (OCRHandle, 'ocr.omc')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `trainf_ocr_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ocr_class_knn`, `write_ocr_trainf`, `append_ocr_trainf`, `write_ocr_trainf_image`

Nachfolger

`do_ocr_single_class_knn`, `do_ocr_multi_class_knn`

Alternativen

`read_ocr_class_knn`

Siehe auch

`train_class_knn`

Modul

OCR/OCV

write_ocr_class_knn (: : OCRHandle, FileName :)

Abspeichern eines OCR-Klassifikators in eine Datei.

`write_ocr_class_knn` speichert den OCR-Klassifikator `OCRHandle` in der Datei `FileName` ab. Falls keine Datei-Extension in `FileName` angegeben wird, wird die Standard-Extension *'.onc'* an `FileName` angehängt. `write_ocr_class_knn` wird typischerweise aufgerufen, nachdem der Klassifikator mit `trainf_ocr_class_knn` trainiert wurde. Der Klassifikator kann mit `read_ocr_class_knn` wieder eingelesen werden.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_knn ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.onc`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_ocr_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`trainf_ocr_class_knn`, `read_ocr_class_knn`

Siehe auch

`create_class_knn`, `read_ocr_class_knn`

Literatur

Marius Muja, David G. Lowe: „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration“; International Conference on Computer Vision Theory and Applications (VISAPP 09); 2009.

Modul

OCR/OCV

20.4 Lexika

`clear_lexicon` `( : : LexiconHandle : )`

Löschen eines Lexikons.

`clear_lexicon` löscht ein Lexikon und gibt seine Ressourcen frei.

Parameter

▷ **`LexiconHandle`** (input_control) . . . . . lexicon ⇝ *handle*
Handle des Lexikons.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- LexiconHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

`create_lexicon`

Modul

OCR/OCV

`create_lexicon` `( : : Name, Words : LexiconHandle )`

Erzeugen eines Lexikons aus einem Tupel von Wörtern.

`create_lexicon` erzeugt ein neues Lexikon basierend auf dem Tupel von Wörtern in `Words`. Durch die Vergabe eines eindeutigen Namens in `Name` kann das Lexikon später aus textuellen Ausdrücken heraus referenziert werden, wie z.B. bei `do_ocr_word_mlp`.

Die Unterstützung von Lexika in HALCON zielt derzeit nicht auf natürliche Sprachen ab, sondern auf OCR Applikationen, die lediglich zwischen einem begrenzten Satz von nicht mehr als einigen tausend gültigen Wörtern unterscheiden müssen, z.B. Länder- oder Produktnamen. MVTec selbst bietet keine fertigen Lexika an.

Parameter

- ▷ **Name** (input_control) . . . . . string ⇝ *string*
  Eindeutiger Name für das neue Lexikon.
  Defaultwert: 'lex1'
- ▷ **Words** (input_control) . . . . . string-array ⇝ *string*
  Liste von Worten für das neue Lexikon.
  Defaultwert: ['word1','word2','word3']
- ▷ **LexiconHandle** (output_control) . . . . . lexicon ⇝ *handle*
  Handle des Lexikons.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`do_ocr_word_mlp`, `do_ocr_word_svm`

Alternativen

`import_lexicon`

Siehe auch

`lookup_lexicon`, `suggest_lexicon`

Modul

OCR/OCV

`import_lexicon` (: : Name, FileName : LexiconHandle)

Erzeugen eines Lexikons aus einer Textdatei.

`import_lexicon` erzeugt ein neues Lexikon basierend auf einer Liste von Wörtern in der Datei `FileName`. Durch die Vergabe eines eindeutigen Namens in `Name` kann das Lexikon später aus textuellen Ausdrücken heraus referenziert werden, wie z.B. bei `do_ocr_word_mlp`.

Die Unterstützung von Lexika in HALCON zielt derzeit nicht auf natürliche Sprachen ab, sondern auf OCR Applikationen, die lediglich zwischen einem begrenzten Satz von nicht mehr als einigen tausend gültigen Wörtern unterscheiden müssen, z.B. Länder- oder Produktnamen. Sind in der Lexikon-Datei Einträge mit Sonderzeichen enthalten, muss die Datei UTF-8-kodiert sein. Alte Dateien, die in der lokalen 8-Bit-Kodierung vorliegen, können allerdings weiterhin geladen werden, solange sie mindestens eine Bytefolge enthalten, die eindeutig nicht auch als UTF-8-Zeichen missinterpretiert werden könnte. MVTec selbst bietet keine fertigen Lexika an.

Parameter

- ▷ **Name** (input_control) . . . . . string ⇝ *string*
  Eindeutiger Name für das neue Lexikon.
  Defaultwert: 'lex1'
- ▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
  Name einer Textdatei mit Worten für das neue Lexikon.
  Defaultwert: 'words.txt'
  Dateiendung: `.txt`
- ▷ **LexiconHandle** (output_control) . . . . . lexicon ⇝ *handle*
  Handle des Lexikons.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`do_ocr_word_mlp`, `do_ocr_word_svm`

Alternativen

`create_lexicon`

Siehe auch

`lookup_lexicon`, `suggest_lexicon`

Modul

OCR/OCV

`inspect_lexicon` (: : LexiconHandle : Words)

Abfrage aller Wörter eines Lexikons.

`inspect_lexicon` liefert in `Words` ein Tupel mit allen Wörtern aus dem Lexikon.

Parameter

▷ **`LexiconHandle`** (input_control) ..... lexicon ⇝ *handle*
Handle des Lexikons.

▷ **`Words`** (output_control) ..... string(-array) ⇝ *string*
Liste aller Wörter.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`lookup_lexicon`

Siehe auch

`create_lexicon`

Modul

OCR/OCV

`lookup_lexicon` (: : LexiconHandle, Word : Found)

Prüfen, ob ein Wort in einem Lexikon enthalten ist.

`lookup_lexicon` prüft, ob `Word` im Lexikon `LexiconHandle` enthalten ist, und liefert 1 in `Found`, falls das Wort gefunden wurde, ansonsten 0.

Parameter

- ▷ **LexiconHandle** (input_control) . . . . . lexicon ⇝ *handle*
  Handle des Lexikons.
- ▷ **Word** (input_control) . . . . . string ⇝ *string*
  Nachzuschlagendes Wort.
  Defaultwert: 'word'
- ▷ **Found** (output_control) . . . . . integer ⇝ *integer*
  Ergebnis der Suche.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

suggest_lexicon

Siehe auch

create_lexicon

Modul

OCR/OCV

suggest_lexicon (: : LexiconHandle, Word : Suggestion, NumCorrections)

Finden eines ähnlichen Wortes in einem Lexikon.

suggest_lexicon vergleicht Word mit allen Wörtern im Lexikon und berechnet die minimale Anzahl von Editieroperationen NumCorrections, welche nötig sind um Word in ein Wort aus dem Lexikon zu überführen. Gültige Operationen sind das Einfügen, Löschen und Ersetzen von Zeichen. Das ähnlichste Wort aus dem Lexikon wird in Suggestion zurückgegeben. Wenn es mehrere Wörter mit derselben minimalen Anzahl von Korrekturen gibt, wird nur das erste dieser Wörter zurückgegeben.

Parameter

- ▷ **LexiconHandle** (input_control) . . . . . lexicon ⇝ *handle*
  Handle des Lexikons.
- ▷ **Word** (input_control) . . . . . string ⇝ *string*
  Nachzuschlagendes Wort.
  Defaultwert: 'word'
- ▷ **Suggestion** (output_control) . . . . . string ⇝ *string*
  Ähnlichstes Wort aus dem Lexikon.
- ▷ **NumCorrections** (output_control) . . . . . integer ⇝ *integer*
  Unterschied der Wörter in Editieroperationen

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

lookup_lexicon

Siehe auch

create_lexicon

Literatur

Vladimir I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Doklady Akademii Nauk SSSR, 163(4):845-848, 1965 (Russian). English translation in Soviet Physics Doklady, 10(8):707-710, 1966.

Modul

OCR/OCV

20.5 Neuronale Netze

clear_ocr_class_mlp (: : OCRHandle :)

Löschen eines OCR-Klassifikators.

clear_ocr_class_mlp löscht den durch OCRHandle angegebenen OCR-Klassifikator, der mit create_ocr_class_mlp erzeugt wurde, und gibt sämtlichen für den Klassifikator benötigten Speicher frei. Der Klassifikator kann nach dem Aufruf von clear_ocr_class_mlp nicht mehr verwendet werden. Der Handle OCRHandle ist nach dem Aufruf ungültig.

Parameter

▷ **OCRHandle** (input_control) ..... ocr_mlp(-array) ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Falls OCRHandle korrekt ist, dann liefert clear_ocr_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

do_ocr_single_class_mlp, do_ocr_multi_class_mlp

Siehe auch

create_ocr_class_mlp, read_ocr_class_mlp, write_ocr_class_mlp, trainf_ocr_class_mlp

Modul

OCR/OCV

create_ocr_class_mlp (: : WidthCharacter, HeightCharacter, Interpolation, Features, Characters, NumHidden, Preprocessing, NumComponents, RandSeed : OCRHandle)

Erzeugen eines OCR-Klassifikators unter Verwendung eines mehrschichtigen Perzeptrons.

create_ocr_class_mlp erzeugt einen OCR-Klassifikator unter Verwendung eines mehrschichtigen Perzeptrons (MLP). Der Handle des OCR-Klassifikators wird in OCRHandle zurückgegeben.

Für eine Beschreibung der Funktionsweise eines MLP siehe create_class_mlp. create_ocr_class_mlp erzeugt ein MLP mit OutputFunction = *'softmax'*. Die Länge des

Merkmalsvektors des MLP (NumInput in create_class_mlp) wird aus den für die OCR zu verwendenden Merkmalen, die in Features übergeben werden, festgelegt. Die Merkmale sind unten beschrieben. Die Anzahl der Einheiten in der versteckten Schicht wird durch NumHidden festgelegt. Die Anzahl der Ausgabevariablen des MLP (NumOutput in create_class_mlp) wird aus den für die OCR zu verwendenden Namen der Zeichen, die in Characters übergeben werden, festgelegt. Wie bei create_class_mlp beschrieben, kann mit Preprocessing und NumComponents eine Vorverarbeitung der Daten (Merkmalsvektoren) festgelegt werden. Bei der OCR werden die Merkmale intern schon annähernd normalisiert, so dass im Normalfall hier *'none'* angegeben werden kann. Der Parameter RandSeed hat dieselbe Bedeutung wie bei create_class_mlp. Weiterhin kann es wie bei allgemeinen MLP-Klassifikatoren (siehe create_class_mlp and set_regularization_params_class_mlp) wünschenswert sein, einen OCR-Klassifikator zu regularisieren. Dies kann durch einen Aufruf von set_regularization_params_ocr_class_mlp vor dem Training des OCR-Klassifikators erreicht werden. Zusätzlich kann es wie bei allgemeinen MLP-Klassifikatoren (siehe create_class_mlp and set_rejection_params_class_mlp) wünschenswert sein, den OCR-Klassifikator mit der Fähigkeit auszustatten, unbekannte Zeichen zurückzuweisen. Die Rückweisungsklasse ist bei Konvention das zusätzliche Symbol chr(26), dass in Characters übergeben werden muss. Die Parameter der Rückweisungsklasse kann durch einen Aufruf von set_rejection_params_ocr_class_mlp vor dem Training des OCR-Klassifikators gesetzt werden.

Die zur Klassifikation zu verwendenden Merkmale werden mit Features angegeben. Dabei kann Features ein Tupel mit mehreren Merkmalsnamen beinhalten. Jeder dieser Merkmalsnamen führt zur Berechnung von einem oder mehreren Merkmalen für den Klassifikator. Einige der Merkmalsnamen erzeugen Grauwertmerkmale (z.B. *'pixel_invar'*). Da ein Klassifikator immer eine konstante Anzahl von Merkmalen (Eingabevariablen) benötigt, wird ein zu klassifizierendes Zeichen auf eine Standardgröße transformiert, die durch WidthCharacter und HeightCharacter festgelegt wird. Die bei der Transformation anzuwendende Interpolation wird mit Interpolation festgelegt. Sie hat dieselbe Bedeutung wie bei affine_trans_image. Die Interpolation sollte so gewählt werden, dass bei der Transformation keine Aliasing-Effekte auftreten. Im Normalfall sollte Interpolation = *'constant'* verwendet werden. Bei der Wahl der Größe des transformierten Zeichens ist zu beachten, dass die Größe nicht zu groß gewählt wird, da die Generalisierungseigenschaften des Klassifikators sonst sehr schlecht werden können. Insbesondere können bei großen Größen kleine Segmentierungsfehler einen großen Einfluss auf die berechneten Merkmale haben, wenn Grauwertmerkmale verwendet werden, da sich durch die Segmentierungsfehler das umschließende Rechteck des Zeichens ändern kann, was dazu führt, dass das Zeichen anders als trainiert skaliert wird. Im Normalfall sollten Größen zwischen 6×8 und 10×14 verwendet werden.

Mit Features können die folgenden Merkmalsnamen zur Erkennung der Zeichen ausgewählt werden.

'default' *'ratio'* und *'pixel_invar'* werden ausgewählt.

'pixel' Grauwerte des Zeichens (WidthCharacter × HeightCharacter Merkmale).

'pixel_invar' Grauwerte des Zeichens mit automatischer maximaler Grauwertspreizung (WidthCharacter × HeightCharacter Merkmale).

'pixel_binary' Region des Zeichens als Binärbild auf eine Größe von WidthCharacter × HeightCharacter skaliert. (WidthCharacter × HeightCharacter Merkmale).

'gradient_8dir' Die Gradienten des Zeichens werden berechnet. Die Gradientenrichtungen werden in 8 Richtungen diskretisiert. Das Amplitudenbild wird entsprechend der diskretisierten Richtungen in 8 Kanäle aufgeteilt. In jedem der 8 Kanäle werden auf einem 5×5 Gitter Stichproben entnommen. Diese Stichproben bilden die Merkmale (200 Merkmale).

'projection_horizontal' Horizontale Projektion der Grauwerte (siehe gray_projections, HeightCharacter Merkmale).

'projection_horizontal_invar' Horizontale Projektion der Grauwerte, die maximal gespreizt sind (HeightCharacter Merkmale).

'projection_vertical' Vertikale Projektion der Grauwerte (siehe gray_projections, WidthCharacter Merkmale).

'projection_vertical_invar' Vertikale Projektion der Grauwerte, die maximal gespreizt sind (WidthCharacter Merkmale).

'ratio' Seitenverhältnis des Zeichens (siehe height_width_ratio, 1 Merkmal).

'anisometry' Formmerkmal Anisometrie (siehe eccentricity, 1 Merkmal).

'width' Breite des Zeichens vor der Normierung auf die Standardgröße (nicht skalierungsinvariant, siehe height_width_ratio, 1 Merkmal).

'height' Höhe des Zeichens vor der Normierung auf die Standardgröße (nicht skalierungsinvariant, siehe `height_width_ratio`, 1 Merkmal).

'zoom_factor' Größenunterschied zwischen aktuellem Zeichen und den Werten von `WidthCharacter` und `HeightCharacter` (nicht skalierungsinvariant, 1 Merkmal).

'foreground' Relativer Anteil der Vordergrund-Pixel (1 Merkmal).

'foreground_grid_9' Relativer Anteil der Vordergrund-Pixel in einem 3×3 Raster innerhalb des umschließenden Rechtecks des Zeichens (9 Merkmale).

'foreground_grid_16' Relativer Anteil der Vordergrund-Pixel in einem 4×4 Raster innerhalb des umschließenden Rechtecks des Zeichens (16 Merkmale).

'compactness' Formmerkmal Kompaktheit (siehe `compactness`, 1 Merkmal).

'convexity' Formmerkmal Konvexität (siehe `convexity`, 1 Merkmal).

'moments_region_2nd_invar' Normierte zweite geometrische Momente der Region (siehe `moments_region_2nd_invar`, 3 Merkmale).

'moments_region_2nd_rel_invar' Normierte zweite relative geometrische Momente der Region (siehe `moments_region_2nd_rel_invar`, 2 Merkmale).

'moments_region_3rd_invar' Normierte dritte geometrische Momente der Region (siehe `moments_region_3rd_invar`, 4 Merkmale).

'moments_central' Normierte zentrale geometrische Momente der Region (siehe `moments_region_central`, 4 Merkmale).

'moments_gray_plane' Normierte Grauwertmomente und die Winkel der Grauwertebene (siehe `moments_gray_plane`, 4 Merkmale)

'phi' Sinus und Kosinus der Orientierung (Winkel) des Zeichens (siehe `elliptic_axis`, 2 Merkmal).

'num_connect' Anzahl der Zusammenhangskomponenten (siehe `connect_and_holes`, 1 Merkmal).

'num_holes' Anzahl der Hohlflächen (siehe `connect_and_holes`, 1 Merkmal).

'cooc' Werte der binären Cooccurrence-Matrix (siehe `gen_cooc_matrix`, 8 Merkmale).

'num_runs' Anzahl der Sehnen der Region normiert auf die Höhe (1 Merkmal).

'chord_histo' Häufigkeit der Sehnen pro Zeile (nicht skalierungsinvariant, `HeightCharacter` Merkmale).

Nachdem der Klassifikator erzeugt wurde, wird er mit `trainf_ocr_class_mlp` trainiert. Daraufhin kann der Klassifikator mit `write_ocr_class_mlp` abgespeichert werden. Alternativ können auch sofort nach dem Training Zeichen mit `do_ocr_single_class_mlp` oder `do_ocr_multi_class_mlp` klassifiziert werden.

HALCON stellt eine Anzahl vortrainierter OCR-Klassifikatoren zur Verfügung (siehe „`Solution Guide I`", Kapitel 'OCR', Abschnitt 'Pretrained OCR Fonts'). Diese vortrainierten OCR-Klassifikatoren können direkt mit `read_ocr_class_mlp` eingelesen werden und ermöglichen es, eine Vielzahl an Zeichensätzen zu lesen, ohne vorher einen OCR-Klassifikator trainieren zu müssen. Es ist daher empfehlenswert, zuerst zu testen, ob einer der vortrainierten OCR-Klassifikatoren verwendet werden kann. Ist dies der Fall, erspart man sich das Erzeugen und Trainieren des OCR-Klassifikators.

Ein Vergleich zwischen MLP und Support-Vektor-Maschinen (SVM) (siehe `create_ocr_class_svm`) zeigt typischerweise, dass SVMs generell schneller trainiert werden, insbesondere bei großen Trainingsdatensätzen, und eine leicht verbesserte Erkennungsrate haben. Das MLP weist schnellere Klassifikationszeiten auf und sollte daher in zeitkritischen Anwendungen verwendet werden. Es ist zu beachten, dass der Vergleich von optimal abgestimmten Parametern ausgeht.

Parameter

▷ **`WidthCharacter`** (input_control) . . . . . integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 8
Wertevorschläge: `WidthCharacter` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich: $4 \leq$ `WidthCharacter` ≤ 20

▷ **`HeightCharacter`** (input_control) . . . . . integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 10
Wertevorschläge: `HeightCharacter` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich: $4 \leq$ `HeightCharacter` ≤ 20

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Interpolationsmodus bei der Skalierung der Zeichen.
Defaultwert: 'constant'
Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}

▷ **Features** (input_control) . . . . . string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden.
Defaultwert: 'default'
Werteliste: `Features` ∈ {'default', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'projection_horizontal', 'projection_horizontal_invar', 'projection_vertical', 'projection_vertical_invar', 'ratio', 'anisometry', 'width', 'height', 'zoom_factor', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'moments_gray_plane', 'phi', 'num_connect', 'num_holes', 'cooc', 'num_runs', 'chord_histo'}

▷ **Characters** (input_control) . . . . . string-array ⇝ *string*
Alle Zeichen des zu lesenden Zeichensatzes.
Defaultwert: ['0','1','2','3','4','5','6','7','8','9']

▷ **NumHidden** (input_control) . . . . . integer ⇝ *integer*
Anzahl der versteckten Einheiten des MLP.
Defaultwert: 80
Wertevorschläge: `NumHidden` ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 150}
Restriktion: NumHidden >= 1

▷ **Preprocessing** (input_control) . . . . . string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
Defaultwert: 'none'
Werteliste: `Preprocessing` ∈ {'none', 'normalization', 'principal_components', 'canonical_variates'}

▷ **NumComponents** (input_control) . . . . . integer ⇝ *integer*
Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale (ignoriert bei `Preprocessing` = *'none'* und `Preprocessing` = *'normalization'*).
Defaultwert: 10
Wertevorschläge: `NumComponents` ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Restriktion: NumComponents >= 1

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Initialisierungswert des Zufallszahlengenerators, der zur Initialisierung des MLP mit zufälligen Werten verwendet wird.
Defaultwert: 42

▷ **OCRHandle** (output_control) . . . . . ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

Beispiel

```
read_image (Image, 'letters')
* Segment the image.
binary_threshold(Image,&Region, 'otsu', 'dark', &UsedThreshold);
dilation_circle (Region, RegionDilation, 3.5)
connection (RegionDilation, ConnectedRegions)
intersection (ConnectedRegions, Region, RegionIntersection)
sort_region (RegionIntersection, Characters, 'character', 'true', 'row')
* Generate the training file.
count_obj (Characters, Number)
Classes := []
for J := 0 to 25 by 1
    Classes := [Classes,gen_tuple_const(20,chr(ord('a')+J))]
endfor
Classes := [Classes,gen_tuple_const(20,'.')]
write_ocr_trainf (Characters, Image, Classes, 'letters.trf')
* Generate and train the classifier.
read_ocr_trainf_names ('letters.trf', CharacterNames, CharacterCount)
create_ocr_class_mlp (8, 10, 'constant', 'default', CharacterNames, 20, \
                      'none', 81, 42, OCRHandle)
```

```
trainf_ocr_class_mlp (OCRHandle, 'letters.trf', 100, 0.01, 0.01, Error, \
                      ErrorLog)
* Re-classify the characters in the image.
do_ocr_multi_class_mlp (Characters, Image, OCRHandle, Class, Confidence)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_ocr_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`trainf_ocr_class_mlp`, `set_regularization_params_ocr_class_mlp`, `set_rejection_params_ocr_class_mlp`

Alternativen

`create_ocr_class_svm`

Siehe auch

`do_ocr_single_class_mlp`, `do_ocr_multi_class_mlp`, `clear_ocr_class_mlp`, `create_class_mlp`, `train_class_mlp`, `classify_class_mlp`

Modul

OCR/OCV

deserialize_ocr_class_mlp (: : SerializedItemHandle : OCRHandle)

Deserialisieren eines serialisierten MLP-basierten OCR-Klassifikators.

`deserialize_ocr_class_mlp` deserialisiert einen MLP-basierten OCR-Klassifikator, welcher mit `serialize_ocr_class_mlp` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Der serialisierte OCR-Klassifikator wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten OCR-Klassifikator mit dem Handle `OCRHandle` gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **OCRHandle** (output_control) ..... ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_ocr_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_ocr_class_mlp

Nachfolger

do_ocr_single_class_mlp, do_ocr_multi_class_mlp

Siehe auch

create_ocr_class_mlp, write_ocr_class_mlp, read_class_mlp, write_class_mlp, serialize_ocr_class_mlp

Modul

OCR/OCV

do_ocr_multi_class_mlp (Character, Image : : OCRHandle : Class, Confidence)

Klassifikation mehrerer Zeichen mit einem OCR-Klassifikator.

do_ocr_multi_class_mlp berechnet mit dem durch OCRHandle gegebenen OCR-Klassifikator zu den durch die Regionen Character und die Grauwerte Image gegebenen Zeichen die jeweils beste Klasse und gibt die Klassen in Class und die zugehörigen Konfidenzen (Wahrscheinlichkeiten) der Klassen in Confidence zurück. Im Gegensatz zu do_ocr_single_class_mlp kann do_ocr_multi_class_mlp also mehrere Zeichen gleichzeitig klassifizieren und ist damit typischerweise schneller als eine Schleife, die do_ocr_single_class_mlp zur Klassifikation einzelner Zeichen verwendet. Allerdings kann do_ocr_multi_class_mlp nur die beste Klasse des jeweiligen Zeichens zurückliefern. Da sich die Konfidenzen als Wahrscheinlichkeiten interpretieren lassen (siehe classify_class_mlp und evaluate_class_mlp), und damit auch leicht überprüfbar ist, ob ein Zeichen zu unsicher klassifiziert worden ist, ist dies im Normalfall kein Nachteil, außer die Klassen überlappen sich so stark, dass in vielen Fällen noch die zweitbeste Klasse untersucht werden muss, um sicher zu entscheiden, welche Klasse vorliegt. In diesem Fall sollte do_ocr_single_class_mlp verwendet werden.

Eine Zeichenkette des Zahlenwertes '\032' (alternativ als '\0x1A' dargestellt) im Resultat Class bedeutet, dass die Region als Rückweisungsklasse klassifiziert wurde.

Der OCR-Klassifikator muss vor der Verwendung von do_ocr_multi_class_mlp mit trainf_ocr_class_mlp trainiert werden.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte der Zeichen.
- ▷ **OCRHandle** (input_control) ..... ocr_mlp ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Class** (output_control) ..... string(-array) ⇝ *string*
  Ergebnis der Klassifikation der Zeichen durch das MLP.
  Parameteranzahl: Class == Character
- ▷ **Confidence** (output_control) ..... real(-array) ⇝ *real*
  Konfidenz der Klasse der Zeichen.
  Parameteranzahl: Confidence == Character

Ergebnis

Sind die Parameterwerte korrekt, dann liefert do_ocr_multi_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

trainf_ocr_class_mlp, read_ocr_class_mlp

Alternativen

do_ocr_word_mlp, do_ocr_single_class_mlp

Siehe auch

create_ocr_class_mlp, classify_class_mlp

Modul

OCR/OCV

do_ocr_single_class_mlp (Character, Image : : OCRHandle, Num : Class, Confidence)

Klassifikation eines Zeichens mit einem OCR-Klassifikator.

do_ocr_single_class_mlp berechnet mit dem durch OCRHandle gegebenen OCR-Klassifikator zu dem durch die Region Character und die Grauwerte Image gegebenen Zeichen die Num besten Klassen und gibt die Klassen in Class und die zugehörigen Konfidenzen (Wahrscheinlichkeiten) der Klassen in Confidence zurück. Da potentiell mehrere Klassen zurückgeliefert werden können, darf Character nur eine Region (ein Zeichen) enthalten. Falls mehrere Zeichen in einem Aufruf klassifiziert werden sollen, muss do_ocr_multi_class_mlp verwendet werden. Da do_ocr_multi_class_mlp bei der Klassifikation mehrerer Zeichen typischerweise schneller ist als eine Schleife mit do_ocr_single_class_mlp und da sich die Konfidenzen als Wahrscheinlichkeiten interpretieren lassen (siehe classify_class_mlp und evaluate_class_mlp), und damit auch leicht überprüfbar ist, ob ein Zeichen zu unsicher klassifiziert worden ist, empfiehlt sich in den meisten Fällen die Verwendung von do_ocr_multi_class_mlp, außer die zweitbeste Klasse soll explizit untersucht werden.

Eine Zeichenkette des Zahlenwertes '$\backslash 032$' (alternativ als '$\backslash 0x1A$' dargestellt) im Resultat Class bedeutet, dass die Region als Rückweisungsklasse klassifiziert wurde.

Der OCR-Klassifikator muss vor der Verwendung von do_ocr_single_class_mlp mit trainf_ocr_class_mlp trainiert werden.

Parameter

- ▷ **Character** (input_object) . . . . . region ⇝ *object*
  Zu erkennendes Zeichen.
- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte des Zeichens.
- ▷ **OCRHandle** (input_control) . . . . . ocr_mlp ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Num** (input_control) . . . . . integer-array ⇝ *integer*
  Anzahl der zu bestimmenden besten Klassen.
  Defaultwert: 1
  Wertevorschläge: Num ∈ {1, 2, 3, 4, 5}
- ▷ **Class** (output_control) . . . . . string(-array) ⇝ *string*
  Ergebnis der Klassifikation des Zeichens durch das MLP.
- ▷ **Confidence** (output_control) . . . . . real(-array) ⇝ *real*
  Konfidenz(en) der Klasse(n) des Zeichens.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert do_ocr_single_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_mlp, read_ocr_class_mlp

Alternativen

do_ocr_multi_class_mlp

Siehe auch

create_ocr_class_mlp, classify_class_mlp

Modul

OCR/OCV

do_ocr_word_mlp (Character, Image : : OCRHandle, Expression, NumAlternatives, NumCorrections : Class, Confidence, Word, Score)

Klassifikation einer Zeichengruppe mit einem OCR-Klassifikator.

do_ocr_word_mlp verhält sich wie do_ocr_multi_class_mlp, indem es mit dem durch OCRHandle gegebenen OCR-Klassifikator zu den durch die Regionen Character und die Grauwerte Image gegebenen Zeichen die jeweils beste Klasse berechnet, und die Klassen in Class und die zugehörigen Konfidenzen (Wahrscheinlichkeiten) der Klassen in Confidence zurückgibt.

Im Gegensatz zu do_ocr_multi_class_mlp behandelt do_ocr_word_mlp die Zeichengruppe als eine Einheit, die durch Aneinanderhängen von Klassennamen für jedes einzelne Zeichen ein Wort Word liefert. Dies ermöglicht die Einschränkung der erlaubten Klassifikationsergebnisse auf Textebene durch Angabe eines Ausdrucks Expression, der das erwartete Wort beschreibt.

Ein Ausdruck kann das Wort auf ein festes Lexikon, welches zuvor durch create_lexicon oder import_lexicon erzeugt wurde, einschränken, indem der Name des Lexikons in spitzen Klammern wie in *'<meinlexikon>'* angegeben wird. Ist der Ausdruck nicht von dieser Form, so wird er als regulärer Ausdruck mit derselben Syntax wie für tuple_regexp_match interpretiert. Bei Verwendung von Quantifikatoren wie '*' sollte man normalerweise mit einem Ausdruck der Form '^...*$*'' dafür sorgen, dass das gesamte Wort verwendet wird.

Im Gegensatz zu tuple_regexp_match erlaubt do_ocr_word_mlp **keine** Angabe von zusätzlichen Optionen für Expression in einem Tupel.

Falls das aus den jeweils besten Klassen abgeleitete Wort dem Ausdruck nicht genügt, versucht do_ocr_word_mlp eine Korrektur durch Berücksichtigung der NumAlternatives besten Klassen für jedes Zeichen, so wie sie von do_ocr_single_class_mlp für ein einzelnes Zeichen bestimmt werden. Hierzu werden alle möglichen Korrekturen ausprobiert, bei denen die Klassifikation von höchstens NumCorrections Zeichenregionen geändert wird. Zu beachten ist hierbei, dass NumAlternatives und NumCorrections Auswirkungen auf die Komplexität des Algorithmus haben und daher in manchen Fällen intern beschränkt werden müssen. Für weitere Informationen siehe den Abschnitt 'Komplexität' weiter unten.

Falls der Ausdruck ein Lexikon beschreibt und die obige Prozedur nicht zum Erfolg führte, so wird das ähnlichste Wort aus dem Lexikon zurückgegeben, solange für die Korrektur weniger als NumCorrections Editieroperationen benötigt werden (siehe suggest_lexicon).

Das Ergebnis wird mit einem Score zwischen 0.0 (keine Korrektur gefunden) und 1.0 (ursprüngliches Wort war schon korrekt) bewertet. Verringert wird der Score durch das Anbringen einer Strafe entsprechend der Anzahl korrigierter Zeichen und einer (kleineren) Strafe für das Ignorieren von Klassen mit höherer Konfidenz um einen Ausdruck Expression zu erhalten:

$$\mathtt{Score} = 1.0 - \mathtt{PenaltyCorrections} - \mathtt{PenaltyAlternatives}$$

$$\begin{aligned} \mathtt{PenaltyCorrections} &= \alpha * \mathtt{num_corr} \\ \mathtt{PenaltyAlternatives} &= \alpha * \beta * \mathtt{num_alt} \end{aligned}$$

wobei num_corr die Anzahl tatsächlich angebrachter Korrekturen und num_alt die Anzahl ignorierter Alternativen ist.

$$\alpha = \frac{1}{\texttt{NumCorrections} + 1}$$
$$\beta = \frac{1}{\texttt{NumCorrections} * (\texttt{NumAlternatives} - 1) + 2}$$

Diese Bewertung ist rein kombinatorisch und berücksichtigt **nicht** die ursprünglichen Konfidenzen der besten Klassen.

Eine Zeichenkette des Zahlenwertes '\032' (alternativ als '\0x1A' dargestellt) im Resultat `Class` bedeutet, dass die Region als Rückweisungsklasse klassifiziert wurde.

Parameter

- ▷ **`Character`** (input_object) ..... region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object* : byte / uint2
  Grauwerte der Zeichen.
- ▷ **`OCRHandle`** (input_control) ..... ocr_mlp ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **`Expression`** (input_control) ..... string ⇝ *string*
  Ausdruck, der die erlaubte Wortstruktur beschreibt
- ▷ **`NumAlternatives`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Klassen pro Zeichen, die für die interne Wortkorrektur berücksichtigt werden.
  Defaultwert: 3
  Wertevorschläge: `NumAlternatives` ∈ {3, 4, 5}
  Typischer Wertebereich: $1 \leq$ NumAlternatives
- ▷ **`NumCorrections`** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl der korrigierten Zeichen.
  Defaultwert: 2
  Wertevorschläge: `NumCorrections` ∈ {1, 2, 3, 4, 5}
  Typischer Wertebereich: $0 \leq$ NumCorrections
- ▷ **`Class`** (output_control) ..... string(-array) ⇝ *string*
  Ergebnis der Klassifikation der Zeichen durch das MLP.
  Parameteranzahl: Class == Character
- ▷ **`Confidence`** (output_control) ..... real(-array) ⇝ *real*
  Konfidenz der Klasse der Zeichen.
  Parameteranzahl: Confidence == Character
- ▷ **`Word`** (output_control) ..... string ⇝ *string*
  Text des Wortes nach Klassifikation und Korrektur.
- ▷ **`Score`** (output_control) ..... real ⇝ *real*
  Ähnlichkeitsmaß von korrigiertem Wort zum Original.

Komplexität

Die Komplexität des Prüfens aller möglichen Korrekturen hat die Größenordnung $O((n * a)^{min(c,n)})$, wobei a die Anzahl der Alternativen, n die Anzahl der Zeichenregionen und c die Anzahl der erlaubten Korrekturen ist. Als Absicherung gegen quasi-unendliche Schleifen im Falle von großem n wird der Wert von c intern beschränkt auf 5, 3, oder 1, falls $a * n >= 30$, 60, bzw. 90.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `do_ocr_word_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_mlp, read_ocr_class_mlp

Alternativen

do_ocr_multi_class_mlp

Siehe auch

create_ocr_class_mlp, classify_class_mlp

Modul

OCR/OCV

get_features_ocr_class_mlp (Character : : OCRHandle, Transform : Features)

Berechnung der Merkmale eines Zeichens.

get_features_ocr_class_mlp berechnet die Merkmale des Zeichens, das durch Character angegeben wird, mit dem OCR-Klassifikator OCRHandle und gibt sie in Features zurück. Im Gegensatz zu do_ocr_single_class_mlp und do_ocr_multi_class_mlp wird das Zeichen als ein Bildobjekt angegeben. Vor der Verwendung von get_features_ocr_class_mlp muss also reduce_domain aufgerufen werden. Der Parameter Transform legt fest, ob die im Klassifikator durch Preprocessing in create_ocr_class_mlp festgelegte Merkmalstransformation angewendet werden soll (Transform = *'true'*) oder ob die untransformierten Merkmale zurückgegeben werden sollen (Transform = *'false'*). get_features_ocr_class_mlp kann dazu verwendet werden, die zur Klassifikation verwendeten Merkmale zu inspizieren.

Parameter

- ▷ **Character** (input_object) ..... singlechannelimage ⇝ *object* : byte / uint2
  Eingabezeichen.
- ▷ **OCRHandle** (input_control) ..... ocr_mlp ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Transform** (input_control) ..... string ⇝ *string*
  Soll der Merkmalsvektor mit der Vorverarbeitung transformiert werden?
  Defaultwert: 'true'
  Werteliste: Transform ∈ {'true', 'false'}
- ▷ **Features** (output_control) ..... real-array ⇝ *real*
  Merkmalsvektor des Zeichens.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_features_ocr_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_mlp

Siehe auch

create_ocr_class_mlp

Modul

OCR/OCV

get_params_ocr_class_mlp (: : OCRHandle : WidthCharacter, HeightCharacter, Interpolation, Features, Characters, NumHidden, Preprocessing, NumComponents)

Auslesen der Parameter eines OCR-Klassifikators.

get_params_ocr_class_mlp gibt die Parameter des durch OCRHandle angegebenen OCR-Klassifikators, die bei der Erzeugung mit create_ocr_class_mlp verwendet wurden, zurück. Dies ist insbesondere nützlich, wenn der Klassifikator mit read_ocr_class_mlp eingelesen wurde. Die Ausgabedaten von get_params_ocr_class_mlp können z.B. verwendet werden, um zu überprüfen, ob ein zu lesendes Zeichen in dem Klassifikator vorhanden ist. Zur Beschreibung der Parameter siehe create_ocr_class_mlp.

Parameter

- ▷ **OCRHandle** (input_control) ..... ocr_mlp ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **WidthCharacter** (output_control) ..... integer ⇝ *integer*
  Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
- ▷ **HeightCharacter** (output_control) ..... integer ⇝ *integer*
  Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
- ▷ **Interpolation** (output_control) ..... string ⇝ *string*
  Interpolationsmodus bei der Skalierung der Zeichen.
- ▷ **Features** (output_control) ..... string(-array) ⇝ *string*
  Merkmale, die zur Klassifikation verwendet werden.
- ▷ **Characters** (output_control) ..... string-array ⇝ *string*
  Zeichen des zu lesenden Zeichensatzes.
- ▷ **NumHidden** (output_control) ..... integer ⇝ *integer*
  Anzahl der versteckten Einheiten des MLP.
- ▷ **Preprocessing** (output_control) ..... string ⇝ *string*
  Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
- ▷ **NumComponents** (output_control) ..... integer ⇝ *integer*
  Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_params_ocr_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_ocr_class_mlp, read_ocr_class_mlp

Nachfolger

do_ocr_single_class_mlp, do_ocr_multi_class_mlp

Siehe auch

trainf_ocr_class_mlp, get_params_class_mlp

Modul

OCR/OCV

get_prep_info_ocr_class_mlp (: : OCRHandle, TrainingFile, Preprocessing : InformationCont, CumInformationCont)

Berechnen des Informationsgehaltes der vorverarbeiteten Merkmalsvektoren eines OCR-Klassifikators.

get_prep_info_ocr_class_mlp berechnet den Informationsgehalt der mit der durch Preprocessing gegebenen Vorverarbeitung transformierten Komponenten der Trainingsvektoren. Preprocessing kann

auf *'principal_components'* oder *'canonical_variates'* gesetzt werden. Der OCR-Klassifikator `OCRHandle` muss mit `create_ocr_class_mlp` erzeugt worden sein. Die zugrundeliegenden Vorverarbeitungen sind bei `create_class_mlp` beschrieben. Der Informationsgehalt wird aus die Variation der transformierten Komponenten der Trainingsvektoren berechnet, d.h. er wird rein aufgrund der Trainingsdaten unabhängig von einer Fehlerrate bei einer Klassifikation der Trainingsdaten berechnet. Der Informationsgehalt wird für alle relevanten Komponenten der transformierten Merkmalsvektoren (`NumInput` für *'principal_components'* und $\min(\texttt{NumOutput}-1, \texttt{NumInput})$ für *'canonical_variates'*, siehe `create_class_mlp`) in `InformationCont` zurückgegeben. Der Informationsgehalt wird als eine Zahl zwischen 0 und 1 dargestellt. Ein prozentualer Informationsgehalt kann leicht durch Multiplikation mit 100 berechnet werden. Der kumulierte Informationsgehalt der ersten n Komponenten wird in der n-ten Komponente von `CumInformationCont` zurückgegeben, d.h. `CumInformationCont` enthält die Summen der ersten n Elemente von `InformationCont`. Um `get_prep_info_ocr_class_mlp` verwenden zu können, müssen Trainingsmuster in den durch `TrainingFile` angegebenen Trainingsdateien abgespeichert werden (siehe `write_ocr_trainf`).

`InformationCont` und `CumInformationCont` können dazu verwendet werden, um zu entscheiden, wie viele Komponenten der transformierten Merkmalsvektoren relevante Information enthalten. Ein oft verwendetes Kriterium ist z.B. zu verlangen, dass die transformierten Daten x% (z.B. 90%) der Daten enthalten sollten. Dies kann leicht anhand des ersten Wertes von `CumInformationCont`, der über x% liegt, bestimmt werden. Der so erhaltene Wert kann bei einem neuerlichen Aufruf von `create_ocr_class_mlp` als `NumComponents` verwendet werden. Da zum Aufruf von `get_prep_info_ocr_class_mlp` schon ein Klassifikator mit `create_ocr_class_mlp` erzeugt werden muss, also auch ein initialer Wert von `NumComponents` bei `create_ocr_class_mlp` angegeben werden muss, aber bei Verwendung von `get_prep_info_ocr_class_mlp` typischerweise noch nicht bekannt ist, wie viele Komponenten relevant sind, empfiehlt sich folgendes zweistufiges Vorgehen, um `NumComponents` zu bestimmen: In einem ersten Schritt wird ein Klassifikator mit der maximalen Anzahl von `NumComponents` (`NumInput` für *'principal_components'* und $\min(\texttt{NumOutput}-1, \texttt{NumInput})$ für *'canonical_variates'*) erzeugt. Dann werden die Trainingsmuster mit `write_ocr_trainf` in einer Datei gespeichert. Anschließend wird mit `get_prep_info_ocr_class_mlp` der Informationsgehalt der Komponenten und somit `NumComponents` bestimmt. Danach wird ein neuer Klassifikator mit der gewünschten Anzahl Komponenten erzeugt und der Klassifikator mit `trainf_ocr_class_mlp` trainiert.

Parameter

- ▷ **`OCRHandle`** (input_control) ..... ocr_mlp ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **`TrainingFile`** (input_control) ..... filename.read(-array) ⇝ *string*
  Name(n) der Trainingsdatei(en).
  Defaultwert: 'ocr.trf'
  Dateiendung: `.trf`, `.otr`
- ▷ **`Preprocessing`** (input_control) ..... string ⇝ *string*
  Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
  Defaultwert: 'principal_components'
  Werteliste: `Preprocessing` ∈ {'principal_components', 'canonical_variates'}
- ▷ **`InformationCont`** (output_control) ..... real-array ⇝ *real*
  Relativer Informationsgehalt der transformierten Merkmalsvektoren.
- ▷ **`CumInformationCont`** (output_control) ..... real-array ⇝ *real*
  Kumulierter Informationsgehalt der transformierten Merkmalsvektoren.

Beispiel

```
* Create the initial OCR classifier.
read_ocr_trainf_names ('ocr.trf', CharacterNames, CharacterCount)
create_ocr_class_mlp (8, 10, 'constant', 'default', CharacterNames, 80, \
                      'canonical_variates', |CharacterNames|, 42, OCRHandle)
* Get the information content of the transformed feature vectors.
get_prep_info_ocr_class_mlp (OCRHandle, 'ocr.trf', 'canonical_variates', \
                             InformationCont, CumInformationCont)
* Determine the number of transformed components.
* NumComp = [...]
* Create the final OCR classifier.
```

```
create_ocr_class_mlp (8, 10, 'constant', 'default', CharacterNames, 80, \
                      'canonical_variates', NumComp, 42, OCRHandle)
* Train the final classifier.
trainf_ocr_class_mlp (OCRHandle, 'ocr.trf', 100, 1, 0.01, Error, ErrorLog)
write_ocr_class_mlp (OCRHandle, 'ocr.omc')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_prep_info_ocr_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls `get_prep_info_ocr_class_mlp` den Fehler 9211 (Matrix ist nicht positiv definit) bei `Preprocessing` = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_ocr_class_mlp`, `write_ocr_trainf`, `append_ocr_trainf`, `write_ocr_trainf_image`

Nachfolger

`clear_ocr_class_mlp`, `create_ocr_class_mlp`

Modul

OCR/OCV

get_regularization_params_ocr_class_mlp (: : OCRHandle, GenParamName : GenParamValue)

Auslesen der Regularisierungsparameter eines OCR-Klassifikators.

`get_regularization_params_ocr_class_mlp` gibt die Regularisierungsparameter eines OCR-Klassifikators, die mit `set_regularization_params_ocr_class_mlp` spezifiziert wurden, zurück. Weiterhin gibt `get_regularization_params_ocr_class_mlp` die Parameter zurück, die durch die automatische Bestimmung der Regularisierungsparameter bestimmt wurden. Zur Beschreibung der Parameter siehe `set_regularization_params_ocr_class_mlp`.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **GenParamName** (input_control) . . . . . string ⇝ *string*
Name des auszulesenden Regularisierungsparameters.
Defaultwert: 'weight_prior'
Werteliste: `GenParamName` ∈ {'weight_prior', 'num_well_determined_params', 'fraction_well_determined_params', 'num_outer_iterations', 'num_inner_iterations'}

▷ **GenParamValue** (output_control) . . . . . number(-array) ⇝ *real* / integer
Wert des Regularisierungsparameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_regularization_params_ocr_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_regularization_params_ocr_class_mlp, read_ocr_class_mlp

Nachfolger

trainf_ocr_class_mlp

Modul

OCR/OCV

get_rejection_params_ocr_class_mlp (: : OCRHandle, GenParamName : GenParamValue)

Auslesen der Parameter der Rückweisungsklasse eines OCR-Klassifikators.

get_rejection_params_ocr_class_mlp gibt die Parameter der Rückweisungsklasse eines OCR-Klassifikators, die mit set_rejection_params_ocr_class_mlp spezifiziert wurden zurück. Zur Beschreibung der Parameter siehe set_rejection_params_ocr_class_mlp und set_rejection_params_class_mlp.

Parameter

▷ **OCRHandle** (input_control) ..... ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **GenParamName** (input_control) ..... string(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'sampling_strategy'
Werteliste: GenParamName ∈ {'sampling_strategy', 'hyperbox_tolerance', 'rejection_sample_factor', 'random_seed', 'rejection_class_index'}

▷ **GenParamValue** (output_control) ..... string(-array) ⇝ *string*
Wert des generischen Parameters.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_rejection_params_ocr_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_ocr_class_mlp, read_ocr_class_mlp

Nachfolger

trainf_ocr_class_mlp

Modul

OCR/OCV

read_ocr_class_mlp (: : FileName : OCRHandle)

Lesen eines OCR-Klassifikators aus Datei.

read_ocr_class_mlp liest einen mit write_ocr_class_mlp abgespeicherten OCR-Klassifikator ein. Da das Trainieren eines OCR-Klassifikators relativ lange dauern kann, wird der Klassifikator typischerweise in einem Offline-Prozess trainiert und mit write_ocr_class_mlp gespeichert. Im Online-Prozess wird der Klassifikator mit read_ocr_class_mlp gelesen und danach zur Klassifikation mit do_ocr_single_class_mlp oder do_ocr_multi_class_mlp verwendet.

HALCON stellt eine Anzahl vortrainierter OCR-Klassifikatoren zur Verfügung (siehe „Solution Guide I", Kapitel 'OCR', Abschnitt 'Pretrained OCR Fonts'). Diese vortrainierten OCR-Klassifikatoren ermöglichen es, eine Vielzahl an Zeichensätzen zu lesen, ohne vorher einen OCR-Klassifikator trainieren zu müssen. Das Training der vortrainierten OCR-Klassifikatoren erfolgte mit Zeichen, die dunkel auf hellem Hintergrund gedruckt waren.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Datei.
Wertevorschläge: FileName ∈ {'Document_0-9_NoRej.omc', 'Document_0-9_Rej.omc', 'Document_0-9A-Z_NoRej.omc', 'Document_0-9A-Z_Rej.omc', 'Document_A-Z+_NoRej.omc', 'Document_A-Z+_Rej.omc', 'Document_NoRej.omc', 'Document_Rej.omc', 'DotPrint_0-9_NoRej.omc', 'DotPrint_0-9_Rej.omc', 'DotPrint_0-9+_NoRej.omc', 'DotPrint_0-9+_Rej.omc', 'DotPrint_0-9A-Z_NoRej.omc', 'DotPrint_0-9A-Z_Rej.omc', 'DotPrint_A-Z+_NoRej.omc', 'DotPrint_A-Z+_Rej.omc', 'DotPrint_NoRej.omc', 'DotPrint_Rej.omc', 'HandWritten_0-9_NoRej.omc', 'HandWritten_0-9_Rej.omc', 'Industrial_0-9_NoRej.omc', 'Industrial_0-9_Rej.omc', 'Industrial_0-9+_NoRej.omc', 'Industrial_0-9+_Rej.omc', 'Industrial_0-9A-Z_NoRej.omc', 'Industrial_0-9A-Z_Rej.omc', 'Industrial_A-Z+_NoRej.omc', 'Industrial_A-Z+_Rej.omc', 'Industrial_NoRej.omc', 'Industrial_Rej.omc', 'OCRA_0-9_NoRej.omc', 'OCRA_0-9_Rej.omc', 'OCRA_0-9A-Z_NoRej.omc', 'OCRA_0-9A-Z_Rej.omc', 'OCRA_A-Z+_NoRej.omc', 'OCRA_A-Z+_Rej.omc', 'OCRA_NoRej.omc', 'OCRA_Rej.omc', 'OCRB_0-9_NoRej.omc', 'OCRB_0-9_Rej.omc', 'OCRB_0-9A-Z_NoRej.omc', 'OCRB_0-9A-Z_Rej.omc', 'OCRB_A-Z+_NoRej.omc', 'OCRB_A-Z+_Rej.omc', 'OCRB_NoRej.omc', 'OCRB_passport_NoRej.omc', 'OCRB_passport_Rej.omc', 'OCRB_Rej.omc', 'Pharma_0-9_NoRej.omc', 'Pharma_0-9_Rej.omc', 'Pharma_0-9+_NoRej.omc', 'Pharma_0-9+_Rej.omc', 'Pharma_0-9A-Z_NoRej.omc', 'Pharma_0-9A-Z_Rej.omc', 'Pharma_NoRej.omc', 'Pharma_Rej.omc', 'SEMI_NoRej.omc', 'SEMI_Rej.omc'}
Dateiendung: .omc, .fnt

▷ **OCRHandle** (output_control) . . . . . ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_ocr_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

do_ocr_single_class_mlp, do_ocr_multi_class_mlp

Siehe auch

create_ocr_class_mlp, write_ocr_class_mlp, read_class_mlp, write_class_mlp

Modul

OCR/OCV

select_feature_set_trainf_mlp (: : TrainingFile, FeatureList, SelectionMethod, Width, Height, GenParamName, GenParamValue : OCRHandle, FeatureSet, Score)

Wählt die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus.

select_feature_set_trainf_mlp wählt die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus. Die Daten für das OCR-Klassifikationsproblem werden mit der Trainingsdatei TrainingFile ausgewählt. Als Klassifikator wird ein mehrschichtiges Perzeptron (MLP) verwendet.

Es wird eine Untermenge aller angegebenen OCR-Merkmale gewählt. Alle zu untersuchenden Merkmale werden in FeatureList spezifiziert. Die Liste der möglichen OCR-Merkmale wird in der Dokumentation von create_ocr_class_mlp näher erläutert. Die letztendlich ausgewählte Merkmalsuntermenge wird in FeatureSet zurückgegeben.

select_feature_set_trainf_mlp ist auf OCR-Klassifikationsprobleme spezialisiert und unterstützt nur Merkmale die in der Auswahlliste stehen. Für andere Merkmale bietet sich somit die Nutzung des generellen Operators select_feature_set_mlp an.

Für den Selektionsprozess können in SelectionMethod zwei verschiedene Methoden ausgewählt werden: entweder die Greedy-Variante *'greedy'* (das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt) oder die dynamisch oszillierende Suche *'greedy_oscillating'* (Das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt. Danach wird getestet ob eines der hinzugefügten Merkmale entbehrlich ist.). Während *'greedy'* schneller terminiert, kann *'greedy_oscillating'* bessere Ergebnisse erzielen wenn viele Trainingsdaten vorliegen. Während *'greedy'* schneller terminiert, kann *'greedy_oscillating'* besser mit voneinander abhängigen Merkmalen umgehen. Als Optimierungskriterium dient die Klassifikationsrate, die mit einem zwei-fachen Kreuzvalidierungsverfahren ermittelt wird. Das beste erreichte Klassifikationsrate wird in Score zurückgegeben.

Über die generischen Parameter GenParamName und GenParamValue kann die Anzahl der verwendeten Neuronen in der versteckten Schicht des MLP über *'num_hidden'* gesetzt werden. Standard ist hierfür *80*, größere Werte führen zu längeren Trainingszeiten, erlauben aber eine ausdrucksstärkere Klassifikation.

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lang sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator select_feature_set_trainf_mlp zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

▷ **TrainingFile** (input_control) ..... filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: ''
Dateiendung: .trf, .otr

▷ **FeatureList** (input_control) ..... string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden können.
Defaultwert: ['zoom_factor','ratio','width','height','foreground','foreground_grid_9','foreground_grid_16', 'anisometry','compactness','convexity','moments_region_2nd_invar','moments_region_2nd_rel_invar', 'moments_region_3rd_invar','moments_central','phi','num_connect','num_holes','projection_horizontal', 'projection_vertical','projection_horizontal_invar','projection_vertical_invar','chord_histo','num_runs', 'pixel','pixel_invar','pixel_binary','gradient_8dir','cooc','moments_gray_plane']
Werteliste: FeatureList ∈ {'default', 'zoom_factor', 'ratio', 'width', 'height', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'anisometry', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'phi', 'num_connect', 'num_holes', 'projection_horizontal', 'projection_vertical', 'projection_horizontal_invar', 'projection_vertical_invar', 'chord_histo', 'num_runs', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'cooc', 'moments_gray_plane'}

▷ **SelectionMethod** (input_control) ..... string ⇝ *string*
Methode zur Auswahl der Merkmale.
Defaultwert: 'greedy'
Werteliste: SelectionMethod ∈ {'greedy', 'greedy_oscillating'}

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 15

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 16

▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
Namen von (optionalen) Parametern für die Steuerung des Verhaltens des verwendeten MLP Klassifikators.
Defaultwert: []
Werteliste: GenParamName ∈ {'nu'}

▷ **GenParamValue** (input_control) . . . . . number-array ⇝ *real* / integer / string
Zu den optionalen generischen Parametern gehörende Werte.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'0.1'}

▷ **OCRHandle** (output_control) . . . . . ocr_mlp ⇝ *handle*
Mit den ausgewählten Merkmalen trainierter OCR-MLP-Klassifikator.

▷ **FeatureSet** (output_control) . . . . . string-array ⇝ *string*
Ausgewählte Merkmale.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Klassifikationsrate, die mit dem ausgewählten Merkmalssatz erreicht wurde.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert select_feature_set_trainf_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Alternativen

select_feature_set_trainf_svm, select_feature_set_trainf_knn

Siehe auch

select_feature_set_trainf_mlp_protected, select_feature_set_mlp

Modul

OCR/OCV

select_feature_set_trainf_mlp_protected (: : TrainingFile, Password, FeatureList, SelectionMethod, Width, Height, GenParamName, GenParamValue : OCRHandle, FeatureSet, Score)

Wählt aus (geschützten) Trainingsdateien die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus.

select_feature_set_trainf_mlp_protected wählt die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus. Die Daten für das OCR-Klassifikationsproblem werden mit der Trainingsdatei TrainingFile ausgewählt. Als Klassifikator wird ein mehrschichtiges Perzeptron (MLP) verwendet.

Die Funktionalität des Operators entspricht der Funktionalität von select_feature_set_trainf_mlp, mit der Erweiterung, dass select_feature_set_trainf_mlp_protected sowohl ungeschützte als auch geschützte Trainingsdateien verarbeiten kann. Geschützte Trainingsdateien können nur mit dem korrekten Benutzerpasswort Password verwendet werden. Falls genau ein Passwort Password übergeben wird, werden alle Eingabedateien TrainingFile mit diesem Passwort überprüft, ansonsten muss die Anzahl der Passworte gleich der Anzahl der Eingabedateien sein, und die Eingabedatei an Position n wird mit dem Passwort an Position n überprüft. Für ungeschützte Trainingsdateien werden die übergebenen Passworte ignoriert.

Für eine genauere Beschreibung der Operatorfunktionalität siehe select_feature_set_trainf_mlp. Das Konzept des Schützens von OCR-Trainingsdaten in HALCON ist in protect_ocr_trainf beschrieben.

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lang sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator

select_feature_set_trainf_mlp_protected zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

▷ **TrainingFile** (input_control) . . . . . filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: ''
Dateiendung: .trf, .otr

▷ **Password** (input_control) . . . . . string(-array) ⇝ *string*
Passworte für geschützte Trainingsdateien.

▷ **FeatureList** (input_control) . . . . . string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden können.
Defaultwert: ['zoom_factor','ratio','width','height','foreground','foreground_grid_9','foreground_grid_16', 'anisometry','compactness','convexity','moments_region_2nd_invar','moments_region_2nd_rel_invar', 'moments_region_3rd_invar','moments_central','phi','num_connect','num_holes','projection_horizontal', 'projection_vertical','projection_horizontal_invar','projection_vertical_invar','chord_histo','num_runs', 'pixel','pixel_invar','pixel_binary','gradient_8dir','cooc','moments_gray_plane']
Werteliste: FeatureList ∈ {'default', 'zoom_factor', 'ratio', 'width', 'height', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'anisometry', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'phi', 'num_connect', 'num_holes', 'projection_horizontal', 'projection_vertical', 'projection_horizontal_invar', 'projection_vertical_invar', 'chord_histo', 'num_runs', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'cooc', 'moments_gray_plane'}

▷ **SelectionMethod** (input_control) . . . . . string ⇝ *string*
Methode zur Auswahl der Merkmale.
Defaultwert: 'greedy'
Werteliste: SelectionMethod ∈ {'greedy', 'greedy_oscillating'}

▷ **Width** (input_control) . . . . . integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 15

▷ **Height** (input_control) . . . . . integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 16

▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
Namen von (optionalen) Parametern für die Steuerung des Verhaltens des verwendeten MLP Klassifikators.
Defaultwert: []
Werteliste: GenParamName ∈ {'nu'}

▷ **GenParamValue** (input_control) . . . . . number-array ⇝ *real* / integer / string
Zu den optionalen generischen Parametern gehörende Werte.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'0.1'}

▷ **OCRHandle** (output_control) . . . . . ocr_mlp ⇝ *handle*
Mit den ausgewählten Merkmalen trainierter OCR-MLP-Klassifikator.

▷ **FeatureSet** (output_control) . . . . . string-array ⇝ *string*
Ausgewählte Merkmale.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Klassifikationsrate, die mit dem ausgewählten Merkmalssatz erreicht wurde.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert select_feature_set_trainf_mlp_protected den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

protect_ocr_trainf

Alternativen

select_feature_set_trainf_svm_protected

Siehe auch

select_feature_set_trainf_mlp, select_feature_set_mlp

Modul

OCR/OCV

serialize_ocr_class_mlp (: : OCRHandle : SerializedItemHandle)

Serialisieren eines MLP-basierten OCR-Klassifikators.

serialize_ocr_class_mlp serialisiert einen MLP-basierten OCR-Klassifikator (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_ocr_class_mlp in eine Datei geschrieben werden. Der OCR-Klassifikator wird in dem Handle OCRHandle übergeben. Der serialisierte OCR-Klassifikator wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_ocr_class_mlp wieder deserialisiert werden.

Parameter

- ▷ **OCRHandle** (input_control) ..... ocr_mlp ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_ocr_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_mlp

Nachfolger

clear_ocr_class_mlp, fwrite_serialized_item, send_serialized_item, deserialize_ocr_class_mlp

Siehe auch

create_ocr_class_mlp, read_ocr_class_mlp, write_class_mlp, read_class_mlp, deserialize_ocr_class_mlp

Modul

OCR/OCV

set_regularization_params_ocr_class_mlp (: : OCRHandle, GenParamName, GenParamValue :)

Setzen der Regularisierungsparameter eines OCR-Klassifikators.

set_regularization_params_ocr_class_mlp setzt die Regularisierungsparameter des OCR-Klassifikators, der in OCRHandle übergeben wurde. Der zu setzende Regularisierungsparameter wird mit GenParamName spezifiziert. Sein Wert wird mit GenParamValue spezifiziert.

Wie bei create_class_mlp beschrieben, kann es wünschenswert sein, den OCR-Klassifikator (d.h. das MLP des Klassifikators) zu regularisieren, um einen glatteren Übergang der Konfidenzen zwischen zwei Klassen zu erzwingen und um eine Überanpassung des OCR-Klassifikators an die Trainingsdaten zu verhindern. Um dies zu erreichen, kann ein Strafterm für große MLP-Gewichte (die den Hauptgrund für scharfe Übergänge zwischen Klassen darstellen) zum Training des MLPs in trainf_ocr_class_mlp hinzugefügt werden. Hierzu muss GenParamName auf *'weight_prior'* und GenParamValue auf einen Wert > 0 gesetzt werden. Weiterhin können die Regularisierungsparameter automatisch bestimmt werden. Die Details hierzu können bei set_regularization_params_class_mlp nachgelesen werden. Falls die Regularisierungsparameter automatisch bestimmt werden sollen, wird dringend empfohlen, die Ratschläge bei set_regularization_params_class_mlp genau zu beachten, wie die Parameter NumHidden des MLPs und *'num_outer_iterations'* zu wählen sind und welche Auswirkungen die automatische Bestimmung der Regularisierungsparameter auf den Speicherverbrauch und die Laufzeit des Trainings des OCR-Klassifikators hat.

Parameter

▷ **OCRHandle** (input_control) ..... ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **GenParamName** (input_control) ..... string ⇝ *string*
Name des zu setzenden Regularisierungsparameters.
Defaultwert: 'weight_prior'
Werteliste: GenParamName ∈ {'weight_prior', 'num_outer_iterations', 'num_inner_iterations'}

▷ **GenParamValue** (input_control) ..... number(-array) ⇝ *real* / integer
Wert des Regularisierungsparameters.
Defaultwert: 1.0
Wertevorschläge: GenParamValue ∈ {0.01, 0.1, 1.0, 10.0, 100.0, 0, 1, 2, 3, 5, 10, 15, 20}

Beispiel

```
* This example shows how to determine the regularization parameters
* automatically without examining the convergence of the
* regularization parameters.
* Create the OCR classifier.
read_ocr_trainf_names ('ocr.trf', CharacterNames, CharacterCount)
create_ocr_class_mlp (8, 10, 'constant', 'default', CharacterNames, \
                      40, 'none', |CharacterNames|, 42, OCRHandle)
* Set up the automatic determination of the regularization
* parameters.
set_regularization_params_ocr_class_mlp (OCRHandle, 'weight_prior', \
                                         [0.01,0.01,0.01,0.01])
set_regularization_params_ocr_class_mlp (OCRHandle, \
                                         'num_outer_iterations', 10)
* Train the classifier.
trainf_ocr_class_mlp (OCRHandle, 'ocr.trf', 100, 1, 0.01, Error, \
                      ErrorLog)
* Read out the estimate of the number of well-determined
* parameters.
get_regularization_params_ocr_class_mlp (OCRHandle, \
                                         'fraction_well_determined_params', \
                                          FractionParams)
* If FractionParams differs substantially from 1, consider reducing
* NumHidden appropriately and consider performing a preprocessing that
* reduces the number of input variables to the net, i.e., canonical
* variates or principal components.
write_ocr_class_mlp (OCRHandle, 'ocr.omc')
```

```
* This example shows how to determine the regularization parameters
* automatically while examining the convergence of the
* regularization parameters.
* Create the OCR classifier.
read_ocr_trainf_names ('ocr.trf', CharacterNames, CharacterCount)
create_ocr_class_mlp (8, 10, 'constant', 'default', CharacterNames, \
                      40, 'none', |CharacterNames|, 42, OCRHandle)
* Set up the automatic determination of the regularization
* parameters.
set_regularization_params_ocr_class_mlp (OCRHandle, 'weight_prior', \
                                         [0.01,0.01,0.01,0.01])
set_regularization_params_ocr_class_mlp (OCRHandle, \
                                         'num_outer_iterations', 1)
for OuterIt := 1 to 10 by 1
    * Train the classifier.
    trainf_ocr_class_mlp (OCRHandle, 'ocr.trf', 100, 1, 0.01, Error, \
                          ErrorLog)
    * Read out the regularization parameters
    get_regularization_params_ocr_class_mlp (OCRHandle, \
                                             'weight_prior', \
                                             WeightPrior)
    * Inspect the regularization parameters manually for
    * convergence and exit the loop manually if they have
    * converged.
    * [...]
endfor
* Read out the estimate of the number of well-determined
* parameters.
get_regularization_params_ocr_class_mlp (OCRHandle, \
                                         'fraction_well_determined_params', \
                                          FractionParams)
* If FractionParams differs substantially from 1, consider reducing
* NumHidden appropriately and consider performing a preprocessing that
* reduces the number of input variables to the net, i.e., canonical
* variates or principal components.
write_ocr_class_mlp (OCRHandle, 'ocr.omc')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_regularization_params_ocr_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ocr_class_mlp`

Nachfolger

`get_regularization_params_ocr_class_mlp`, `trainf_ocr_class_mlp`

Modul

OCR/OCV

set_rejection_params_ocr_class_mlp (: : OCRHandle, GenParamName, GenParamValue :)

Setzen der Parameter der Rückweisungsklasse eines OCR-Klassifikators.

set_rejection_params_ocr_class_mlp setzt die Parameter einer Rückweisungsklasse eines OCR-Klassifikators, der in OCRHandle übergeben wurde. Der zu setzende Parameter wird mit GenParamName spezifiziert. Sein Wert wird mit GenParamValue. Für einige OCR Anwendungen ist es wichtig zu wissen, ob ein Buchstabe ähnlich zu der Trainingsmenge ist oder nicht. Falls der Buchstabe zu der Trainingsmenge verschieden ist kann es als eine Rückweisungsklasse klassifiziert werden. Diese zusätzliche Rückweisungsklasse muss als zusätzlicher Buchstabe in create_ocr_class_mlp spezifiziert werden. Die Details, wie eine Rückweisungsklasse spezifiziert wird können bei set_rejection_params_class_mlp nachgelesen werden.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: 'sampling_strategy'
Werteliste: GenParamName ∈ {'sampling_strategy', 'hyperbox_tolerance', 'rejection_sample_factor', 'random_seed', 'rejection_class_index'}

▷ **GenParamValue** (input_control) . . . . . string(-array) ⇝ *string*
Wert des generischen Parameters.
Defaultwert: 'hyperbox_around_all_classes'
Werteliste: GenParamValue ∈ {'no_rejection_class', 'hyperbox_around_all_classes', 'hyperbox_around_each_class', 'hyperbox_ring_around_each_class'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_rejection_params_ocr_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_ocr_class_mlp, read_ocr_class_mlp

Nachfolger

trainf_ocr_class_mlp

Modul

OCR/OCV

trainf_ocr_class_mlp (: : OCRHandle, TrainingFile, MaxIterations, WeightTolerance, ErrorTolerance : Error, ErrorLog)

Trainieren eines OCR-Klassifikators.

trainf_ocr_class_mlp trainiert den OCR-Klassifikator OCRHandle mit den Trainingszeichen, die in den durch TrainingFile angegebenen OCR-Trainingsdateien enthalten sind. Die Trainingsdateien müssen zuvor, z.B. mit write_ocr_trainf, erzeugt worden sein.

Die restlichen Parameter haben dieselbe Bedeutung wie bei `train_class_mlp` und sind dort ausführlich beschrieben. Eine Regularisierung des OCR-Klassifikators und eine automatische Bestimmung der Regularisierungsparameter (siehe `set_regularization_params_ocr_class_mlp`) wird beim Training berücksichtigt. Weiterhin werden vor dem eigentlichen Training die Trainingsdaten für die Rückweisungsklasse erzeugt, falls eine Rückweisungsklasse mittels `set_rejection_params_ocr_class_mlp` spezifiziert wurde.

Es ist zu beachten, dass Trainingszeichen, die keine entsprechende Klasse im Klassifikator `OCRHandle` haben, verworfen werden.

Parameter

- ▷ **`OCRHandle`** (input_control) . . . . . ocr_mlp ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **`TrainingFile`** (input_control) . . . . . filename.read(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert: 'ocr.trf'
  Dateiendung: `.trf`, `.otr`
- ▷ **`MaxIterations`** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl von Iterationen des Optimierungsverfahrens.
  Defaultwert: 200
  Wertevorschläge: `MaxIterations` ∈ {20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 260, 280, 300}
- ▷ **`WeightTolerance`** (input_control) . . . . . real ⇝ *real*
  Schwellenwert für den Unterschied der Gewichte des MLP zwischen zwei Iterationen des Optimierungsverfahrens.
  Defaultwert: 1.0
  Wertevorschläge: `WeightTolerance` ∈ {1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001}
  Restriktion: WeightTolerance >= 1.0e-8
- ▷ **`ErrorTolerance`** (input_control) . . . . . real ⇝ *real*
  Schwellenwert für den Unterschied des mittleren Fehlers des MLP auf den Trainingsdaten zwischen zwei Iterationen des Optimierungsverfahrens.
  Defaultwert: 0.01
  Wertevorschläge: `ErrorTolerance` ∈ {1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001}
  Restriktion: ErrorTolerance >= 1.0e-8
- ▷ **`Error`** (output_control) . . . . . real ⇝ *real*
  Mittlerer Fehler des MLP auf den Trainingsdaten.
- ▷ **`ErrorLog`** (output_control) . . . . . real-array ⇝ *real*
  Mittlerer Fehler des MLP auf den Trainingsdaten pro Iteration des Optimierungsverfahrens.

Beispiel

```
* Train an OCR classifier
read_ocr_trainf_names ('ocr.trf', CharacterNames, CharacterCount)
create_ocr_class_mlp (8, 10, 'constant', 'default', CharacterNames, 80, \
                      'none', 81, 42, OCRHandle)
trainf_ocr_class_mlp (OCRHandle, 'ocr.trf', 100, 1, 0.01, Error, ErrorLog)
write_ocr_class_mlp (OCRHandle, 'ocr.omc')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `trainf_ocr_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls `trainf_ocr_class_mlp` den Fehler 9211 (Matrix ist nicht positiv definit) bei `Preprocessing` = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind. Für diesen Fall ist es sinnvoll, `Preprocessing` auf *'normalization'* zu setzen. Ansonsten ist es auch möglich, mehr Trainingsmuster hinzuzufügen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ocr_class_mlp`, `write_ocr_trainf`, `append_ocr_trainf`, `write_ocr_trainf_image`, `set_regularization_params_ocr_class_mlp`

Nachfolger

`do_ocr_single_class_mlp`, `do_ocr_multi_class_mlp`, `write_ocr_class_mlp`

Alternativen

`read_ocr_class_mlp`

Siehe auch

`train_class_mlp`

Modul

OCR/OCV

`trainf_ocr_class_mlp_protected` `( : : OCRHandle, TrainingFile, Password, MaxIterations, WeightTolerance, ErrorTolerance : Error, ErrorLog )`

Trainieren eines OCR-Klassifikators mit Daten aus (geschützten) Trainingsdateien.

`trainf_ocr_class_mlp_protected` trainiert den OCR-Klassifikator `OCRHandle` mit den Trainingsdaten, die in den durch `TrainingFile` angegebenen OCR-Trainingsdateien enthalten sind. Die Funktionalität des Operators entspricht der Funktionalität von `trainf_ocr_class_mlp`, mit der Erweiterung, dass `trainf_ocr_class_mlp_protected` sowohl ungeschützte als auch geschützte Trainingsdateien verarbeiten kann. Geschützte Trainingsdateien können nur mit dem korrekten Benutzerpasswort `Password` verwendet werden. Falls genau ein Passwort `Password` übergeben wird, werden alle Eingabedateien `TrainingFile` mit diesem Passwort überprüft, ansonsten muss die Anzahl der Passworte gleich der Anzahl der Eingabedateien sein, und die Eingabedatei an Position n wird mit dem Passwort an Position n überprüft. Für ungeschützte Trainingsdateien werden die übergebenen Passworte ignoriert.

Für eine genauere Beschreibung der Operatorfunktionalität siehe `trainf_ocr_class_mlp`. Das Konzept des Schützens von OCR-Trainingsdaten in HALCON ist in `protect_ocr_trainf` beschrieben.

Parameter

▷ **`OCRHandle`** (input_control) . . . . . ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **`TrainingFile`** (input_control) . . . . . filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: 'ocr.trf'
Dateiendung: `.trf`, `.otr`

▷ **`Password`** (input_control) . . . . . string(-array) ⇝ *string*
Passworte für geschützte Trainingsdateien.

▷ **`MaxIterations`** (input_control) . . . . . integer ⇝ *integer*
Maximale Anzahl von Iterationen des Optimierungsverfahrens.
Defaultwert: 200
Wertevorschläge: `MaxIterations` ∈ {20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 260, 280, 300}

▷ **`WeightTolerance`** (input_control) . . . . . real ⇝ *real*
Schwellenwert für den Unterschied der Gewichte des MLP zwischen zwei Iterationen des Optimierungsverfahrens.
Defaultwert: 1.0
Wertevorschläge: `WeightTolerance` ∈ {1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001}
Restriktion: WeightTolerance >= 1.0e-8

▷ **ErrorTolerance** (input_control) . . . . . real ⇝ *real*
Schwellenwert für den Unterschied des mittleren Fehlers des MLP auf den Trainingsdaten zwischen zwei Iterationen des Optimierungsverfahrens.
Defaultwert: 0.01
Wertevorschläge: ErrorTolerance ∈ {1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001}
Restriktion: ErrorTolerance >= 1.0e-8

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlerer Fehler des MLP auf den Trainingsdaten.

▷ **ErrorLog** (output_control) . . . . . real-array ⇝ *real*
Mittlerer Fehler des MLP auf den Trainingsdaten pro Iteration des Optimierungsverfahrens.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert trainf_ocr_class_mlp_protected den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls trainf_ocr_class_mlp_protected den Fehler 9211 (Matrix ist nicht positiv definit) bei Preprocessing = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind. Für diesen Fall ist es sinnvoll, Preprocessing auf *'normalization'* zu setzen. Ansonsten ist es auch möglich, mehr Trainingsmuster hinzuzufügen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_ocr_class_mlp, write_ocr_trainf, append_ocr_trainf, write_ocr_trainf_image, protect_ocr_trainf

Nachfolger

do_ocr_single_class_mlp, do_ocr_multi_class_mlp, write_ocr_class_mlp

Alternativen

read_ocr_class_mlp

Siehe auch

trainf_ocr_class_mlp, train_class_mlp

Modul

OCR/OCV

write_ocr_class_mlp (: : OCRHandle, FileName :)

Abspeichern eines OCR-Klassifikators in Datei.

write_ocr_class_mlp speichert den OCR-Klassifikator OCRHandle in der Datei FileName ab. Falls keine Datei-Extension in FileName angegeben wird, wird die Standard-Extension *'.omc'* an FileName angehängt. write_ocr_class_mlp wird typischerweise aufgerufen, nachdem der Klassifikator mit trainf_ocr_class_mlp trainiert wurde. Der Klassifikator kann mit read_ocr_class_mlp wieder eingelesen werden.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_mlp ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: .omc

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_ocr_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`trainf_ocr_class_mlp`

Nachfolger

`clear_ocr_class_mlp`

Siehe auch

`create_ocr_class_mlp`, `read_ocr_class_mlp`, `write_class_mlp`, `read_class_mlp`

Modul

OCR/OCV

20.6 Segmentation

clear_text_model (: : TextModel :)

Freigeben eines Textmodells.

`clear_text_model` gibt das Textmodell `TextModel` und allen zugrundeliegenden Speicher frei.

Parameter

▷ **TextModel** (input_control) . . . . . text_model(-array) ⇝ *handle*
Freizugebendes Textmodell.

Ergebnis

`clear_text_model` gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

`create_text_model_reader`

Modul

OCR/OCV

clear_text_result (: : TextResultID :)

Freigeben eines Ergebnisses der Textsegmentierung.

clear_text_result gibt TextResultID und allen zugrundeliegenden Speicher frei.

Parameter

▷ **TextResultID** (input_control) ..... text_result(-array) ⇝ *handle*
Freizugebendes Ergebnis der Textsegmentierung.

Ergebnis

clear_text_result gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextResultID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Modul

Foundation

create_text_model_reader (: : Mode, OCRClassifier : TextModel)

Erzeugen eines Textmodells.

create_text_model_reader erzeugt das neue Textmodell TextModel, welches den Text beschreibt, der von find_text segmentiert werden soll.

Mode bestimmt welche Methode der Textsegmetierung gewählt werden soll. Mögliche Werte für Mode sind *'auto'* und *'manual'*.

Im Allgemeinen sollte Mode auf *'auto'* gesetzt werden, da die automatische Textsegmentierung stabiler ist und weniger Anpassungen notwendig sind. Hierbei ist zu beachten, dass ein OCR Klassifikator OCRClassifier übergeben werden muss. Mode muss nur in den folgenden Fällen auf *'manual'* gesetzt werden:

- Der zu segmentierende Text enthält sehr viele lokale Änderungen der Polarität aufgrund von Reflexionen. Dies ist häufig bei geprägtem Text der Fall.
- Es steht kein geeigneter OCR Klassifikator zur Verfügung (siehe unten).

Für Textmodelle mit Mode = *'auto'*, kann mit find_text automatisch Text jeglicher Größe extrahiert und klassifiziert werden. Es ist aber auch möglich, die Suche auf Buchstaben mit bestimmten Eigenschaften einzuschränken, siehe set_text_model_param. Wenn der zu segmentierende Text Punktdruck-Buchstaben enthält, muss der Textmodell-Parameter *'dot_print'* auf *'true'* gesetzt werden. Für Textmodelle mit Mode = *'auto'* muss ein OCR Klassifikator OCRClassifier übergeben werden, der unter Verwendung eines Convolutional Neural Networks (CNN) oder eines mehrschichtigen Perzeptrons (MLP) erzeugt wurde. Es wird ausdrücklich empfohlen einen CNN-basierten OCR Klassifikator mit Rückweisungsklasse oder einen regularisierten MLP-basierter Klassifikator (siehe set_regularization_params_ocr_class_mlp) zu verwenden, der unbekannte Zeichen zurückweist (siehe set_rejection_params_ocr_class_mlp). Ein geeigneter OCR Klassifikator OCRClassifier kann entweder mit read_ocr_class_cnn oder read_ocr_class_mlp eingelesen, oder mit create_ocr_class_mlp erstellt werden. Außerdem ist es möglich in OCRClassifier einen Pfad zu einem vortrainierten OCR Klassifikator oder einem OCR Klassifikator, der mit write_ocr_class_mlp gespeichert wurde, zu übergeben.

Bei Textmodellen mit Mode = *'manual'* müssen für die Textsegmentierung geeignete Parameter wie die Höhe und Breite von Buchstaben mittels set_text_model_param gesetzt werden. In diesem Fall wird der Wert von OCRClassifier ignoriert.

Parameter des Textmodells werden mit set_text_model_param gesetzt und mit get_text_model_param abgefragt.

Da beim Aufruf von create_text_model_reader und folgenden Operationen für das Textmodell Speicher allokiert wird, sollte das Modell mit clear_text_model freigegeben werden, sobald es nicht mehr benötigt wird.

Parameter

▷ **Mode** (input_control) ..... string ⇝ *string*
Modus des Textmodells.
Defaultwert: 'auto'
Werteliste: Mode ∈ {'manual', 'auto'}

▷ **OCRClassifier** (input_control) ..... string ⇝ *string* / integer
OCR Klassifikator.
Defaultwert: 'Universal_Rej.occ'
Wertevorschläge: OCRClassifier ∈ {'Document_Rej.omc', 'Document_0-9_Rej.omc', 'Document_0-9A-Z_Rej.omc', 'Document_A-Z+_Rej.omc', 'DotPrint_Rej.omc', 'DotPrint_0-9_Rej.omc', 'DotPrint_0-9+_Rej.omc', 'DotPrint_0-9A-Z_Rej.omc', 'DotPrint_A-Z+_Rej.omc', 'HandWritten_0-9_Rej.omc', 'Industrial_Rej.omc', 'Industrial_0-9_Rej.omc', 'Industrial_0-9+_Rej.omc', 'Industrial_0-9A-Z_Rej.omc', 'Industrial_A-Z+_Rej.omc', 'OCRA_Rej.omc', 'OCRA_0-9_Rej.omc', 'OCRA_0-9A-Z_Rej.omc', 'OCRA_A-Z+_Rej.omc', 'OCRB_Rej.omc', 'OCRB_0-9_Rej.omc', 'OCRB_0-9A-Z_Rej.omc', 'OCRB_A-Z+_Rej.omc', 'OCRB_passport_Rej.omc', 'Pharma_Rej.omc', 'Pharma_0-9_Rej.omc', 'Pharma_0-9+_Rej.omc', 'Pharma_0-9A-Z_Rej.omc', 'SEMI_Rej.omc', 'Universal_Rej.occ', 'Universal_0-9_Rej.occ', 'Universal_0-9+_Rej.occ', 'Universal_0-9A-Z_Rej.occ', 'Universal_0-9A-Z+_Rej.occ', 'Universal_A-Z+_Rej.occ'}

▷ **TextModel** (output_control) ..... text_model ⇝ *handle*
Neu erzeugtes Textmodell.

Beispiel

```
read_image (Image, 'numbers_scale')
create_text_model_reader ('auto', 'Document_Rej.omc', TextModel)
* Optionally specify text properties
set_text_model_param (TextModel, 'min_char_height', 20)
find_text (Image, TextModel, TextResultID)
* Return character regions and corresponding classification results
get_text_object (Characters, TextResultID, 'all_lines')
get_text_result (TextResultID, 'class', Class)
```

Ergebnis

create_text_model_reader gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

set_text_model_param, get_text_model_param, find_text

Siehe auch

clear_text_model

Modul

OCR/OCV

find_text (Image : : TextModel : TextResultID)

Segmentieren von Text in einem Bild.

find_text segmentiert Text im Bild Image welcher durch das Textmodell TextModel beschrieben wird und gibt das Ergebnis in TextResultID zurück. Die einzelnen Ergebnisse in TextResultID können mit get_text_result und get_text_object abgefragt werden.

Die Ergebnisse in TextResultID hängen davon ab, welcher Mode bei der Erzeugung von TextModel mit create_text_model_reader gewählt wurde.

Für Textmodelle mit Mode = *'auto'* extrahiert und klassifiziert find_text Text jeglicher Größe im Bild Image. Es ist außerdem möglich, die Suche auf Buchstaben mit besonderen Eigenschaften (minimale/maximale Höhe, Breite, Strichbreite...) einzuschränken, siehe set_text_model_param.

Für die Textsegmentierung mit Textmodellen mit Mode = *'manual'* müssen immer geeignete Parameter wie die Höhe und Breite von Buchstaben mittels set_text_model_param gesetzt werden. Das Eingabebild wird dann vorverarbeitet, so dass Strukturen die möglicherweise Buchstaben sind hervorgehoben werden. Das vorverarbeitete Bild kann anschließend von TextResultID abgefragt werden, falls für TextModel *'persistence'* auf *'true'* gesetzt wurde.

Sowohl für Textmodelle mit Mode = *'auto'*, als auch für diese mit Mode = *'manual'* wendet find_text mehrere Schwellenwerte auf das Eingabebild an und selektiert Buchstabenregionen anhand von regionenbasierten und von grauwertbasierten Merkmalen. Anschließend werden diese Regionen zu Zeilen gruppiert. Jede Zeile wird für sich vervollständigt und überprüft ob sie die Restriktionen in TextModel erfüllt. Für Textmodelle mit Mode = *'auto'* besteht eine Zeile aus mindestens zwei Buchstaben, während für Textmodelle mit Mode = *'manual'* eine Zeile aus mindestens drei Buchstaben besteht. Anschließend werden Satz- und Trennzeichen hinzugefügt, sofern die entsprechenden Parameter via set_text_model_param gesetzt wurden. Enthält TextModel vorgegebenen Zeilenstrukturen (*'text_line_structure'*), so wird getestet, ob die entstandene Zeile zu einer diesen Strukturen passt.

find_text segmentiert Text der ungefähr horizontal in Image ausgerichtet ist. Die Operatoren text_line_orientation und rotate_image können verwendet werden um diese Ausrichtung zu erreichen.

Da beim Aufruf von find_text und folgenden Operationen für das Ergebnis der Segmentierung Speicher allokiert wird, sollte das Segmentierungsergebnis *'TextResultID'* mit clear_text_result freigegeben werden, sobald es nicht mehr benötigt wird.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **TextModel** (input_control) ..... text_model ⇝ *handle*
  Textmodell welches den zu segmentierenden Text beschreibt.
- ▷ **TextResultID** (output_control) ..... text_result ⇝ *handle*
  Ergebnis der Segmentierung.

Beispiel

```
read_image (Image, 'numbers_scale')
create_text_model_reader ('auto', 'Document_Rej.omc', TextModel)
* Optionally specify text properties
set_text_model_param (TextModel, 'min_char_height', 20)
find_text (Image, TextModel, TextResultID)
* Return character regions and corresponding classification results
get_text_object (Characters, TextResultID, 'all_lines')
get_text_result (TextResultID, 'class', Class)
```

Ergebnis

Sind alle Parameter gültig, gibt find_text TRUE zurück. Wenn nötig wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`create_text_model_reader`, `set_text_model_param`, `text_line_orientation`, `text_line_slant`

Nachfolger

`get_text_result`, `get_text_object`

Modul

OCR/OCV

`get_text_model_param` (: : `TextModel`, `GenParamName` : `GenParamValue`)

Abfragen von Parametern eines Textmodells.

`get_text_model_param` fragt Parameter des TextModells `TextModel` ab und gibt deren Werte in `GenParamValue` zurück. Mit Ausnahme von *'ocr_classifier'* können alle gültigen Parameter des Textmodells abgefragt werden.

Siehe `set_text_model_param` für eine ausführliche Beschreibung aller möglichen Werte für `GenParamName`.

Parameter

▷ **`TextModel`** (input_control) . . . . . text_model ⇝ *handle*
Textmodell.

▷ **`GenParamName`** (input_control) . . . . . string ⇝ *string*
Abzufragende Parameter.
Defaultwert: 'min_contrast'
Wertevorschläge: `GenParamName` ∈ {'add_fragments', 'dot_print', 'dot_print_max_dot_gap', 'dot_print_min_char_gap', 'dot_print_tight_char_spacing', 'eliminate_border_blobs', 'max_char_height', 'max_char_width', 'max_stroke_width', 'min_char_height', 'min_char_width', 'min_contrast', 'min_stroke_width', 'num_classes', 'polarity', 'return_punctuation', 'return_separators', 'return_whole_line', 'separate_touching_chars', 'text_line_separators', 'text_line_structure', 'text_line_structure_0', 'text_line_structure_1', 'text_line_structure_2', 'manual_add_fragments', 'manual_base_line_tolerance', 'manual_char_height', 'manual_char_width', 'manual_eliminate_border_blobs', 'manual_eliminate_horizontal_lines', 'manual_fragment_size_min', 'manual_is_dotprint', 'manual_is_imprinted', 'manual_max_line_num', 'manual_persistence', 'manual_polarity', 'manual_return_punctuation', 'manual_return_separators', 'manual_stroke_width', 'manual_text_line_structure', 'manual_text_line_structure_0', 'manual_text_line_structure_1', 'manual_text_line_structure_2', 'manual_uppercase_only'}

▷ **`GenParamValue`** (output_control) . . . . . string(-array) ⇝ *integer* / real / string
Werte der Parameter.

Ergebnis

Sind alle Eingabeparameter korrekt, gibt `get_text_model_param` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`set_text_model_param`

Modul

OCR/OCV

get_text_object (: Characters : TextResultID, ResultName :)

Abfragen von ikonischen Ergebnissen der Textsegmentierung.

get_text_object fragt das ikonische Ergebnis ResultName im von find_text zurückgegebenen TextResultID ab. Welche Werte für ResultName zulässig sind, hängt davon ab, welche Art von Textmodell für die Textsegmentierung verwendet wurde.

Im Folgenden werden zuerst die erlaubten Parameterwerte für Textmodelle mit Mode = *'auto'* und anschließend diese für Textmodelle mit Mode = *'manual'* beschrieben.

Die folgenden Ergebnisse können abgefragt werden.

- **Ergebnisse der Textsegmentierung mit einem Textmodell mit Mode = 'auto'**
  Die Textzeilen sind für jede Polarität getrennt von oben nach unten und von links nach rechts sortiert. Innerhalb einer Zeile sind die Buchstaben von links nach rechts sortiert.

  'all_lines': Gibt die Buchstaben aller segmentierten Zeilen zurück.

  ['line', Zeilenindex]: Gibt die Zeile mit Index *Zeilenindex* zurück. *['line', 0]* gibt zum Beispiel die erste Zeile zurück.

  ['element', Index]: Gibt den Buchstaben an der Position *Index* zurück. *['element', 0]* gibt zum Beispiel den ersten Buchstaben zurück.

- **Ergebnisse der Textsegmentierung mit einem Textmodell mit Mode = 'manual'**

  'manual_all_lines': Gibt die Buchstaben aller segmentierten Zeilen zurück. Die Zeilen sind von oben nach unten und von links nach rechts sortiert. Innerhalb der Zeilen sind die Buchstaben von links nach rechts sortiert.

  ['manual_line',Index]: Gibt die Zeile mit der Nummer *Index* zurück (*['manual_line',0]* gibt zum Beispiel die erste Zeile zurück). Die Buchstaben sind von links nach rechts sortiert.

  Wurde bei einem Textmodell mit Mode = *'manual'*, das benutzt wurde um TextResultID zu erzeugen, *'manual_persistence'* gesetzt, kann zusätzlich der folgende Wert abgefragt werden:

  'manual_compensated_image': Das für die Segmentierung vorverarbeitete Bild.

Parameter

▷ **Characters** (output_object) ..... object(-array) ⇝ *object*
Zurückgegebenes Ergebnis.

▷ **TextResultID** (input_control) ..... text_result ⇝ *handle*
Ergebnis der Textsegmentierung.

▷ **ResultName** (input_control) ..... string(-array) ⇝ *string* / integer
Name des abzufragenden Ergebnisses.
Defaultwert: 'all_lines'
Werteliste: ResultName ∈ {'all_lines', 'element', 'line', 'manual_all_lines', 'manual_line', 'manual_compensated_image'}

Beispiel

```
read_image (Image, 'numbers_scale')
create_text_model_reader ('auto', 'Document_Rej.omc', TextModel)
* Optionally specify text properties
set_text_model_param (TextModel, 'min_char_height', 20)
find_text (Image, TextModel, TextResultID)
* Return character regions and corresponding classification results
get_text_object (Characters, TextResultID, 'all_lines')
get_text_result (TextResultID, 'class', Class)
```

Ergebnis

Sind alle Parameter gültig, gibt `get_text_object` TRUE zurück. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`find_text`

Siehe auch

`get_text_result`

Modul

Foundation

`get_text_result` (: : `TextResultID`, `ResultName` : `ResultValue`)

Abfragen von Ergebnissen der Textsegmentierung.

`get_text_result` fragt das Ergebnis `ResultName` des von `find_text` erhaltenen Ergebnisses `TextResultID` ab. Welche Werte für `ResultName` zulässig sind, hängt davon ab, welche Art von Textmodell für die Textsegmentierung verwendet wurde.

Im Folgenden werden zuerst die erlaubten Parameterwerte für Textmodelle mit `Mode` = *'auto'* und anschließend diese für Textmodelle mit `Mode` = *'manual'* beschrieben.

Die folgenden Ergebnisse können abgefragt werden.

- **Ergebnisse der Textsegmentierung mit einem Textmodell mit Mode = 'auto'**
  Die Textzeilen sind für jede Polarität getrennt von oben nach unten und von links nach rechts sortiert. Innerhalb einer Zeile sind die Buchstaben von links nach rechts sortiert.

  'num_lines': Anzahl der gefundenen Zeilen.

  'num_classes': Anzahl der abgespeicherten besten Klassen für jeden Buchstaben. Je nachdem wie viele Klassen der verwendete Klassifikator enthält, kann dieser Wert kleiner sein als die im Textmodell gesetzte Anzahl an Klassen, siehe `set_text_model_param`.

  'class': Klassifikationsergebnisse aller segmentierten Buchstaben durch den OCR Klassifikator, der im zugehörigen Textmodell enthalten ist. Falls der Klassifikator mit Rückweisungsklasse trainiert wurde, liefert `get_text_result` für Buchstaben, die als Rückweisungsklasse klassifiziert wurden, das zweitbeste Klassifikationsergebnis zurück.

  ['class', n]: Ähnlich wie *'class'* mit dem Unterschied, dass für jeden Buchstaben das Klassifikationsergebnis mit der (n+1)-höchsten Konfidenz zurückgegeben wird. *['class', 0]* gibt zum Beispiel für jeden Buchstaben das Klassifikationsergebniss mit der höchsten Konfidenz zurück. Falls der Klassifikator des Textmodells mit Rückweisungsklasse trainiert wurde, wird auch die Rückweisungsklasse zurückgegeben (im Gegensatz zu *'class'*).

  ['class_line', Zeilenindex]: Klassifikationsergebnisse der Buchstaben in der Zeile mit Index *Zeilenindex* durch den OCR Klassifikator, der im entsprechenden Textmodell enthalten ist. *['class_line', 0]* gibt zum Beispiel die Klassen der Buchstaben der erste Zeile zurück. Falls der Klassifikator mit Rückweisungsklasse trainiert wurde, liefert `get_text_result` für Buchstaben, die als Rückweisungsklasse klassifiziert wurden, das zweitbeste Klassifikationsergebnis zurück.

  ['class_line', Zeilenindex, n]: Ähnlich wie *['class_line', Zeilenindex]* mit dem Unterschied, dass für jeden Buchstaben das Klassifikationsergebnis mit der (n+1)-höchsten Konfidenz zurückgegeben wird. *['class_line', Zeilenindex, 0]* gibt zum Beispiel für jeden Buchstaben in der Zeile mit Index *Zeilenindex* das Klassifikationsergebniss mit der höchsten Konfidenz zurück. Falls der Klassifikator des Textmodells mit Rückweisungsklasse trainiert wurde, wird auch die Rückweisungsklasse zurückgegeben (im Gegensatz zu *['class_line', Zeilenindex]*).

['class_element', Index]: Klassifikationsergebnisse des Buchstabens an Position *Index* durch den OCR Klassifikator, der im zugehörigen Textmodell enthalten ist. *['class_element', 0]* gibt zum Beispiel die *'num_classes'* besten Klassifikationsergebnisse (sortiert nach Konfidenz) des ersten Buchstabens zurück.

'confidence': Konfidenzen der Klassen aller segmentierten Buchstaben, siehe *'class'*.

['confidence', n]: Konfidenzen der Klassen mit der jeweils (n+1)-höchsten Konfidenz, siehe *['class', n]*.

['confidence_line', Zeilenindex]: Konfidenzen der Klassen der Buchstaben in der Zeile mit Index *Zeilenindex*, siehe *['class_line', Zeilenindex]*.

['confidence_line', Zeilenindex, n]: Konfidenzen der Klassen mit der jeweils (n+1)-höchsten Konfidenz in der Zeile mit Index *Zeilenindex*, siehe *['class_line', Zeilenindex, n]*.

['confidence_element', Index]: Konfidenzen der Klassen des Buchstabens an Position *Index*, siehe *['class_element', Index]*.

'polarity': Polarität aller segmentierten Buchstaben.

['polarity_line', Zeilenindex]: Polarität der Buchstaben in der Zeile mit Index *Zeilenindex*. *['polarity_line', 0]* gibt zum Beispiel die Polarität der Buchstaben der erste Zeile zurück.

['polarity_element', Index]: Polarität des Buchstaben an der Position *Index*. *['polarity_element', 0]* gibt zum Beispiel die Polarität des ersten Buchstabens zurück.

- **Ergebnisse der Textsegmentierung mit einem Textmodell mit Mode = 'manual'**

  'manual_num_lines': Anzahl der gefundenen Zeilen.

  Wurde bei einem Textmodell mit `Mode` = *'manual'*, das benutzt wurde um `TextResultID` zu erzeugen, *'manual_persistence'* gesetzt, kann zusätzlich der folgende Wert abgefragt werden:

  'manual_thresholds': Die bei der Segmentierung verwendeten Schwellenwerte.

Parameter

▷ **`TextResultID`** (input_control) . . . . . text_result ⇝ *handle*
Ergebnis der Textsegmentierung.

▷ **`ResultName`** (input_control) . . . . . string(-array) ⇝ *string* / integer
Name des abzufragenden Ergebnisses.
Defaultwert: 'class'
Werteliste: `ResultName` ∈ {'class', 'class_element', 'class_line', 'confidence', 'confidence_element', 'confidence_line', 'polarity', 'polarity_element', 'polarity_line', 'num_classes', 'num_lines', 'manual_num_lines', 'manual_thresholds'}

▷ **`ResultValue`** (output_control) . . . . . string(-array) ⇝ *integer* / real / string
Wert des Ergebnisses.

Beispiel

```
read_image (Image, 'numbers_scale')
create_text_model_reader ('auto', 'Document_Rej.omc', TextModel)
* Optionally specify text properties
set_text_model_param (TextModel, 'min_char_height', 20)
find_text (Image, TextModel, TextResultID)
* Return character regions and corresponding classification results
get_text_object (Characters, TextResultID, 'all_lines')
get_text_result (TextResultID, 'class', Class)
```

Ergebnis

Sind alle Parameter gültig, gibt `get_text_result` TRUE zurück. Wenn nötig wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

find_text

Siehe auch

get_text_object

Modul

Foundation

segment_characters (Region, Image : ImageForeground, RegionForeground : Method, EliminateLines, DotPrint, StrokeWidth, CharWidth, CharHeight, ThresholdOffset, Contrast : UsedThreshold)

Segmentieren von Zeichen in einem gegebenen Bildbereich.

Dieser Operator wird verwendet, um Zeichen aus einer Region im Bild Image zu segmentieren. Region wird lediglich zum Einschränken des Verarbeitungsbereiches verwendet. Die segmentierten Zeichen werden in RegionForeground zurückgeliefert.

Es stehen zwei unterschiedliche Methoden zur Verfügung, um die Zeichen zu detektieren. Beide Methoden gehen davon aus, dass die Zeichen dunkler als der Hintergrund sind. Sollte dies nicht zutreffen, dann sollte das Bild mittels invert_image invertiert werden.

Der Parameter Method bestimmt den Algorithmus der zur Segmentierung des Textes verwendet wird. Die möglichen Werte sind

'local_contrast_best' Diese Methode extrahiert Text, der lokal von seinem Hintergrund abweicht. Aus diesem Grund eignet er sich für inhomogen beleuchtete Bilder. Die Hervorhebung der Texträndern führt zu einer genaueren Bestimmung des Textes. Dies ist vor allem dann nützlich, wenn der Hintergrund sehr texturiert ist. Der Parameter Contrast definiert den minimalen Kontrast, also die minimale Grauwertdifferenz zwischen dem Text und dem Hintergrund. Das Verhalten dieses Segmentierungsverfahrens ist mit dem von var_threshold und dem Setzen von LightDark auf *'dark'* identisch.

'local_auto_shape' Der minimale Kontrast wird automatisch so geschätzt, dass die Zahl der winzigen Regionen reduziert wird. Dieses Verfahren ist besonders für verrauschte Bilder geeignet. Der Parameter ThresholdOffset kann zum Anpassen des Schwellenwerts verwendet werden. Sei $g(x, y)$ der Grauwert des Eingabebildes Image an der Position (x, y), dann wird die Schwellenwertbedingung bestimmt durch: $g(x, y) \leq$ UsedThreshold $+$ ThresholdOffset.

Falls zusätzlich zu den Zeichen horizontale oder vertikale Linien in Richtung der Textzeilen segmentiert werden, so sollte EliminateLines auf *'true'* gesetzt werden. Das Entfernen der Linien wird durch den maximalen Wert von CharWidth und den maximalen Wert von CharHeight beeinflusst. Weitere Informationen findet man in deren Parameterbeschreibungen.

DotPrint: Sollte auf *'true'* gesetzt werden, wenn Dot-Prints gelesen werden sollen, ansonsten auf *'false'*.

StrokeWidth: Spezifiziert die Strichstärke des Textes und wird zur Bestimmung von Filtergrößen für die Zeichendetektion verwendet. Die Filtergrößen werden auch durch die Parameter DotPrint, die durchschnittliche Zeichenbreite CharWidth und die durchschnittliche Zeichenhöhe CharHeight festgelegt.

CharWidth: Dieser Parameter kann durch ein Tupel, welches bis zu drei Werten besitzen kann, gesetzt werden. Der erste Wert ist die durchschnittliche Breite, der zweite ist die minimale und der dritte die maximale Zeichenbreite. Wenn die minimale Zeichenbreite auf -1 gesetzt wird, berechnet der Operator den Wert automatisch in Abhängigkeit von der durchschnittlichen Zeichenbreite. Das gleiche gilt für den Fall, dass der maximale Wert nicht gesetzt wird. Einige Beispiele:

[10] setzt die durchschnittliche Zeichenbreite auf 10, die minimale und maximale Zeichenbreite werden vom Operator berechnet.

[10,-1,20] setzt die durchschnittliche Zeichenbreite auf 10, der minimale Wert wird vom Operator bestimmt und der maximale Wert auf 20 gesetzt.

[10,5,20] setzt die durchschnittliche Zeichenbreite auf 10, der minimale Wert wird auf 5 und der maximale Wert auf 20 gesetzt.

CharHeight: Dieser Parameter kann durch ein Tupel, welches bis zu drei Werten besitzen kann, gesetzt werden. Der erste Wert ist die durchschnittliche Höhe, der zweite ist die minimale und der dritte die maximale Zeichenhöhe.

Wenn die minimale Zeichenhöhe auf -1 gesetzt wird, berechnet der Operator den Wert automatisch in Abhängigkeit von der durchschnittlichen Zeichenhöhe. Das gleiche gilt für den Fall, dass der maximale Wert nicht gesetzt wird. Einige Beispiele:

[10] setzt die durchschnittliche Zeichenhöhe auf 10, die minimale und maximale Zeichenhöhe werden vom Operator berechnet.

[10,-1,20] setzt die durchschnittliche Zeichenhöhe auf 10, der minimale Wert wird vom Operator bestimmt und der maximale Wert auf 20 gesetzt.

[10,5,20] setzt die durchschnittliche Zeichenhöhe auf 10, der minimale Wert wird auf 5 und der maximale Wert auf 20 gesetzt.

`ThresholdOffset`: Hiermit kann der Schwellenwert angepasst werden, der verwendet wird, wenn die Segmentierungsmethode `Method` auf den Wert *'local_auto_shape'* gesetzt wurde.

`Contrast`: Definiert den minimalen Kontrast zwischen dem Text und dem Hintergrund. Dieser Parameter wird verwendet, wenn `Method` auf *'local_contrast_best'* gesetzt wurde.

`UsedThreshold`: Dieser Parameter enthält nach der Ausführung den Schwellenwert, der zur Segmentierung der Zeichen verwendet wurde.

`ImageForeground` enthält das Bild, das intern für die Segmentierung verwendet wurde.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Bildbereich mit den Textzeilen.
- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **`ImageForeground`** (output_object) ..... image(-array) ⇝ *object :* byte / uint2
  Zur Segmentierung verwendetes Bild.
- ▷ **`RegionForeground`** (output_object) ..... singlechannelregion(-array) ⇝ *object*
  Region der Zeichen.
- ▷ **`Method`** (input_control) ..... string ⇝ *string*
  Methode zum Segmentieren der Textzeichen.
  Defaultwert: 'local_auto_shape'
  Werteliste: `Method` ∈ {'local_contrast_best', 'local_auto_shape'}
- ▷ **`EliminateLines`** (input_control) ..... string ⇝ *string*
  Entfernen von horizontalen and vertikalen Linien?
  Defaultwert: 'false'
  Werteliste: `EliminateLines` ∈ {'true', 'false'}
- ▷ **`DotPrint`** (input_control) ..... string ⇝ *string*
  Sollen Dot-Print-Zeichen detektiert werden?
  Defaultwert: 'false'
  Werteliste: `DotPrint` ∈ {'true', 'false'}
- ▷ **`StrokeWidth`** (input_control) ..... string ⇝ *string*
  Strichstärke der Zeichen.
  Defaultwert: 'medium'
  Werteliste: `StrokeWidth` ∈ {'ultra_light', 'light', 'medium', 'bold'}
- ▷ **`CharWidth`** (input_control) ..... integer-array ⇝ *integer*
  Breite eines Zeichens.
  Defaultwert: 25
  Typischer Wertebereich: 1 ≤ CharWidth
  Restriktion: CharWidth >= 1
- ▷ **`CharHeight`** (input_control) ..... integer-array ⇝ *integer*
  Höhe eines Zeichens.
  Defaultwert: 25
  Typischer Wertebereich: 1 ≤ CharHeight
  Restriktion: CharHeight >= 1
- ▷ **`ThresholdOffset`** (input_control) ..... integer ⇝ *integer*
  Wert zur Anpassung der Segmentierung.
  Defaultwert: 0

▷ **Contrast** (input_control) . . . . . integer ⇝ *integer*
Minimale Grauwertdifferenz zwischen Text und Hintergrund.
Defaultwert: 10
Typischer Wertebereich: 1 ≤ Contrast
Restriktion: Contrast >= 1

▷ **UsedThreshold** (output_control) . . . . . integer(-array) ⇝ *integer*
Verwendete Schwellenwert zur Segmentierung der Zeichen.

Beispiel

```
for Index := 1 to 5 by 1
  read_image (Image, 'dot_print_rotated/dot_print_rotated_'+Index$'02d')
  text_line_orientation (Image, Image, 50, rad(-30), rad(30), OrientationAngle)
  rotate_image (Image, ImageRotate, deg(-OrientationAngle), 'constant')
  segment_characters (ImageRotate, ImageRotate, ImageForeground, \
                      RegionForeground, 'local_auto_shape', 'false', 'false', 'medium',
                      25, 25, 0, 10, UsedThreshold)
endfor
```

Ergebnis

Sind die Eingabeparameter korrekt besetzt, dann liefert `segment_characters` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`text_line_orientation`

Nachfolger

`select_characters`, `connection`

Alternativen

`threshold`

Modul

Foundation

select_characters (Region : RegionCharacters : DotPrint, StrokeWidth, CharWidth, CharHeight, Punctuation, DiacriticMarks, PartitionMethod, PartitionLines, FragmentDistance, ConnectFragments, ClutterSizeMax, StopAfter :)

Selektieren von Zeichen aus einer gegebenen Region.

`select_characters` selektiert aus der gegebenen `Region` die Bereiche, die Zeichen darstellen. Hierfür werden Eigenschaften wie `StrokeWidth`, `DotPrint`, die Größe der Zeichen und einige weitere verwendet. Die übergebene `Region` sollte vereinigt sein, da ansonsten jede `Region` einzeln verarbeitet wird. Aus diesem Grund sollte der Operator `connection` nicht vorher aufgerufen werden. Sollten mehrere Textbereiche vorhanden sein, so können diese selbstverständlich ohne vorheriges Zusammenfügen der Regionen behandelt werden. Die übergebene `Region` wird typischerweise vom Operator `segment_characters` geliefert, sie kann jedoch auch von anderen Segmentierungsoperatoren bereitgestellt werden.

Der Prozess zur Auswahl der Zeichen kann in vier Schritte unterteilt werden. Jeder Verarbeitungsschritt wird durch die Parameter `StrokeWidth`, `CharHeight` und `CharWidth` beeinflusst. Wenn kleine Objekte wie Punkte verloren gehen, dann sollte die minimale `CharWidth` und die minimale `CharHeight` angepasst werden. Die Ergebnisse der einzelnen Schritte sind allerdings von manchen Parametern besonders abhängig. Eine nähere Beschreibung folgt. Mit dem Parameter `StopAfter` kann die Verarbeitung nach einem bestimmten Schritt beendet werden.

Der erste Schritt ist *'step1_select_candidates'*. In diesem Schritt werden `CharWidth` und `CharHeight` verwendet, um die Kandidaten auszuwählen. Zusätzlich beeinflusst `ClutterSizeMax` das Ergebnis von diesem Schritt.

Im nächsten Schritt, *'step2_partition_characters'*, nehmen die Parameter `PartitionMethod` und `PartitionLines` Einfluss auf das Ergebnis.

Schritt drei, *'step3_connect_fragments'*, verwendet die Parameter `ConnectFragments` und `DotPrint`. Sollen Dot-Prints gelesen werden und manche Punkte werden nicht zu den Zeichen hinzugefügt, so gibt es zwei Möglichkeiten, dieses Problem zu beheben. Zum einen kann man den Wert des Parameter `FragmentDistance` erhöhen und/oder die `StrokeWidth` herunter setzen.

Im letzten Schritt, *'step4_select_characters'*, beeinflussen die Parameter `DiacriticMarks` und `Punctuation` das Ergebnis.

`RegionCharacters`: Nach dem Ausführen enthält `RegionCharacters` die extrahierten Zeichen von der übergebenen `Region`.

`DotPrint`: Sollte auf *'true'* gesetzt werden, wenn Dot-Prints gelesen werden sollen, ansonsten auf *'false'*.

`StrokeWidth`: Spezifiziert die Strichstärke des Textes und wird zur Bestimmung von Filtergrößen für die Zeichendetektion verwendet. Die Filtergrößen werden auch durch die Parameter `DotPrint`, die durchschnittliche Zeichenbreite `CharWidth` und die durchschnittliche Zeichenhöhe `CharHeight` festgelegt.

`CharWidth`: Dieser Parameter kann durch ein Tupel, welches bis zu drei Werten besitzen kann, gesetzt werden. Der erste Wert ist die durchschnittliche Breite, der zweite ist die minimale und der dritte die maximale Zeichenbreite. Wenn die minimale Zeichenbreite auf -1 gesetzt wird, berechnet der Operator den Wert automatisch in Abhängigkeit von der durchschnittlichen Zeichenbreite. Das gleiche gilt für den Fall, dass der maximale Wert nicht gesetzt wird. Einige Beispiele:

[10] setzt die durchschnittliche Zeichenbreite auf 10, die minimale und maximale Zeichenbreite werden vom Operator berechnet.

[10,-1,20] setzt die durchschnittliche Zeichenbreite auf 10, der minimale Wert wird vom Operator bestimmt und der maximale auf 20 gesetzt.

[10,5,20] setzt die durchschnittliche Zeichenbreite auf 10, der minimale Wert wird auf 5 und der maximale Wert auf 20 gesetzt.

`CharHeight`: Dieser Parameter kann durch ein Tupel, welches bis zu drei Werten besitzen kann, gesetzt werden. Der erste Wert ist die durchschnittliche Höhe, der zweite ist die minimale und der dritte die maximale Zeichenhöhe. Wenn die minimale Zeichenhöhe auf -1 gesetzt wird, berechnet der Operator den Wert automatisch in Abhängigkeit von der durchschnittlichen Zeichenhöhe. Das gleiche gilt für den Fall, dass der maximale Wert nicht gesetzt wird. Einige Beispiele:

[10] setzt die durchschnittliche Zeichenhöhe auf 10, die minimale und maximale Zeichenhöhe werden vom Operator berechnet.

[10,-1,20] setzt die durchschnittliche Zeichenhöhe auf 10, der minimale Wert wird vom Operator bestimmt und der maximale auf 20 gesetzt.

[10,5,20] setzt die durchschnittliche Zeichenhöhe auf 10, der minimale Wert wird auf 5 und der maximale Wert auf 20 gesetzt.

`Punctuation`: Dieser Parameter sollte auf *'true'* gesetzt werden, wenn der Operator Satzzeichen (wie .,:'´") detektieren soll, ansonsten werden diese unterdrückt.

`DiacriticMarks`: Dieser Parameter sollte auf *'true'* gesetzt werden, wenn die Anwendung diakritisches Zeichen (wie â,é,ö) enthält oder auf *'false'* um diese zu unterdrücken.

`PartitionMethod`: Benachbarte Zeichen, welche sehr dicht nebeneinander gedruckt wurden, können teilweise nicht getrennt detektiert werden. Mit diesem Parameter kann die Methode zum Trennen dieser Zeichen festgelegt werden. Die möglichen Werte sind *'none'*, wodurch kein Auftrennen durchgeführt wird. *'fixed_width'* bedeutet, dass die Auftrennung davon ausgeht, dass alle Zeichen gleich breit sind. Falls die Breite der extrahierten Region merklich über der durchschnittlichen Zeichenbreite liegt, wird die Region in Teile getrennt, welche dann die durchschnittliche Zeichenbreite besitzen. Das Aufteilen beginnt an der linken Seite der Region. *'variable_width'* bedeutet, dass die Zeichen an ihrer dünnsten Verbindung getrennt werden. Diese Methode kann für Fonts mit unterschiedlich breiten Zeichen ausgewählt werden, oder falls viele aufeinanderfolgende Zeichen als eines extrahiert werden. Es könnte hilfreich sein, `text_line_slant` und/oder `text_line_orientation` auszuführen, bevor `select_characters` aufgerufen wird.

PartitionLines: Wenn einige Textzeilen oder einige Zeichen von unterschiedlichen Textzeilen verbunden sind, sollte dieser Parameter auf *'true'* gesetzt werden.

FragmentDistance: Dieser Parameter beeinflusst die Verknüpfung von Zeichenfragmenten und Zeichen. Wenn zu viel verbunden wird, sollte der Parameter auf *'narrow'* oder *'medium'* gesetzt werden. Im Fall dass mehr Fragmente verbunden werden sollen, sollte der Parameter auf *'medium'* oder *'wide'* gesetzt werden. Das Verbinden wird außerdem vom Maximum von CharWidth und dem Maximum von CharHeight beeinflusst. Siehe auch ConnectFragments.

ConnectFragments Dieser Parameter sollte auf *'true'* gesetzt werden, wenn die extrahierten Zeichen fragmentiert sind, zum Beispiel wenn ein Zeichen nicht als ein Zeichen detektiert wird, sondern in mehrere Einzelteile zerfällt. Weitere Infos findet man unter FragmentDistance und StopAfter im Schritt *'step3_connect_fragments'*.

ClutterSizeMax: Sollten die segmentierten Zeichen Störungen enthalten, zum Beispiel kleine Regionen neben den Zeichen, sollte dieser Wert erhöht werden. Wenn Teile der Zeichen fehlen, dann sollte dieser Wert verkleinert werden.

StopAfter: Dieser Parameter sollte verwendet werden, wenn `select_characters` nicht das gewünschte Ergebnis liefert. Durch Modifizieren des Parameters wird bestimmt, nach welchem Schritt der Operator die Ausführung beendet und es werden die dazugehörigen Ergebnisse geliefert. Um den Operator nicht abzubrechen sollte StopAfter auf *'completion'* gesetzt werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Region mit Textzeilen, in der die Zeichen segmentiert werden sollen.

▷ **RegionCharacters** (output_object) ..... region(-array) ⇝ *object*
Selektierte Zeichen.

▷ **DotPrint** (input_control) ..... string ⇝ *string*
Sollen Dot-Print-Zeichen detektiert werden?
Defaultwert: 'false'
Werteliste: DotPrint ∈ {'true', 'false'}

▷ **StrokeWidth** (input_control) ..... string ⇝ *string*
Strichstärke der Zeichen.
Defaultwert: 'medium'
Werteliste: StrokeWidth ∈ {'ultra_light', 'light', 'medium', 'bold'}

▷ **CharWidth** (input_control) ..... integer-array ⇝ *integer*
Breite eines Zeichens.
Defaultwert: 25
Typischer Wertebereich: 1 ≤ CharWidth
Restriktion: CharWidth >= 1

▷ **CharHeight** (input_control) ..... integer-array ⇝ *integer*
Höhe eines Zeichens.
Defaultwert: 25
Typischer Wertebereich: 1 ≤ CharHeight
Restriktion: CharHeight >= 1

▷ **Punctuation** (input_control) ..... string ⇝ *string*
Satzzeichen hinzufügen?
Defaultwert: 'false'
Werteliste: Punctuation ∈ {'true', 'false'}

▷ **DiacriticMarks** (input_control) ..... string ⇝ *string*
Sind diakritische Zeichen vorhanden?
Defaultwert: 'false'
Werteliste: DiacriticMarks ∈ {'true', 'false'}

▷ **PartitionMethod** (input_control) ..... string ⇝ *string*
Methode zum Trennen von benachbarten Zeichen.
Defaultwert: 'none'
Werteliste: PartitionMethod ∈ {'none', 'fixed_width', 'variable_width'}

▷ **PartitionLines** (input_control) ..... string ⇝ *string*
Sollen Zeilen getrennt werden?
Defaultwert: 'false'
Werteliste: `PartitionLines` ∈ {'true', 'false'}

▷ **FragmentDistance** (input_control) ..... string ⇝ *string*
Abstand von Fragmenten.
Defaultwert: 'medium'
Werteliste: `FragmentDistance` ∈ {'narrow', 'medium', 'wide'}

▷ **ConnectFragments** (input_control) ..... string ⇝ *string*
Sollen Fragmente zusammengefügt werden?
Defaultwert: 'false'
Werteliste: `ConnectFragments` ∈ {'true', 'false'}

▷ **ClutterSizeMax** (input_control) ..... integer ⇝ *integer*
Maximaler Größe von Störungen.
Defaultwert: 0
Typischer Wertebereich: 0 ≤ ClutterSizeMax
Restriktion: 0 < ClutterSizeMax

▷ **StopAfter** (input_control) ..... string ⇝ *string*
Beende die Ausführung nach diesem Schritt.
Defaultwert: 'completion'
Werteliste: `StopAfter` ∈ {'step1_select_candidates', 'step2_partition_characters', 'step3_connect_fragments', 'step4_select_characters', 'completion'}

Beispiel

```
for Index := 1 to 5 by 1
  read_image (Image, 'dot_print_rotated/dot_print_rotated_'+Index$'02d')
  text_line_orientation (Image, Image, 50, rad(-30), rad(30), OrientationAngle)
  rotate_image (Image, ImageRotate, deg(-OrientationAngle), 'constant')
  segment_characters (ImageRotate, ImageRotate, ImageForeground, \
                      RegionForeground, 'local_auto_shape', 'false', 'false', 'medium',
                      25, 25, 0, 10, UsedThreshold)
  select_characters (RegionForeground, RegionCharacters, 'true', 'ultra_light', \
                     [60,1,100], [60,1,100], 'false', 'false', 'none', 'true', 'wide', 'tr
                     'completion')
endfor
```

Ergebnis

Sind die Eingabeparameter korrekt besetzt, dann liefert `select_characters` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`segment_characters`, `text_line_slant`

Nachfolger

`do_ocr_single_class_mlp`, `do_ocr_multi_class_mlp`

Alternativen

`connection`

Modul

Foundation

set_text_model_param (: : TextModel, GenParamName, GenParamValue :)

Setzen von Parametern eines Textmodells.

set_text_model_param setzt Parameter eines Textmodells. Welche Werte für GenParamName zulässig sind hängt davon ab, welcher Mode bei der Erstellung des Textmodells mittels create_text_model_reader gewählt wurde. Im Folgenden werden zuerst die erlaubten Parameterwerte für Textmodelle mit Mode = *'auto'* und anschließend diese für Textmodelle mit Mode = *'manual'* beschrieben.

Die Parameter und ihre Werte werden in GenParamName und GenParamValue übergeben werden. Die folgenden Parameter können gesetzt werden:

- **Parameter für Textmodelle mit Mode = 'auto'**
  - **Segmentierungseigenschaften**

    'min_contrast': Minimaler Kontrast zwischen Buchstaben und ihrem Hintergrund.
    Werteliste: Integer- oder float-Werte zwischen 1 und 255 für Byte-Bilder und zwischen 1 und 65.535 für uint2-Bilder.
    Defaultwert: 15

    'polarity': *'dark_on_light'* wenn der zu segmentierende Text dunkler ist als sein Hintergrund, *'light_on_dark'* wenn der zu segmentierende Text heller ist als sein Hintergrund und *'both'* falls beide Arten von Text segmentiert werden sollen.
    Werteliste: 'dark_on_light', *'light_on_dark'*, *'both'*
    Defaultwert: 'both'

    'eliminate_border_blobs': *'true'* wenn Regionen, die den Rand des Definitionsbereichs des Bildes berühren verworfen werden sollen, ansonsten *'false'*.
    Werteliste: 'true','false'
    Defaultwert: 'false'

    'add_fragments': *'true'* wenn Fragmente wie der Punkt auf einem 'i' zu den segmentierten Buchstaben hinzugefügt werden sollen. Das kann allerdings auch dazu führen, dass Störungen zu den segmentierten Buchstaben hinzugefügt werden.
    Werteliste: 'true','false'
    Defaultwert: 'true'

    'separate_touching_chars': Bestimmt die Trennung von Paaren oder kleinen Gruppen nebeneinander liegender Zeichen, die als eine einzige Region segmentiert werden. Wird *'standard'* oder *'enhanced'* gewählt, werden solche Regionen erkannt und in zwei oder mehrere einzelne Zeichen aufgeteilt. Während die Methode *'enhanced'* genauere Ergebnisse liefert, ist die Methode *'standard'* weniger komplex und damit schneller. Wird *'separate_touching_chars'* auf *'false'* gesetzt, wird keine Trennung von sich berührenden Zeichen durchgeführt.
    Anmerkung: Für die Methode *'enhanced'* wird die Datei *find_text_support.hotc* aus dem ocr Unterverzeichnis des Hauptverzeichnisses der HALCON-Installation benötigt. Es ist auch möglich, diese Datei im aktuell verwendeten Verzeichnis abzulegen.
    Werteliste: 'false', *'standard'*, *'enhanced'*
    Defaultwert: 'standard'

  - **Buchstabengröße**

    'min_char_height': Minimale Höhe eines Buchstabens in Pixel. Um Buchstaben jeglicher Höhe zu segmentieren, kann der Wert *'auto'* übergeben werden. *'min_char_height'* schränkt die Höhe von Buchstaben ein, jedoch nicht die Höhe von Satz- oder Trennzeichen.
    Werteliste: Integer- oder float-Werte größer gleich 1
    Defaultwert: 'auto'

    'max_char_height': Maximale Höhe eines Buchstabens in Pixel. Um Buchstaben jeglicher Höhe zu segmentieren, kann der Wert *'auto'* übergeben werden. *'max_char_height'* schränkt die Höhe von Buchstaben ein, jedoch nicht die Höhe von Satz- oder Trennzeichen.
    Werteliste: Integer- oder float-Werte größer gleich 1
    Defaultwert: 'auto'

    'min_char_width': Minimale Breite eines Buchstabens in Pixel. Um Buchstaben jeglicher Breite zu segmentieren, kann der Wert *'auto'* übergeben werden. *'min_char_width'* schränkt die Breite von Buchstaben ein, jedoch nicht die Breite von Satz- oder Trennzeichen.
    Werteliste: Integer- oder float-Werte größer gleich 1
    Defaultwert: 'auto'

'max_char_width': Maximale Breite eines Buchstabens in Pixel. Um Buchstaben jeglicher Breite zu segmentieren, kann der Wert *'auto'* übergeben werden. *'max_char_width'* schränkt die Breite von Buchstaben ein, jedoch nicht die Breite von Satz- oder Trennzeichen.
Werteliste: Integer- oder float-Werte größer gleich 1
Defaultwert: 'auto'

'min_stroke_width': Minimale Strichbreite eines Buchstabens in Pixel. Soll die minimale Strichbreite während der Textsegmentierung automatisch geschätzt werden, kann der Wert *'auto'* übergeben werden. *'min_stroke_width'* schränkt die Strichbreite von Buchstaben ein, jedoch nicht die Strichbreite von Satz- oder Trennzeichen.
Werteliste: Integer- oder float-Werte größer gleich 1
Defaultwert: 'auto'

'max_stroke_width': Maximale Strichbreite eines Buchstabens in Pixel. Soll die maximale Strichbreite während der Textsegmentierung automatisch geschätzt werden, kann der Wert *'auto'* übergeben werden. *'max_stroke_width'* schränkt die Strichbreite von Buchstaben ein, jedoch nicht die Strichbreite von Satz- oder Trennzeichen.
Werteliste: Integer- oder float-Werte größer gleich 1
Defaultwert: 'auto'

– **Sonderzeichen**

'return_punctuation': *'true'* wenn kleine Satzzeichen, die nahe an der Grundlinie der Textzeile liegen (wie Punkt oder Komma), zu den segmentierten Buchstaben hinzugefügt werden sollen.
Werteliste: 'true','false'
Defaultwert: 'true'

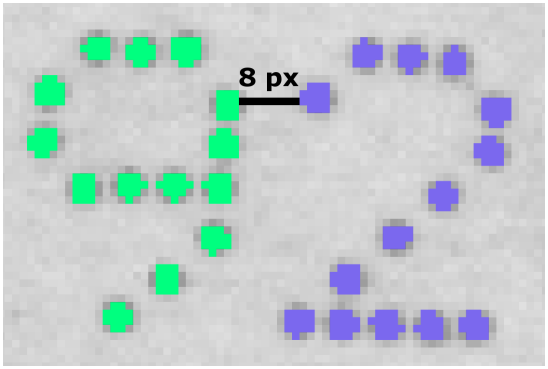
'return_separators': *'true'* wenn Trennzeichen wie ein Minus oder das Gleichheitszeichen zu den segmentierten Buchstaben hinzugefügt werden sollen.
Werteliste: 'true','false'
Defaultwert: 'true'

– **Punktdruck-Buchstaben**

'dot_print': *'true'* wenn der zu segmentierende Text Punktdruck-Buchstaben enthält, ansonsten *'false'*.
Werteliste: 'true','false'
Defaultwert: 'false'

'dot_print_tight_char_spacing': *'true'* wenn die Lücken zwischen benachbarten Buchstaben kleiner sind als die Lücke zwischen zwei Punkten innerhalb eines Buchstabens, ansonsten *'false'*. Falls *'dot_print'* auf *'false'* gesetzt ist, hat dieser Parameter keine Auswirkung. Ist die genaue Größe der kleinsten Lücke zwischen benachbarten Buchstaben bekannt, kann diese stattdessen mittels *'dot_print_min_char_gap'* gesetzt werden. In diesem Fall wird der Wert von *'dot_print_tight_char_spacing'* ignoriert.
Werteliste: 'true','false'
Defaultwert: 'false'

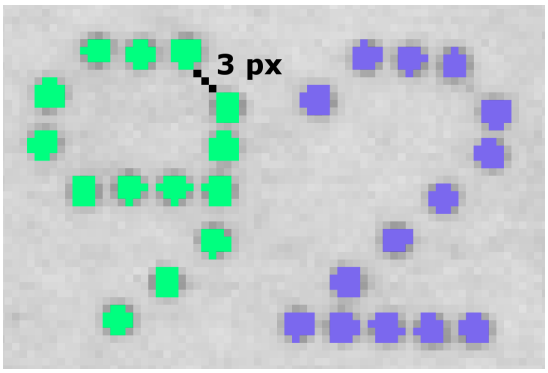
'dot_print_min_char_gap': Größe der kleinsten Lücke zwischen zwei Buchstaben in Pixel. Dieser Parameter kann in Fällen, in denen die Lücken zwischen Buchstaben kleiner sind als die zwischen Punkten innerhalb der Buchstaben, zu einer genaueren Segmentierung führen. Falls die minimale Größe nicht bekannt oder größer als die maximale Lücke zwischen den Punkten innerhalb eines Buchstaben ist, kann der Wert *'auto'* übergeben werden. Falls *'dot_print'* auf *'false'* gesetzt ist, hat dieser Parameter keine Auswirkung. Sind Buchstaben sehr nah nebeneinander gedruckt, jedoch die Größe der minimalen Lücke unbekannt, kann stattdessen *'dot_print_tight_char_spacing'* gesetzt werden.



Hier ist die kleinste Lücke zwischen zwei Buchstaben 8 Pixel.

Werteliste: Integer- oder float-Werte größer gleich 0
Defaultwert: 'auto'

'dot_print_max_dot_gap': Größe der größten Lücke zwischen zwei Punkten innerhalb eines Buchstabens in Pixel. Um beliebige Punktdruck-Buchstaben zu segmentieren kann der Wert *'auto'* übergeben werden. Falls *'dot_print'* auf *'false'* gesetzt ist, hat dieser Parameter keine Auswirkung. Ist die größte Lücke zwischen zwei Punkten größer als die kleinste Lücke zwischen zwei Buchstaben, sollte zusätzlich *'dot_print_tight_char_spacing'* oder *'dot_print_min_char_gap'* entsprechend gesetzt werden. Wird *'dot_print_max_dot_gap'* gesetzt, kann die Laufzeit von `find_text` erheblich reduziert werden.



Hier ist die größte Lücke zwischen zwei Punkten 3 Pixel.

Werteliste: Integer- oder float-Werte größer gleich 1
Defaultwert: 'auto'

– **Struktur der Textzeilen**

'text_line_structure': legt die Struktur des gesuchten Textes fest. Für jede Zeile wird der Abstand zwischen aufeinanderfolgenden Buchstaben berechnet. Ausgehend von diesen Abständen wird die Zeile in einzelne Buchstabenblöcke aufgeteilt. Hierbei werden sehr kleine Zeichen wie z.B. '.', '_' und '-' vernachlässigt und als Leerzeichen interpretiert. Darüber hinaus können zusätzliche Trennzeichen angegeben werde, die ebenfalls vernachlässigt werden sollen, siehe *'text_line_separators'*. Für die entstandenen Buchstabenblöcke wird dann geprüft ob sie einer der übergebenen Strukturen entsprechen.
Eine Struktur besteht aus einem String, der die Anzahl der Buchstaben/Ziffern pro Buchstabenblock enthält, welche durch Leerzeichen getrennt sind. Soll zum Beispiel ein Datum mit jeweils zwei Ziffern für Tag, Monat und Jahr gefunden werden, wäre die Struktur '2 2 2'. Kann das Jahr auch aus vier Ziffern bestehen, wäre die Struktur '2 2 2-4', so dass der letzte Block aus zwei bis vier Buchstaben/Ziffern bestehen kann. Es ist möglich mehr als eine Struktur zu setzen, indem ein Index an den Parameter angehängt wird, z.B. *'text_line_structure_0'*, *'text_line_structure_1'*. Wenn *'text_line_structure'* auf einen leeren String ' ' gesetzt ist, kann der zu findende Texte aus jeder beliebigen Struktur bestehen.
Jede gefundene Struktur wird als einzelne Zeile gespeichert. Folglich bezieht sich dann 'line' in `get_text_object` auf die gefundenen Strukturen. Soll stattdessen die komplette Zeile zurückgeliefert werden, die eine Struktur enthält, kann *'return_whole_line'* auf *'true'* gesetzt werden.
Defaultwert: ' '

'text_line_separators': String von Trennzeichen, die bei der Suche nach Strukturen vernachlässigt werden sollen, siehe *'text_line_structure'*. Hierbei ist zu beachten, dass ein solches Trennzeichen immer auch im Klassifikator enthalten sein muss. Sollen z.B. die Zeichen *':'* und '\' als Trennzeichen behandelt werden, kann der String *':\\'* übergeben werden. Hierbei ist zu beachten, dass das vorangestellte '\' die Verwendung des Symbols '\' als Literal ermöglicht.
Werteliste: '/',*':'*, *':\\'*, *':\\/:'*,...
Defaultwert: ' '

'return_whole_line': *'true'* falls nur die gefundenen Strukturen als Zeilen zurückgegeben werden sollen, *'false'* falls jede Zeile, die eine passende Stuktur enthält komplett zurückgegeben werden soll.
Werteliste: 'true',*'false'*
Defaultwert: 'false'

– **OCR Klassifikator**

'ocr_classifier': OCR Klassifikator, der für die Textsegmentierung und Klassifikation verwendet

wird. Ein initialer Klassifikator wird bereits bei der Erstellung des Textmodells gesetzt. Siehe create_text_model_reader für weitere Informationen.

'num_classes': Anzahl der zu bestimmenden besten Klassen für jeden Buchstaben. (Falls zum Beispiel *'num_classes'* auf 2 gesetzt wird, speichert find_text für jeden Buchstaben die Klasse mit der höchsten und der zweithöchsten erreichten Konfidenz ab). Die entsprechenden Klassen können anschließend mit get_text_result abgefragt werden. Falls der Wert *'num_classes'* die Anzahl der Klassen des im Textmodell gespeicherten Klassifikators überschreitet, wird *'num_classes'* intern entsprechend reduziert. Die tatsächlich verwendete Anzahl von Klassen kann mittels get_text_result abgefragt werden. Für Klassifikatoren mit Rückweisungsklasse sollte *'num_classes'* mindestens 2 sein, um auf das zweitbeste Klassifikationsergebnis zurückgreifen zu können, falls ein Buchstabe als Rückweisungsklasse erkannt wurde.
Werteliste: Integer größer oder gleich 1.
Defaultwert: 2

- **Parameter für Textmodelle mit Mode = 'manual'**

'manual_char_height': Höhe eines Buchstabens in Pixel. Dieser Wert bezieht sich auf einen Großbuchstaben. Standardwert: 30px

'manual_char_width': Breite eines Buchstabens in Pixel. Dieser Wert bezieht sich auf einen Großbuchstaben. Standardwert: 20px

'manual_stroke_width': Strichbreite eines Buchstabens in Pixel. Standardwert: 4.0px

'manual_base_line_tolerance': Maximale Abweichung der Buchstaben von der Grundlinie einer Zeile in Prozent der Buchstabenhöhe. Standardwert: 0.15

'manual_polarity': *'dark_on_light'* wenn der zu segmentierende Text dunkler ist als sein Hintergrund, ansonsten *'light_on_dark'*. Standardwert: *'dark_on_light'*

'manual_uppercase_only': *'true'* wenn der zu segmentierende Text nur Großbuchstaben oder Zahlen enthält, ansonsten *'false'*. Standardwert: *'false'*

'manual_is_dotprint': *'true'* wenn der zu segmentierende Text ein Punktdruck ist, ansonsten *'false'*. Standardwert: *'false'*

'manual_is_imprinted': *'true'* wenn der zu segmentierende Text sehr viele lokale Änderungen der Polarität aufgrund von Reflexionen enthält, ansonsten *'false'*. Standardwert: *'false'*

'manual_eliminate_horizontal_lines': *'true'* wenn sich Linien in der Nähe des zu segmentierenden Textes befinden, ansonsten *'false'*. Standardwert: *'false'*

'manual_eliminate_border_blobs': *'true'* wenn Regionen, die den Rand der Bilddomäne berühren verworfen werden sollen, ansonsten *'false'*. Standardwert: *'false'*

'manual_max_line_num': Maximale Anzahl zu findender Zeilen. Null oder ein negativer Wert impliziert keine Begrenzung. Wird *'manual_max_line_num'* auf einen kleinen Wert gesetzt, kann dies einen positiven Einfluss auf die Laufzeit von find_text haben. Standardwert: ohne Begrenzung

'manual_return_punctuation': *'true'* wenn Interpunktionszeichen (wie Punkt oder Komma) zu den segmentierten Buchstaben hinzugefügt werden sollen. Standardwert: *'true'*

'manual_return_separators': *'true'* wenn Trennzeichen wie ein Minus oder das Gleichheitszeichen zu den segmentierten Buchstaben hinzugefügt werden sollen. Standardwert: *'true'*

'manual_add_fragments': *'true'* wenn Fragmente wie der Punkt auf einem 'i' zu den segmentierten Buchstaben hinzugefügt werden sollen. Das kann auch dazu führen, dass Störungen zu den segmentierten Buchstaben hinzugefügt werden. Standardwert: *'true'*

'manual_fragment_size_min': minimale Größe von Fragmenten die hinzugefügt werden, sofern *'manual_add_fragments'* auf *'true'* gesetzt ist. Standardwert: 1

'manual_text_line_structure': legt die Struktur des zu findenden Textes fest um den Suchraum zu verkleinern und falsche Treffer zu vermeiden. Die Struktur besteht aus einem String, der die Anzahl der Buchstaben/Ziffern pro Buchstabenblock enthält, welche durch Leerzeichen getrennt sind. Soll zum Beispiel ein Datum mit jeweils zwei Ziffern für Tag, Monat und Jahr gefunden werden, wäre die Struktur '2 2 2'. Kann das Jahr auch aus vier Ziffern bestehen, wäre die Struktur '2 2 2-4', so dass der letzte Block aus zwei bis vier Buchstaben/Ziffern bestehen kann. Es ist möglich mehr als eine Struktur zu setzen, indem ein Index an den Parameter angehängt wird, z.B. *'manual_text_line_structure_0'*, *'manual_text_line_structure_1'*. Wenn *'manual_text_line_structure'* auf einen leeren String *' '* gesetzt ist, kann der zu findende Texte aus jeder beliebigen Struktur bestehen.

'manual_persistence': *'true'* wenn ausgewählte Zwischenergebnisse in der Ausgabe von find_text vorgehalten werden sollen, damit sie später mittels get_text_result oder get_text_object abgefragt werden können.

Parameter

▷ **TextModel** (input_control) . . . . . text_model ⇝ *handle*
Textmodell.

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Zu setzende Parameter.
Defaultwert: 'min_contrast'
Wertevorschläge: GenParamName ∈ {'add_fragments', 'dot_print', 'dot_print_max_dot_gap', 'dot_print_min_char_gap', 'dot_print_tight_char_spacing', 'eliminate_border_blobs', 'max_char_height', 'max_char_width', 'max_stroke_width', 'min_char_height', 'min_char_width', 'min_contrast', 'min_stroke_width', 'num_classes', 'ocr_classifier', 'polarity', 'return_punctuation', 'return_separators', 'return_whole_line', 'separate_touching_chars', 'text_line_separators', 'text_line_structure', 'text_line_structure_0', 'text_line_structure_1', 'text_line_structure_2', 'manual_add_fragments', 'manual_base_line_tolerance', 'manual_char_height', 'manual_char_width', 'manual_eliminate_border_blobs', 'manual_eliminate_horizontal_lines', 'manual_fragment_size_min', 'manual_is_dotprint', 'manual_is_imprinted', 'manual_max_line_num', 'manual_persistence', 'manual_polarity', 'manual_return_punctuation', 'manual_return_separators', 'manual_stroke_width', 'manual_text_line_structure', 'manual_text_line_structure_0', 'manual_text_line_structure_1', 'manual_text_line_structure_2', 'manual_uppercase_only'}

▷ **GenParamValue** (input_control) . . . . . string(-array) ⇝ *integer* / real / string
Werte der Parameter.
Defaultwert: 10
Wertevorschläge: GenParamValue ∈ {'true', 'false', 'dark_on_light', 'light_on_dark', 'both', 'auto', 'standard', 'enhanced'}

Beispiel

```
read_image (Image, 'numbers_scale')
create_text_model_reader ('auto', 'Document_Rej.omc', TextModel)
* Optionally specify text properties
set_text_model_param (TextModel, 'min_char_height', 20)
find_text (Image, TextModel, TextResultID)
* Return character regions and corresponding classification results
get_text_object (Characters, TextResultID, 'all_lines')
get_text_result (TextResultID, 'class', Class)
```

Ergebnis

Sind alle Eingabeparameter korrekt, gibt set_text_model_param den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- TextModel

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_text_model_reader

Nachfolger

find_text

Siehe auch

get_text_model_param

Modul

OCR/OCV

text_line_orientation (Region, Image : : CharHeight, OrientationFrom, OrientationTo : OrientationAngle)

Bestimmen der Orientierung einer Textzeile oder eines Textabschnittes.

text_line_orientation bestimmt die Orientierung einer Textzeile oder eines Textabschnitts in Bezug auf die Bildhorizontale.

Der Parameter Region gibt an, in welchem Bereich des Bildes die Textzeilen liegen. Er dient lediglich zum Einschränken des Verarbeitungsbereichs. Innerhalb dieser Region werden die Grauwerte zur Bestimmung der Orientierung verwendet. Die Textzeilen werden vom Operator selbst segmentiert. Wird mehr als eine Region übergeben, dann werden die Zahlenwerte des Orientierungswinkels in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

CharHeight gibt die ungefähre Höhe der Textzeilen im Bereich Region an. Hierbei wird angenommen, dass die Textzeilen dunkler sind als der Hintergrund.

Durch OrientationFrom und OrientationTo kann der Suchbereich eingeschränkt werden, womit auch die Laufzeit von text_line_orientation beinflusst wird.

Mit Hilfe des berechneten Winkels OrientationAngle und Operatoren wie affine_trans_image kann der Bereich Region des Bildes Image so gedreht werden, dass die Textzeilen wieder horizontal im Bild erscheinen. Dies vereinfacht möglicherweise die Zeichensegmentierung für OCR-Anwendungen.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Bildbereich der Textzeilen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **CharHeight** (input_control) ..... integer ⇝ *integer*
  Höhe der Textzeilen.
  Defaultwert: 25
  Typischer Wertebereich: 1 ≤ CharHeight
  Restriktion: CharHeight >= 1
- ▷ **OrientationFrom** (input_control) ..... angle.rad ⇝ *real*
  Minimale Rotation der Textzeilen.
  Defaultwert: -0.523599
  Typischer Wertebereich: -3.141593 ≤ OrientationFrom ≤ 3.141593
  Restriktion: -2 * pi <= OrientationFrom <= 2 * pi
- ▷ **OrientationTo** (input_control) ..... angle.rad ⇝ *real*
  Maximale Rotation der Textzeilen.
  Defaultwert: 0.523599
  Typischer Wertebereich: -3.141593 ≤ OrientationTo ≤ 3.141593
  Restriktion: OrientationFrom <= OrientationTo && OrientationTo < OrientationFrom + pi
- ▷ **OrientationAngle** (output_control) ..... angle.rad(-array) ⇝ *real*
  Berechnete Rotationswinkel der Textzeilen.

Beispiel

```
read_image(Image,'letters')
text_line_orientation(Image,Image,50,rad(-80),rad(80),OrientationAngle)
rotate_image(Image,ImageRotate,-OrientationAngle/rad(180)*180,'constant')
```

Ergebnis

Sind die Eingabeparameter korrekt besetzt, dann liefert text_line_orientation den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`rotate_image`, `affine_trans_image`, `affine_trans_image_size`

Modul

Foundation

`text_line_slant` (`Region`, `Image` : : `CharHeight`, `SlantFrom`, `SlantTo` : `SlantAngle`)

Bestimmen der Neigung von Zeichen einer Textzeile oder eines Textabschnitts.

`text_line_slant` bestimmt die Neigung der Zeichen einer Textzeile oder eines Textabschnitts.

Der Parameter `Region` gibt an, in welchem Bereich des Bildes die Textzeilen liegen. Sie dient lediglich zum Einschränken des Verarbeitungsbereichs. Innerhalb dieser Region werden die Grauwerte zur Bestimmung des Neigungswinkels verwendet. Die Textzeilen werden vom Operator selbst segmentiert. Wird mehr als eine Region übergeben, dann werden die Zahlenwerte des Neigungswinkels in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

`CharHeight` gibt die ungefähre Höhe der Textzeilen im Bereich `Region` an. Hierbei wird angenommen, dass die Textzeilen dunkler sind als der Hintergrund.

Durch `SlantFrom` und `SlantTo` kann der Suchbereich eingeschränkt werden, womit auch die Laufzeit von `text_line_slant` beeinflusst wird.

Mit Hilfe des berechneten Winkels `SlantAngle` und den Operatoren zur affinen Transformation kann die Neigung der Zeichen entfernt werden. Dies vereinfacht möglicherweise die Trennung von Zeichen für OCR-Anwendungen. Damit das Verfahren funktioniert, müssen alle Zeichen einer Region eine ähnliche Neigung besitzen.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Bildbereich der Textzeilen.
- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object* : byte / uint2
  Eingabebild.
- ▷ **`CharHeight`** (input_control) ..... integer ⇝ *integer*
  Höhe der Textzeilen.
  Defaultwert: 25
  Typischer Wertebereich: 1 ≤ CharHeight
  Restriktion: CharHeight >= 1
- ▷ **`SlantFrom`** (input_control) ..... angle.rad ⇝ *real*
  Minimale Neigung der Zeichen
  Defaultwert: -0.523599
  Typischer Wertebereich: -0.785398 ≤ `SlantFrom` ≤ 0.785398
  Restriktion: - pi / 4 <= SlantFrom && SlantFrom <= SlantTo
- ▷ **`SlantTo`** (input_control) ..... angle.rad ⇝ *real*
  Maximale Neigung der Zeichen
  Defaultwert: 0.523599
  Typischer Wertebereich: -0.785398 ≤ `SlantTo` ≤ 0.785398
  Restriktion: - pi / 4 <= SlantTo && SlantTo <= pi / 4
- ▷ **`SlantAngle`** (output_control) ..... angle.rad(-array) ⇝ *real*
  Berechnete Neigungswinkel der Zeichen in der Region

Beispiel

```
hom_mat2d_identity(HomMat2DIdentity)
read_image(Image,'dot_print_slanted')
* correct slant
text_line_slant(Image,Image,50,rad(-45),rad(45),SlantAngle)
hom_mat2d_slant(HomMat2DIdentity,-SlantAngle,'x',0,0,HomMat2DSlant)
affine_trans_image(Image,Image,HomMat2DSlant,'constant','true')
```

Ergebnis

Sind die Eingabeparameter korrekt besetzt, dann liefert `text_line_slant` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`hom_mat2d_slant`, `affine_trans_image`, `affine_trans_image_size`

Modul

Foundation

20.7 Support-Vektor-Maschinen

`clear_ocr_class_svm` (: : OCRHandle :)

Löschen eines auf SVM basierenden OCR-Klassifikators.

`clear_ocr_class_svm` löscht den durch `OCRHandle` angegebenen OCR-Klassifikator und gibt sämtlichen für den Klassifikator benötigten Speicher frei. Der Klassifikator kann nach dem Aufruf von `clear_ocr_class_svm` nicht mehr verwendet werden. Der Handle `OCRHandle` ist nach dem Aufruf ungültig.

Parameter

▷ **`OCRHandle`** (input_control) ..... ocr_svm(-array) ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Falls `OCRHandle` korrekt ist, liefert `clear_ocr_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`do_ocr_single_class_svm`, `do_ocr_multi_class_svm`

Siehe auch

`create_ocr_class_svm`, `read_ocr_class_svm`, `write_ocr_class_svm`, `trainf_ocr_class_svm`

Modul

OCR/OCV

create_ocr_class_svm (: : WidthCharacter, HeightCharacter, Interpolation, Features, Characters, KernelType, KernelParam, Nu, Mode, Preprocessing, NumComponents : OCRHandle)

Erzeugen eines OCR-Klassifikators unter Verwendung einer Support-Vektor-Maschine.

create_ocr_class_svm erzeugt einen OCR-Klassifikator unter Verwendung einer Support-Vektor-Maschine (SVM). Der Handle des OCR-Klassifikators wird in OCRHandle zurückgegeben.

Für eine Beschreibung der Funktionsweise einer SVM siehe create_class_svm. create_ocr_class_svm erzeugt eine SVM für die Klassifikation mit dem Modus Mode. Die Länge des Merkmalsvektors der SVM (NumFeatures in create_class_svm) wird aus den für die OCR zu verwendenden Merkmalen, die in Features übergeben werden, festgelegt. Die Merkmale sind unten beschrieben. Die Kernelfunktion und ihre Parametrisierung wird in KernelType, KernelParam und Nu analog zu create_class_svm angegeben. Die Anzahl der Klassen der SVM (NumClasses in create_class_svm) wird aus den für die OCR zu verwendenden Namen der Zeichen, die in Characters übergeben werden, festgelegt. Wie bei create_class_svm beschrieben, kann mit Preprocessing und NumComponents eine Vorverarbeitung der Daten (Merkmalsvektoren) festgelegt werden. Um die numerische Stabilität zu erhöhen, soll hier im Normalfall *'normalization'* angegeben werden. Um die Klassifikationszeit zu beschleunigen, eignet sich insbesondere *'principal_components'*, da die Anzahl der Eingabefeatures in den Klassifikator deutlich reduziert werden kann, ohne dass sich die Erkennungsrate verschlechtert.

Die zur Klassifikation zu verwendenden Merkmale werden in Features angegeben. Dabei kann Features ein Tupel mit mehreren Merkmalsnamen beinhalten. Jeder dieser Merkmalsnamen führt zur Berechnung von einem oder mehreren Merkmalen für den Klassifikator. Einige der Merkmalsnamen erzeugen Grauwertmerkmale (z.B. *'pixel_invar'*). Da ein Klassifikator immer eine konstante Anzahl von Merkmalen (Eingabevariablen) benötigt, wird ein zu klassifizierendes Zeichen auf eine Standardgröße transformiert, die durch WidthCharacter und HeightCharacter festgelegt wird. Die bei der Transformation anzuwendende Interpolation wird mit Interpolation festgelegt. Sie hat dieselbe Bedeutung wie bei affine_trans_image. Die Interpolation sollte so gewählt werden, dass bei der Transformation keine Aliasing-Effekte auftreten. Im Normalfall sollte Interpolation = *'constant'* verwendet werden. Bei der Wahl der Größe des transformierten Zeichens ist zu beachten, dass die Größe nicht zu groß gewählt wird, da die Generalisierungseigenschaften des Klassifikators sonst sehr schlecht werden können. Insbesondere können bei großen Größen kleine Segmentierungsfehler einen großen Einfluss auf die berechneten Merkmale haben, wenn Grauwertmerkmale verwendet werden, da sich durch die Segmentierungsfehler das umschließende Rechteck des Zeichens ändern kann, was dazu führt, dass das Zeichen anders als trainiert skaliert wird. Im Normalfall sollten Größen zwischen 6×8 und 10×14 verwendet werden.

Mit Features können die folgenden Merkmalsnamen zur Erkennung der Zeichen ausgewählt werden.

'default' *'ratio'* und *'pixel_invar'* werden ausgewählt.

'pixel' Grauwerte des Zeichens (WidthCharacter × HeightCharacter Merkmale).

'pixel_invar' Grauwerte des Zeichens mit automatischer maximaler Grauwertspreizung (WidthCharacter × HeightCharacter Merkmale).

'pixel_binary' Region des Zeichens als Binärbild auf eine Größe von WidthCharacter × HeightCharacter skaliert. (WidthCharacter × HeightCharacter Merkmale).

'gradient_8dir' Die Gradienten des Zeichens werden berechnet. Die Gradientenrichtungen werden in 8 Richtungen diskretisiert. Das Amplitudenbild wird entsprechend der diskretisierten Richtungen in 8 Kanäle aufgeteilt. In jedem der 8 Kanäle werden auf einem 5×5 Gitter Stichproben entnommen. Diese Stichproben bilden die Merkmale (200 Merkmale).

'projection_horizontal' Horizontale Projektion der Grauwerte (siehe gray_projections, HeightCharacter Merkmale).

'projection_horizontal_invar' Horizontale Projektion der Grauwerte, die maximal gespreizt sind (HeightCharacter Merkmale).

'projection_vertical' Vertikale Projektion der Grauwerte (siehe gray_projections, WidthCharacter Merkmale).

'projection_vertical_invar' Vertikale Projektion der Grauwerte, die maximal gespreizt sind (WidthCharacter Merkmale).

'ratio' Seitenverhältnis des Zeichens (siehe height_width_ratio, 1 Merkmal).

'anisometry' Formmerkmal Anisometrie (siehe eccentricity, 1 Merkmal).

'width' Breite des Zeichens vor der Normierung auf die Standardgröße (nicht skalierungsinvariant, siehe `height_width_ratio`, 1 Merkmal).

'height' Höhe des Zeichens vor der Normierung auf die Standardgröße (nicht skalierungsinvariant, siehe `height_width_ratio`, 1 Merkmal).

'zoom_factor' Größenunterschied zwischen aktuellem Zeichen und den Werten von `WidthCharacter` und `HeightCharacter` (nicht skalierungsinvariant, 1 Merkmal).

'foreground' Relativer Anteil der Vordergrund-Pixel (1 Merkmal).

'foreground_grid_9' Relativer Anteil der Vordergrund-Pixel in einem 3×3 Raster innerhalb des umschließenden Rechtecks des Zeichens (9 Merkmale).

'foreground_grid_16' Relativer Anteil der Vordergrund-Pixel in einem 4×4 Raster innerhalb des umschließenden Rechtecks des Zeichens (16 Merkmale).

'compactness' Formmerkmal Kompaktheit (siehe `compactness`, 1 Merkmal).

'convexity' Formmerkmal Konvexität (siehe `convexity`, 1 Merkmal).

'moments_region_2nd_invar' Normierte zweite geometrische Momente der Region (siehe `moments_region_2nd_invar`, 3 Merkmale).

'moments_region_2nd_rel_invar' Normierte zweite relative geometrische Momente der Region (siehe `moments_region_2nd_rel_invar`, 2 Merkmale).

'moments_region_3rd_invar' Normierte dritte geometrische Momente der Region (siehe `moments_region_3rd_invar`, 4 Merkmale).

'moments_central' Normierte zentrale geometrische Momente der Region (siehe `moments_region_central`, 4 Merkmale).

'moments_gray_plane' Normierte Grauwertmomente und die Winkel der Grauwertebene (siehe `moments_gray_plane`, 4 Merkmale)

'phi' Orientierung (Winkel) des Zeichens (siehe `elliptic_axis`, 1 Merkmal).

'num_connect' Anzahl der Zusammenhangskomponenten (siehe `connect_and_holes`, 1 Merkmal).

'num_holes' Anzahl der Hohlflächen (siehe `connect_and_holes`, 1 Merkmal).

'cooc' Werte der binären Cooccurrence-Matrix (siehe `gen_cooc_matrix`, 12 Merkmale).

'num_runs' Anzahl der Sehnen der Region normiert auf die Höhe (1 Merkmal).

'chord_histo' Häufigkeit der Sehnen pro Zeile (nicht skalierungsinvariant, `HeightCharacter` Merkmale).

Nachdem der Klassifikator erzeugt wurde, wird er mit `trainf_ocr_class_svm` trainiert. Daraufhin kann der Klassifikator mit `write_ocr_class_svm` abgespeichert werden. Alternativ können auch sofort nach dem Training Zeichen mit `do_ocr_single_class_svm` oder `do_ocr_multi_class_svm` klassifiziert werden.

Ein Vergleich zwischen SVM und mehrschichtigem Perzeptron (engl.: multilayer perceptron, MLP; siehe `create_ocr_class_mlp`) zeigt typischerweise, dass SVMs generell schneller trainiert werden, insbesondere bei großen Trainingsdatensätzen, und eine leicht verbesserte Erkennungsrate haben. Das MLP weist schnellere Klassifikationszeiten auf und sollte daher in zeitkritischen Anwendungen verwendet werden. Es ist zu beachten, dass der Vergleich von optimal abgestimmten Parametern ausgeht.

Parameter

▷ **`WidthCharacter`** (input_control) ..... integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 8
Wertevorschläge: `WidthCharacter` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich: $4 \leq$ `WidthCharacter` ≤ 20

▷ **`HeightCharacter`** (input_control) ..... integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 10
Wertevorschläge: `HeightCharacter` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich: $4 \leq$ `HeightCharacter` ≤ 20

▷ **`Interpolation`** (input_control) ..... string ⇝ *string*
Interpolationsmodus bei der Skalierung der Zeichen.
Defaultwert: 'constant'
Werteliste: `Interpolation` ∈ {'nearest_neighbor', 'bilinear', 'bicubic', 'constant', 'weighted'}

▷ **Features** (input_control) . . . . . string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden.
Defaultwert: 'default'
Werteliste: Features ∈ {'default', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'projection_horizontal', 'projection_horizontal_invar', 'projection_vertical', 'projection_vertical_invar', 'ratio', 'anisometry', 'width', 'height', 'zoom_factor', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'moments_gray_plane', 'phi', 'num_connect', 'num_holes', 'cooc', 'num_runs', 'chord_histo'}

▷ **Characters** (input_control) . . . . . string-array ⇝ *string*
Alle Zeichen des zu lesenden Zeichensatzes.
Defaultwert: ['0','1','2','3','4','5','6','7','8','9']

▷ **KernelType** (input_control) . . . . . string ⇝ *string*
Art des Kernels.
Defaultwert: 'rbf'
Werteliste: KernelType ∈ {'linear', 'rbf', 'polynomial_inhomogeneous', 'polynomial_homogeneous'}

▷ **KernelParam** (input_control) . . . . . real ⇝ *real*
Zusätzlicher Parameter für die Kernelfunktion.
Defaultwert: 0.02
Wertevorschläge: KernelParam ∈ {0.01, 0.02, 0.05, 0.1, 0.5}

▷ **Nu** (input_control) . . . . . real ⇝ *real*
Regularisierungskonstante der SVM.
Defaultwert: 0.05
Wertevorschläge: Nu ∈ {0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3}
Restriktion: Nu > 0.0 && Nu < 1.0

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Modus der SVM.
Defaultwert: 'one-versus-one'
Werteliste: Mode ∈ {'one-versus-all', 'one-versus-one'}

▷ **Preprocessing** (input_control) . . . . . string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
Defaultwert: 'normalization'
Werteliste: Preprocessing ∈ {'none', 'normalization', 'principal_components', 'canonical_variates'}

▷ **NumComponents** (input_control) . . . . . integer ⇝ *integer*
Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale (ignoriert bei Preprocessing = *'none'* und Preprocessing = *'normalization'*).
Defaultwert: 10
Wertevorschläge: NumComponents ∈ {1, 2, 3, 4, 5, 8, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Restriktion: NumComponents >= 1

▷ **OCRHandle** (output_control) . . . . . ocr_svm ⇝ *handle*
Handle des OCR-Klassifikators.

Beispiel

```
read_image (Image, 'letters')
* Segment the image.
binary_threshold(Image,&Region, 'otsu', 'dark', &UsedThreshold);
dilation_circle (Region, RegionDilation, 3.5)
connection (RegionDilation, ConnectedRegions)
intersection (ConnectedRegions, Region, RegionIntersection)
sort_region (RegionIntersection, Characters, 'character', 'true', 'row')
* Generate the training file.
count_obj (Characters, Number)
Classes := []
for J := 0 to 25 by 1
    Classes := [Classes,gen_tuple_const(20,chr(ord('a')+J))]
endfor
Classes := [Classes,gen_tuple_const(20,'.')]
```

```
write_ocr_trainf (Characters, Image, Classes, 'letters.trf')
* Generate and train the classifier.
read_ocr_trainf_names ('letters.trf', CharacterNames, CharacterCount)
create_ocr_class_svm (8, 10, 'constant', 'default', CharacterNames, \
                      'rbf', 0.01, 0.01, 'one-versus-all', \
                      'principal_components', 10, OCRHandle)
trainf_ocr_class_svm (OCRHandle, 'letters.trf', 0.001, 'default')
* Re-classify the characters in the image.
do_ocr_multi_class_svm (Characters, Image, OCRHandle, Class)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_ocr_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`trainf_ocr_class_svm`

Alternativen

`create_ocr_class_mlp`

Siehe auch

`do_ocr_single_class_svm`, `do_ocr_multi_class_svm`, `clear_ocr_class_svm`, `create_class_svm`, `train_class_svm`, `classify_class_svm`

Modul

OCR/OCV

deserialize_ocr_class_svm (: : SerializedItemHandle : OCRHandle)

Deserialisieren eines serialisierten SVM-basierten OCR-Klassifikators.

`deserialize_ocr_class_svm` deserialisiert einen SVM-basierten OCR-Klassifikator, welcher mit `serialize_ocr_class_svm` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Der serialisierte OCR-Klassifikator wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einem dafür automatisch erzeugten OCR-Klassifikator mit dem Handle `OCRHandle` gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **OCRHandle** (output_control) ..... ocr_svm ⇝ *handle*
Handle des OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_ocr_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_ocr_class_svm`

Nachfolger

`do_ocr_single_class_svm`, `do_ocr_multi_class_svm`

Siehe auch

`create_ocr_class_svm`, `write_ocr_class_svm`, `read_class_svm`, `write_class_svm`, `serialize_ocr_class_svm`

Modul

OCR/OCV

do_ocr_multi_class_svm (Character, Image : : OCRHandle : Class)

Klassifikation mehrerer Zeichen mit einem auf SVM basierenden OCR-Klassifikator.

`do_ocr_multi_class_svm` berechnet mit dem durch `OCRHandle` gegebenen OCR-Klassifikator zu den durch die Regionen `Character` und die Grauwerte `Image` gegebenen Zeichen die jeweils beste Klasse und gibt die Klassen in `Class` zurück. Im Gegensatz zu `do_ocr_single_class_svm` kann `do_ocr_multi_class_svm` also mehrere Zeichen gleichzeitig klassifizieren und ist damit typischerweise schneller als eine Schleife, die `do_ocr_single_class_svm` zur Klassifikation einzelner Zeichen verwendet. Allerdings kann `do_ocr_multi_class_svm` nur die beste Klasse des jeweiligen Zeichens zurückliefern. Der OCR-Klassifikator muss vor der Verwendung von `do_ocr_multi_class_svm` mit `trainf_ocr_class_svm` trainiert werden.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte der Zeichen.
- ▷ **OCRHandle** (input_control) ..... ocr_svm ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Class** (output_control) ..... string(-array) ⇝ *string*
  Ergebnis der Klassifikation der Zeichen durch die SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `do_ocr_multi_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`trainf_ocr_class_svm`, `read_ocr_class_svm`

Alternativen

`do_ocr_single_class_svm`

Siehe auch

`create_ocr_class_svm`, `classify_class_svm`

Modul

OCR/OCV

do_ocr_single_class_svm (Character, Image : : OCRHandle, Num : Class)

Klassifikation eines Zeichens mit einem auf SVM basierenden OCR-Klassifikator.

do_ocr_single_class_svm berechnet mit dem durch OCRHandle gegebenen auf SVM basierendem OCR-Klassifikator zu dem durch die Region Character und die Grauwerte Image gegebenen Zeichen die Num besten Klassen und gibt die Klassen in Class zurück. Da potentiell mehrere Klassen zurückgeliefert werden können, darf Character nur eine Region (ein Zeichen) enthalten. Falls mehrere Zeichen in einem Aufruf klassifiziert werden sollen, muss do_ocr_multi_class_svm verwendet werden. Der OCR-Klassifikator muss vor der Verwendung von do_ocr_single_class_svm mit trainf_ocr_class_svm trainiert werden.

Parameter

- ▷ **Character** (input_object) ..... region ⇝ *object*
  Zu erkennendes Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte des Zeichens.
- ▷ **OCRHandle** (input_control) ..... ocr_svm ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Num** (input_control) ..... integer-array ⇝ *integer*
  Anzahl der zu bestimmenden besten Klassen.
  Defaultwert: 1
  Wertevorschläge: Num ∈ {1, 2, 3, 4, 5}
- ▷ **Class** (output_control) ..... string(-array) ⇝ *string*
  Ergebnis der Klassifikation des Zeichens durch die SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert do_ocr_single_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_svm, read_ocr_class_svm

Alternativen

do_ocr_multi_class_svm

Siehe auch

create_ocr_class_svm, classify_class_svm

Modul

OCR/OCV

do_ocr_word_svm (Character, Image : : OCRHandle, Expression, NumAlternatives, NumCorrections : Class, Word, Score)

Klassifikation einer Zeichengruppe mit einem OCR-Klassifikator.

do_ocr_word_svm verhält sich wie do_ocr_multi_class_svm, indem es mit dem durch OCRHandle gegebenen OCR-Klassifikator zu den durch die Regionen Character und die Grauwerte Image gegebenen Zeichen die jeweils beste Klasse berechnet und diese in Class zurückgibt.

Im Gegensatz zu do_ocr_multi_class_svm behandelt do_ocr_word_svm die Zeichengruppe als eine Einheit, die durch Aneinanderhängen von Klassennamen für jedes einzelne Zeichen ein Wort Word liefert. Dies ermöglicht die Einschränkung der erlaubten Klassifikationsergebnisse auf Textebene durch Angabe eines Ausdrucks Expression, der das erwartete Wort beschreibt.

Ein Ausdruck kann das Wort auf ein festes Lexikon, welches zuvor durch create_lexicon oder import_lexicon erzeugt wurde, einschränken, indem der Name des Lexikons in spitzen Klammern wie in *'<meinlexikon>'* angegeben wird. Ist der Ausdruck nicht von dieser Form, so wird er als regulärer Ausdruck mit derselben Syntax wie für tuple_regexp_match interpretiert. Bei Verwendung von Quantifikatoren wie '*' sollte man normalerweise mit einem Ausdruck der Form '^...*'$*'' dafür sorgen, dass das gesamte Wort verwendet wird.

Im Gegensatz zu tuple_regexp_match erlaubt do_ocr_word_svm **keine** Angabe von zusätzlichen Optionen für Expression in einem Tupel.

Falls das aus den jeweils besten Klassen abgeleitete Wort dem Ausdruck nicht genügt, versucht do_ocr_word_svm eine Korrektur durch Berücksichtigung der NumAlternatives besten Klassen für jedes Zeichen, so wie sie von do_ocr_single_class_svm für ein einzelnes Zeichen bestimmt werden. Hierzu werden alle möglichen Korrekturen ausprobiert, bei denen die Klassifikation von höchstens NumCorrections Zeichenregionen geändert wird. Zu beachten ist hierbei, dass NumAlternatives und NumCorrections Auswirkungen auf die Komplexität des Algorithmus haben und daher in manchen Fällen intern beschränkt werden müssen. Für weitere Informationen siehe den Abschnitt 'Komplexität' weiter unten.

Falls der Ausdruck ein Lexikon beschreibt und die obige Prozedur nicht zum Erfolg führte, so wird das ähnlichste Wort aus dem Lexikon zurückgegeben, solange für die Korrektur weniger als NumCorrections Editieroperationen benötigt werden (siehe suggest_lexicon).

Das Ergebnis wird mit einem Score zwischen 0.0 (keine Korrektur gefunden) und 1.0 (ursprüngliches Wort war schon korrekt) bewertet. Verringert wird der Score durch das Anbringen einer Strafe entsprechend der Anzahl korrigierter Zeichen und einer (kleineren) Strafe für das Ignorieren besserer Klassen um um einen Ausdruck Expression zu erhalten:

$$\mathtt{Score} = 1.0 - \mathtt{PenaltyCorrections} - \mathtt{PenaltyAlternatives}$$

$$\begin{aligned} \mathtt{PenaltyCorrections} &= \alpha * \mathtt{num_corr} \\ \mathtt{PenaltyAlternatives} &= \alpha * \beta * \mathtt{num_alt} \end{aligned}$$

wobei num_corr die Anzahl tatsächlich angebrachter Korrekturen und num_alt die Anzahl ignorierter Alternativen ist.

$$\begin{aligned} \alpha &= \frac{1}{\mathtt{NumCorrections} + 1} \\ \beta &= \frac{1}{\mathtt{NumCorrections} * (\mathtt{NumAlternatives} - 1) + 2} \end{aligned}$$

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu erkennende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte der Zeichen.
- ▷ **OCRHandle** (input_control) ..... ocr_svm ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **Expression** (input_control) ..... string ⇝ *string*
  Ausdruck, der die erlaubte Wortstruktur beschreibt
- ▷ **NumAlternatives** (input_control) ..... integer ⇝ *integer*
  Anzahl der Klassen pro Zeichen, die für die interne Wortkorrektur berücksichtigt werden.
  Defaultwert: 3
  Wertevorschläge: NumAlternatives ∈ {3, 4, 5}
  Typischer Wertebereich: 1 ≤ NumAlternatives
- ▷ **NumCorrections** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl der korrigierten Zeichen.
  Defaultwert: 2
  Wertevorschläge: NumCorrections ∈ {1, 2, 3, 4, 5}
  Typischer Wertebereich: 0 ≤ NumCorrections

▷ **Class** (output_control) . . . . . string(-array) ⇝ *string*
Ergebnis der Klassifikation der Zeichen durch die SVM.
Parameteranzahl: Class == Character

▷ **Word** (output_control) . . . . . string ⇝ *string*
Text des Wortes nach Klassifikation und Korrektur.

▷ **Score** (output_control) . . . . . real ⇝ *real*
Ähnlichkeitsmaß von korrigiertem Wort zum Original.

Komplexität

Die Komplexität des Prüfens aller möglichen Korrekturen hat die Größenordnung $O((n * a)^{min(c,n)})$, wobei a die Anzahl der Alternativen, n die Anzahl der Zeichenregionen und c die Anzahl der erlaubten Korrekturen ist. Als Absicherung gegen quasi-unendliche Schleifen im Falle von großem n wird der Wert von c intern beschränkt auf 5, 3, oder 1, falls $a * n >= 30$, 60, bzw. 90.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `do_ocr_word_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`trainf_ocr_class_svm`, `read_ocr_class_svm`

Alternativen

`do_ocr_multi_class_svm`

Siehe auch

`create_ocr_class_svm`, `classify_class_svm`

Modul

OCR/OCV

get_features_ocr_class_svm (Character : : OCRHandle, Transform : Features)

Berechnung der Merkmale eines Zeichens.

`get_features_ocr_class_svm` berechnet die Merkmale des Zeichens, das durch `Character` angegeben wird, mit dem OCR-Klassifikator `OCRHandle` und gibt sie in `Features` zurück. Im Gegensatz zu `do_ocr_single_class_svm` und `do_ocr_multi_class_svm` wird das Zeichen als ein Bildobjekt angegeben. Vor der Verwendung von `get_features_ocr_class_svm` muss also `reduce_domain` aufgerufen werden. Der Parameter `Transform` legt fest, ob die im Klassifikator durch `Preprocessing` in `create_ocr_class_svm` festgelegte Merkmalstransformation angewendet werden soll (`Transform` = *'true'*) oder ob die untransformierten Merkmale zurückgegeben werden sollen (`Transform` = *'false'*). `get_features_ocr_class_svm` kann dazu verwendet werden, die zur Klassifikation verwendeten Merkmale zu inspizieren.

Parameter

▷ **Character** (input_object) . . . . . singlechannelimage ⇝ *object* : byte / uint2
Eingabezeichen.

▷ **OCRHandle** (input_control) . . . . . ocr_svm ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **Transform** (input_control) . . . . . string ⇝ *string*
Soll der Merkmalsvektor mit der Vorverarbeitung transformiert werden?
Defaultwert: 'true'
Werteliste: `Transform` ∈ {'true', 'false'}

▷ **Features** (output_control) . . . . . real-array ⇝ *real*
Merkmalsvektor des Zeichens.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_features_ocr_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`trainf_ocr_class_svm`

Siehe auch

`create_ocr_class_svm`

Modul

OCR/OCV

get_params_ocr_class_svm (: : OCRHandle : WidthCharacter, HeightCharacter, Interpolation, Features, Characters, KernelType, KernelParam, Nu, Mode, Preprocessing, NumComponents)

Auslesen der Parameter eines OCR-Klassifikators.

`get_params_ocr_class_svm` gibt die Parameter des durch `OCRHandle` angegebenen OCR-Klassifikators, die bei der Erzeugung mit `create_ocr_class_svm` verwendet wurden, zurück. Dies ist insbesondere nützlich, wenn der Klassifikator mit `read_ocr_class_svm` eingelesen wurde. Die Ausgabedaten von `get_params_ocr_class_svm` können z.B. verwendet werden, um zu überprüfen, ob ein zu lesendes Zeichen in dem Klassifikator vorhanden ist. Zur Beschreibung der Parameter siehe `create_ocr_class_svm`.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_svm ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **WidthCharacter** (output_control) . . . . . integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.

▷ **HeightCharacter** (output_control) . . . . . integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.

▷ **Interpolation** (output_control) . . . . . string ⇝ *string*
Interpolationsmodus bei der Skalierung der Zeichen.

▷ **Features** (output_control) . . . . . string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden.

▷ **Characters** (output_control) . . . . . string-array ⇝ *string*
Zeichen des zu lesenden Zeichensatzes.

▷ **KernelType** (output_control) . . . . . string ⇝ *string*
Art des Kernels.

▷ **KernelParam** (output_control) . . . . . real ⇝ *real*
Zusätzliche Parameter für die Kernelfunktion.

▷ **Nu** (output_control) . . . . . real ⇝ *real*
Regularisierungskonstante der SVM.

▷ **Mode** (output_control) . . . . . string ⇝ *string*
Modus der SVM.

▷ **Preprocessing** (output_control) . . . . . string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.

▷ **NumComponents** (output_control) . . . . . integer ⇝ *integer*
Parameter der Vorverarbeitung: Anzahl der transformierten Merkmale (ignoriert bei Preprocessing = *'none'* und Preprocessing = *'normalization'*).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_params_ocr_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_ocr_class_svm, read_ocr_class_svm

Nachfolger

do_ocr_single_class_svm, do_ocr_multi_class_svm

Siehe auch

trainf_ocr_class_svm, get_params_class_svm

Modul

OCR/OCV

get_prep_info_ocr_class_svm (: : OCRHandle, TrainingFile, Preprocessing : InformationCont, CumInformationCont)

Berechnung des Informationsgehaltes der vorverarbeiteten Merkmalsvektoren eines OCR-Klassifikators.

get_prep_info_ocr_class_svm berechnet den Informationsgehalt der mit der durch Preprocessing gegebenen Vorverarbeitung transformierten Komponenten der Trainingsvektoren. Preprocessing kann auf *'principal_components'* oder *'canonical_variates'* gesetzt werden. Der OCR-Klassifikator OCRHandle muss mit create_ocr_class_svm erzeugt worden sein. Die zugrundeliegenden Vorverarbeitungen sind bei create_class_svm beschrieben. Der Informationsgehalt wird aus die Variation der transformierten Komponenten der Trainingsvektoren berechnet, d.h. er wird rein aufgrund der Trainingsdaten unabhängig von einer Fehlerrate bei einer Klassifikation der Trainingsdaten berechnet. Der Informationsgehalt wird für alle relevanten Komponenten der transformierten Merkmalsvektoren (NumFeatures für *'principal_components'* und $\min(\texttt{NumClasses} - 1, \texttt{NumFeatures})$ für *'canonical_variates'*, siehe create_class_svm) in InformationCont zurückgegeben. Der Informationsgehalt wird als eine Zahl zwischen 0 und 1 dargestellt. Ein prozentualer Informationsgehalt kann leicht durch Multiplikation mit 100 berechnet werden. Der kumulierte Informationsgehalt der ersten n Komponenten wird in der n-ten Komponente von CumInformationCont zurückgegeben, d.h. CumInformationCont enthält die Summen der ersten n Elemente von InformationCont. Um get_prep_info_ocr_class_svm verwenden zu können, müssen Trainingsmuster in den durch TrainingFile angegebenen Trainingsdateien abgespeichert werden (siehe write_ocr_trainf).

InformationCont und CumInformationCont können dazu verwendet werden, um zu entscheiden, wie viele Komponenten der transformierten Merkmalsvektoren relevante Information enthalten. Ein oft verwendetes Kriterium ist z.B. zu verlangen, dass die transformierten Daten x% (z.B. 90%) der Daten enthalten sollten. Dies kann leicht anhand des ersten Wertes von CumInformationCont, der über x% liegt, bestimmt werden. Der so erhaltene Wert kann bei einem neuerlichen Aufruf von create_ocr_class_svm als NumComponents verwendet werden. Da zum Aufruf von get_prep_info_ocr_class_svm schon ein Klassifikator mit create_ocr_class_svm erzeugt werden muss, also auch ein initialer Wert von NumComponents bei create_ocr_class_svm angegeben werden muss, aber bei Verwendung von get_prep_info_ocr_class_svm typischerweise noch nicht bekannt ist, wie viele Komponenten relevant sind, empfiehlt sich folgendes zweistufiges Vorgehen, um NumComponents zu bestimmen: In einem ersten Schritt wird ein Klassifikator mit der maximalen Anzahl von NumComponents (NumFeatures für *'principal_components'* und $\min(\texttt{NumClasses} - 1, \texttt{NumFeatures})$ für *'canonical_variates'*) erzeugt. Dann werden die Trainingsmuster mit write_ocr_trainf in einer Datei gespeichert. Anschließend wird mit get_prep_info_ocr_class_svm der Informationsgehalt der Komponenten und somit NumComponents

bestimmt. Danach wird ein neuer Klassifikator mit der gewünschten Anzahl Komponenten erzeugt und der Klassifikator mit trainf_ocr_class_svm trainiert.

Parameter

▷ **OCRHandle** (input_control) ..... ocr_svm ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **TrainingFile** (input_control) ..... filename.read(-array) ⇝ *string*
Name(n) der Trainingsdatei(en).
Defaultwert: 'ocr.trf'
Dateiendung: .trf, .otr

▷ **Preprocessing** (input_control) ..... string ⇝ *string*
Art der Vorverarbeitung (Transformation) der Merkmalsvektoren.
Defaultwert: 'principal_components'
Werteliste: Preprocessing ∈ {'principal_components', 'canonical_variates'}

▷ **InformationCont** (output_control) ..... real-array ⇝ *real*
Relativer Informationsgehalt der transformierten Merkmalsvektoren.

▷ **CumInformationCont** (output_control) ..... real-array ⇝ *real*
Kumulierter Informationsgehalt der transformierten Merkmalsvektoren.

Beispiel

```
* Create the initial OCR classifier.
read_ocr_trainf_names ('ocr.trf', CharacterNames, CharacterCount)
create_ocr_class_svm (8, 10, 'constant', 'default', CharacterNames, \
                      'rbf', 0.01, 0.01, 'one-versus-one', \
                      'principal_components', 81, OCRHandle)
* Get the information content of the transformed feature vectors.
get_prep_info_ocr_class_svm (OCRHandle, 'ocr.trf', 'principal_components', \
                             InformationCont, CumInformationCont)
* Determine the number of transformed components.
* NumComp = [...]
* Create the final OCR classifier.
create_ocr_class_svm (8, 10, 'constant', 'default', CharacterNames, \
                      'rbf', 0.01, 0.01,'one-versus-one', \
                      'principal_components', NumComp, OCRHandle)
* Train the final classifier.
trainf_ocr_class_svm (OCRHandle, 'ocr.trf', 0.001, 'default')
write_ocr_class_svm (OCRHandle, 'ocr.osc')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_prep_info_ocr_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls get_prep_info_ocr_class_svm den Fehler 9211 (Matrix ist nicht positiv definit) bei Preprocessing = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_ocr_class_svm, write_ocr_trainf, append_ocr_trainf, write_ocr_trainf_image

Nachfolger

clear_ocr_class_svm, create_ocr_class_svm

Modul

OCR/OCV

get_support_vector_num_ocr_class_svm (: : OCRHandle : NumSupportVectors, NumSVPerSVM)

Auslesen der Anzahl der Support-Vektoren eines OCR-Klassifikators.

get_support_vector_num_ocr_class_svm liefert in NumSupportVectors die Anzahl der in dem durch OCRHandle angegebenen SVM abgespeicherten Support-Vektoren zurück. get_support_vector_num_ocr_class_svm sollte aufgerufen werden, bevor mit get_support_vector_class_svm die einzelnen Indizes der SV ausgelesen werden, z.B. um zu Visualisieren, welche der Trainingsdaten SV geworden sind (siehe get_support_vector_ocr_class_svm). Die Anzahl der SV in jeder einzelnen SVM wird in NumSVPerSVM zurückgegeben. Die Summe der einzelnen Klassifikatoren unterscheidet sich von NumSupportVectors, da einzelne SV-Auswertungen innerhalb der verschiedenen binären Sub-SVMs wiederverwendet werden.

Parameter

- ▷ **OCRHandle** (input_control) ..... ocr_svm ⇝ *handle*
  Handle des OCR.
- ▷ **NumSupportVectors** (output_control) ..... integer ⇝ *integer*
  Gesamtanzahl der Support-Vektoren.
- ▷ **NumSVPerSVM** (output_control) ..... integer-array ⇝ *integer*
  Anzahl der SV jeder einzelnen Sub-SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_sample_num_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_svm

Nachfolger

get_support_vector_ocr_class_svm

Siehe auch

create_ocr_class_svm

Modul

OCR/OCV

get_support_vector_ocr_class_svm (: : OCRHandle, IndexSupportVector : Index)

Auslesen des Index eines Support-Vektors aus einem trainierten auf SVM basierenden OCR-Klassifikators.

Der Operator get_support_vector_ocr_class_svm erlaubt eine Beziehung zwischen originalem Trainingsdatensatz und der durch das Training entstandenen Support-Vektoren (SV) innerhalb einer auf SVM basierenden OCR (gegeben in OCRHandle) herzustellen. Der Index des abzufragenden SV wird mit IndexSupportVector festgelegt. Er wird ab 0 gezählt, d.h. IndexSupportVector muss zwischen 0 und IndexSupportVectors − 1 liegen, wobei IndexSupportVectors mit get_support_vector_num_ocr_class_svm bestimmt werden kann. Der Index, den dieser Datensatz im ursprünglichen Trainingdatensatz hatte, wird in Index zurückgegeben. get_support_vector_ocr_class_svm kann z.B. dazu verwendet werden, die Support-Vektoren zu visualisieren. Dazu muss die Trainingsdatei, mit dem die SVM trainiert wurde, mit read_ocr_trainf eingelesen werden. Der um *1* erhöhte Wert von Index kann dann mit select_obj verwendet werden, um die Support-Vektoren aus den Trainingszeichen auszuwählen. Falls mehrere Trainingsdateien in trainf_ocr_class_svm

angegeben worden sind, verhält sich Index so, als ob die Trainingsdateien vorher mit concat_ocr_trainf zu einer großen Trainingsdatei zusammengefügt worden wären.

Parameter

- ▷ **OCRHandle** (input_control) . . . . . ocr_svm ⇝ *handle*
  Handle des OCR.
- ▷ **IndexSupportVector** (input_control) . . . . . integer-array ⇝ *integer*
  Anzahl der gespeicherten Support-Vektoren.
- ▷ **Index** (output_control) . . . . . real ⇝ *real*
  Index des Support-Vektors im Trainingsdatensatz.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_support_vector_ocr_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_svm, get_support_vector_num_ocr_class_svm

Siehe auch

create_ocr_class_svm, read_ocr_trainf, append_ocr_trainf, concat_ocr_trainf

Modul

OCR/OCV

read_ocr_class_svm (: : FileName : OCRHandle)

Lesen eines OCR-Klassifikators aus Datei.

read_ocr_class_svm liest einen mit write_ocr_class_svm abgespeicherten OCR-Klassifikator ein. Da das Trainieren eines OCR-Klassifikators relativ lange dauern kann, wird der Klassifikator typischerweise in einem Offline-Prozess trainiert und mit write_ocr_class_svm gespeichert. Im Online-Prozess wird der Klassifikator mit read_ocr_class_svm gelesen und danach zur Klassifikation mit do_ocr_single_class_svm oder do_ocr_multi_class_svm verwendet.

Parameter

- ▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
  Name der Datei.
  Dateiendung: .osc, .fnt
- ▷ **OCRHandle** (output_control) . . . . . ocr_svm ⇝ *handle*
  Handle des OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_ocr_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

do_ocr_single_class_svm, do_ocr_multi_class_svm

Siehe auch

create_ocr_class_svm, write_ocr_class_svm, read_class_svm, write_class_svm

Modul

OCR/OCV

reduce_ocr_class_svm (: : OCRHandle, Method, MinRemainingSV, MaxError : OCRHandleReduced)

Approximieren einer bereits trainierten auf SVM basierenden OCR-Klassifikators durch eine reduzierte SVM.

reduce_ocr_class_svm erzeugt eine reduzierte Kopie des auf SVM basierenden OCR-Klassifikator OCRHandle und gibt diesen in OCRHandleReduced zurück. Die Parameter Method, MinRemainingSV und MaxError haben dieselbe Bedeutung wie bei reduce_class_svm und werden dort beschrieben. Hier ist anzumerken, dass die Klassifikationszeit signifikant auch durch eine Vorverarbeitung in create_ocr_class_svm reduziert werden kann, wobei eventuell weniger Fehler entstehen.

Parameter

- ▷ **OCRHandle** (input_control) . . . . . ocr_svm ⇝ *handle*
  Handle der originalen SVM-basierten OCR-Klassifikators.
- ▷ **Method** (input_control) . . . . . string ⇝ *string*
  Art der Nachbearbeitung, um die Anzahl der SV zu reduzieren.
  Defaultwert: 'bottom_up'
  Werteliste: Method ∈ {'bottom_up'}
- ▷ **MinRemainingSV** (input_control) . . . . . integer ⇝ *integer*
  Minimale Anzahl der verbleibenden SV.
  Defaultwert: 2
  Wertevorschläge: MinRemainingSV ∈ {2, 3, 4, 5, 7, 10, 15, 20, 30, 50}
  Restriktion: MinRemainingSV >= 2
- ▷ **MaxError** (input_control) . . . . . real ⇝ *real*
  Maximal erlaubter Fehler, der durch die Reduktion erzeugt wird.
  Defaultwert: 0.001
  Wertevorschläge: MaxError ∈ {0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05}
  Restriktion: MaxError > 0.0
- ▷ **OCRHandleReduced** (output_control) . . . . . ocr_svm ⇝ *handle*
  Handle des reduzierten OCR-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert reduce_ocr_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_svm, get_support_vector_num_ocr_class_svm

Nachfolger

do_ocr_single_class_svm, do_ocr_multi_class_svm,
get_support_vector_ocr_class_svm, get_support_vector_num_ocr_class_svm

Siehe auch

create_ocr_class_svm

Literatur

Bernhard Schölkopf, Alexander J.Smola: „Learning with Kernels“; The MIT Press, London; 1999.

Modul

OCR/OCV

select_feature_set_trainf_svm (: : TrainingFile, FeatureList, SelectionMethod, Width, Height, GenParamName, GenParamValue : OCRHandle, FeatureSet, Score)

Wählt die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus.

select_feature_set_trainf_svm wählt die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus. Die Daten für das OCR-Klassifikationsproblem werden mit der Trainingsdatei TrainingFile ausgewählt. Als Klassifikator wird eine Support-Vektor-Maschine (SVM) verwendet.

Es wird eine Untermenge aller angegebenen OCR-Merkmale gewählt. Alle zu untersuchenden Merkmale werden in FeatureList spezifiziert. Die Liste der möglichen OCR-Merkmale wird in der Dokumentation von create_ocr_class_svm näher erläutert. Die letztendlich ausgewählte Merkmalsuntermenge wird in FeatureSet zurückgegeben.

select_feature_set_trainf_svm ist auf OCR-Klassifikationsprobleme spezialisiert und unterstützt nur Merkmale, die in der Auswahlliste stehen. Für andere Merkmale bietet sich somit die Nutzung des generellen Operators select_feature_set_svm an.

Für den Selektionsprozess können in SelectionMethod zwei verschiedene Methoden ausgewählt werden: entweder die Greedy-Variante *'greedy'* (das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt) oder die dynamisch oszillierende Suche *'greedy_oscillating'* (Das momentan erfolgversprechendste Merkmal wird zur Auswahl hinzugefügt. Danach wird getestet ob eines der hinzugefügten Merkmale entbehrlich ist.). Während *'greedy'* schneller terminiert, kann *'greedy_oscillating'* bessere Ergebnisse erzielen wenn viele Trainingsdaten vorliegen.

Als Optimierungskriterium dient die Klassifikationsrate, die mit einem zwei-fachen Kreuzvalidierungsverfahren ermittelt wird. Die beste erreichte Klassifikationsrate wird in Score zurückgegeben. OCRHandle gibt einen Klassifikator zurück, der mit der entsprechenden Merkmalsauswahl parametrisiert und trainiert wurde.

Die Parameter *'nu'* und *'gamma'*, die für das SVM benötigt werden, können über die generischen Parameter GenParamName und GenParamValue entweder direkt auf einen Wert oder auf *'auto'* gesetzt werden. Wenn *'auto'* ausgewählt ist, wird für den entsprechenden Wert auch eine Optimierung durchgeführt. Diese automatische Schätzung kann die Ausführung der Merkmalsauswahl signifikant verlangsamen, was bei größeren Datensätzen zu Rechenzeiten im Bereich von Stunden führen kann. Eine genauere Erklärung des Parameters *'nu'* und des Kernelparameter des RBF-Kernels *'gamma'* (RBF = radial basis function) findet sich in der Beschreibung von create_class_svm.

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lang sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator select_feature_set_trainf_svm zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

▷ **TrainingFile** (input_control) ..... filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: ''
Dateiendung: .trf, .otr

▷ **FeatureList** (input_control) ..... string(-array) ⇝ *string*
Merkmale, die zur Klassifikation verwendet werden können.
Defaultwert: ['zoom_factor','ratio','width','height','foreground','foreground_grid_9','foreground_grid_16', 'anisometry','compactness','convexity','moments_region_2nd_invar','moments_region_2nd_rel_invar', 'moments_region_3rd_invar','moments_central','phi','num_connect','num_holes','projection_horizontal', 'projection_vertical','projection_horizontal_invar','projection_vertical_invar','chord_histo','num_runs', 'pixel','pixel_invar','pixel_binary','gradient_8dir','cooc','moments_gray_plane']
Werteliste: FeatureList ∈ {'default', 'zoom_factor', 'ratio', 'width', 'height', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'anisometry', 'compactness', 'convexity',

'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'phi', 'num_connect', 'num_holes', 'projection_horizontal', 'projection_vertical', 'projection_horizontal_invar', 'projection_vertical_invar', 'chord_histo', 'num_runs', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'cooc', 'moments_gray_plane'}

▷ **SelectionMethod** (input_control) ..... string ⇝ *string*
Methode zur Auswahl der Merkmale.
Defaultwert: 'greedy'
Werteliste: SelectionMethod ∈ {'greedy', 'greedy_oscillating'}

▷ **Width** (input_control) ..... integer ⇝ *integer*
Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 15

▷ **Height** (input_control) ..... integer ⇝ *integer*
Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
Defaultwert: 16

▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
Namen von (optionalen) Parametern für die Steuerung des Verhaltens verwendeten des SVMs.
Defaultwert: []
Werteliste: GenParamName ∈ {'nu', 'gamma'}

▷ **GenParamValue** (input_control) ..... number-array ⇝ *real* / integer / string
Zu den optionalen generischen Parametern gehörende Werte.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'auto', '0.1', '0.3'}

▷ **OCRHandle** (output_control) ..... ocr_svm ⇝ *handle*
Mit den selektierten Merkmalen trainierter SVM-Klassifikator.

▷ **FeatureSet** (output_control) ..... string-array ⇝ *string*
Ausgewählten Merkmale.

▷ **Score** (output_control) ..... real-array ⇝ *real*
Klassifikationsrate, die mit dem ausgewählten Merkmalssatz erreicht wurde.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert select_feature_set_trainf_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Alternativen

select_feature_set_trainf_mlp, select_feature_set_trainf_knn, select_feature_set_trainf_mlp_protected

Siehe auch

select_feature_set_trainf_svm_protected, select_feature_set_svm

Modul

OCR/OCV

select_feature_set_trainf_svm_protected (: : TrainingFile, Password, FeatureList, SelectionMethod, Width, Height, GenParamName, GenParamValue : OCRHandle, FeatureSet, Score)

Wählt aus (geschützten) Trainingsdateien die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus.

select_feature_set_trainf_svm_protected wählt die optimalen Merkmale für ein bestimmtes OCR-Klassifikationsproblem aus. Die Daten für das OCR-Klassifikationsproblem werden mit der Trainingsdatei TrainingFile ausgewählt. Als Klassifikator wird eine Support-Vektor-Maschine (SVM) verwendet.

Die Funktionalität des Operators entspricht der Funktionalität von select_feature_set_trainf_svm, mit der Erweiterung, dass select_feature_set_trainf_svm_protected sowohl ungeschützte als auch geschützte Trainingsdateien verarbeiten kann. Geschützte Trainingsdateien können nur mit dem korrekten Benutzerpasswort Password verwendet werden. Falls genau ein Passwort Password übergeben wird, werden alle Eingabedateien TrainingFile mit diesem Passwort überprüft, ansonsten muss die Anzahl der Passworte gleich der Anzahl der Eingabedateien sein, und die Eingabedatei an Position n wird mit dem Passwort an Position n überprüft. Für ungeschützte Trainingsdateien werden die übergebenen Passworte ignoriert.

Für eine genauere Beschreibung der Operatorfunktionalität siehe select_feature_set_trainf_svm. Das Konzept des Schützens von OCR-Trainingsdaten in HALCON ist in protect_ocr_trainf beschrieben.

Achtung

Die Laufzeit dieses Operators kann mit größeren Datensätzen und einer längeren Merkmalsliste unter Umständen sehr lang sein.

Es ist zu beachten, dass dieser Operator nicht aufgerufen werden sollte, wenn für das Training nur ein kleiner Datensatz verfügbar ist. Auf Grund des Risikos der Überanpassung kann der Operator select_feature_set_trainf_svm_protected zwar einerseits einen Klassifikator mit sehr hohem Score liefern. Andererseits weist der Klassifikator jedoch eine schlechte Erkennungsrate auf, wenn dieser getestet wird.

Parameter

- ▷ **TrainingFile** (input_control) ..... filename.read(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert: ''
  Dateiendung: .trf, .otr
- ▷ **Password** (input_control) ..... string(-array) ⇝ *string*
  Passworte für geschützte Trainingsdateien.
- ▷ **FeatureList** (input_control) ..... string(-array) ⇝ *string*
  Merkmale, die zur Klassifikation verwendet werden können.
  Defaultwert: ['zoom_factor','ratio','width','height','foreground','foreground_grid_9','foreground_grid_16', 'anisometry','compactness','convexity','moments_region_2nd_invar','moments_region_2nd_rel_invar', 'moments_region_3rd_invar','moments_central','phi','num_connect','num_holes','projection_horizontal', 'projection_vertical','projection_horizontal_invar','projection_vertical_invar','chord_histo','num_runs', 'pixel','pixel_invar','pixel_binary','gradient_8dir','cooc','moments_gray_plane']
  Werteliste: FeatureList ∈ {'default', 'zoom_factor', 'ratio', 'width', 'height', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'anisometry', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'phi', 'num_connect', 'num_holes', 'projection_horizontal', 'projection_vertical', 'projection_horizontal_invar', 'projection_vertical_invar', 'chord_histo', 'num_runs', 'pixel', 'pixel_invar', 'pixel_binary', 'gradient_8dir', 'cooc', 'moments_gray_plane'}
- ▷ **SelectionMethod** (input_control) ..... string ⇝ *string*
  Methode zur Auswahl der Merkmale.
  Defaultwert: 'greedy'
  Werteliste: SelectionMethod ∈ {'greedy', 'greedy_oscillating'}
- ▷ **Width** (input_control) ..... integer ⇝ *integer*
  Breite des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
  Defaultwert: 15
- ▷ **Height** (input_control) ..... integer ⇝ *integer*
  Höhe des Rechtecks, auf die die Grauwerte des segmentierten Zeichens skaliert werden.
  Defaultwert: 16
- ▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
  Namen von (optionalen) Parametern für die Steuerung des Verhaltens verwendeten des SVMs.
  Defaultwert: []
  Werteliste: GenParamName ∈ {'nu', 'gamma'}
- ▷ **GenParamValue** (input_control) ..... number-array ⇝ *real* / integer / string
  Zu den optionalen generischen Parametern gehörende Werte.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {'auto', '0.1', '0.3'}

▷ **OCRHandle** (output_control) . . . . . ocr_svm ⇝ *handle*
Mit den selektierten Merkmalen trainierter SVM-Klassifikator für OCR.

▷ **FeatureSet** (output_control) . . . . . string-array ⇝ *string*
Ausgewählten Merkmale.

▷ **Score** (output_control) . . . . . real-array ⇝ *real*
Klassifikationsrate, die mit dem ausgewählten Merkmalssatz erreicht wurde.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `select_feature_set_trainf_svm_protected` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`protect_ocr_trainf`

Alternativen

`select_feature_set_trainf_mlp_protected`

Siehe auch

`select_feature_set_trainf_svm`, `select_feature_set_svm`

Modul

OCR/OCV

serialize_ocr_class_svm (: : OCRHandle : SerializedItemHandle)

Serialisieren eines SVM-basierten OCR-Klassifikators.

`write_ocr_class_svm` serialisiert einen SVM-basierten OCR-Klassifikator (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von `write_ocr_class_svm` in eine Datei geschrieben werden. Der OCR-Klassifikator wird in dem Handle `OCRHandle` übergeben. Der serialisierte OCR-Klassifikator wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_ocr_class_svm` wieder deserialisiert werden.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_svm ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_ocr_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_svm

Nachfolger

clear_ocr_class_svm, fwrite_serialized_item, send_serialized_item, deserialize_ocr_class_svm

Siehe auch

create_ocr_class_svm, read_ocr_class_svm, write_class_svm, read_class_svm, deserialize_ocr_class_svm

Modul

OCR/OCV

trainf_ocr_class_svm (: : OCRHandle, TrainingFile, Epsilon, TrainMode :)

Trainieren eines OCR-Klassifikators.

trainf_ocr_class_svm trainiert den OCR-Klassifikator OCRHandle mit den Trainingszeichen, die in den durch TrainingFile angegebenen OCR-Trainingsdateien enthalten sind. Die Trainingsdateien müssen zuvor, z.B. mit write_ocr_trainf, erzeugt worden sein. Die Parameter Epsilon und TrainMode haben dieselbe Bedeutung wie bei train_class_svm und werden dort beschrieben. Es ist zu beachten, dass Trainingszeichen, die keine entsprechende Klasse im Klassifikator OCRHandle haben, verworfen werden.

Parameter

- ▷ **OCRHandle** (input_control) ..... ocr_svm ⇝ *handle*
  Handle des OCR-Klassifikators.
- ▷ **TrainingFile** (input_control) ..... filename.read(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert: 'ocr.trf'
  Dateiendung: .trf, .otr
- ▷ **Epsilon** (input_control) ..... real ⇝ *real*
  Abbruchparameter für das Training.
  Defaultwert: 0.001
  Wertevorschläge: Epsilon ∈ {0.00001, 0.0001, 0.001, 0.01, 0.1}
- ▷ **TrainMode** (input_control) ..... number ⇝ *string* / integer
  Modus des Trainings.
  Defaultwert: 'default'
  Werteliste: TrainMode ∈ {'default', 'add_sv_to_train_set'}

Beispiel

```
* Train an OCR classifier
read_ocr_trainf_names ('ocr.trf', CharacterNames, CharacterCount)
create_ocr_class_svm (8, 10, 'constant', 'default', CharacterNames, \
                      'rbf', 0.01, 0.01, 'one-versus-one', \
                      'normalization', 81, OCRHandle)
trainf_ocr_class_svm (OCRHandle, 'ocr.trf', 0.001, 'default')
write_ocr_class_svm (OCRHandle, 'ocr.osc')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert trainf_ocr_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls trainf_ocr_class_svm den Fehler 9211 (Matrix ist nicht positiv definit) bei Preprocessing = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind. Für diesen Fall ist es sinnvoll, Preprocessing auf *'normalization'* zu setzen. Ansonsten ist es auch möglich, mehr Trainingsmuster hinzuzufügen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ocr_class_svm`, `write_ocr_trainf`, `append_ocr_trainf`, `write_ocr_trainf_image`

Nachfolger

`do_ocr_single_class_svm`, `do_ocr_multi_class_svm`, `write_ocr_class_svm`

Alternativen

`read_ocr_class_svm`

Siehe auch

`train_class_svm`

Modul

OCR/OCV

trainf_ocr_class_svm_protected (: : OCRHandle, TrainingFile, Password, Epsilon, TrainMode :)

Trainieren eines OCR-Klassifikators mit Daten aus (geschützten) Trainingsdateien.

`trainf_ocr_class_svm_protected` trainiert den OCR-Klassifikator `OCRHandle` mit den Trainingszeichen, die in den durch `TrainingFile` angegebenen OCR-Trainingsdateien enthalten sind. Die Funktionalität des Operators entspricht der Funktionalität von `trainf_ocr_class_svm`, mit der Erweiterung, dass `trainf_ocr_class_svm_protected` sowohl ungeschützte als auch geschützte Trainingsdateien verarbeiten kann. Geschützte Trainingsdateien können nur mit dem korrekten Benutzerpasswort `Password` verwendet werden. Falls genau ein Passwort `Password` übergeben wird, werden alle Eingabedateien `TrainingFile` mit diesem Passwort überprüft, ansonsten muss die Anzahl der Passworte gleich der Anzahl der Eingabedateien sein, und die Eingabedatei an Position n wird mit dem Passwort an Position n überprüft. Für ungeschützte Trainingsdateien werden die übergebenen Passworte ignoriert.

Für eine genauere Beschreibung der Operatorfunktionalität siehe `trainf_ocr_class_svm`. Das Konzept des Schützens von OCR-Trainingsdaten in HALCON ist in `protect_ocr_trainf` beschrieben.

Parameter

▷ **OCRHandle** (input_control) . . . . . ocr_svm ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **TrainingFile** (input_control) . . . . . filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: 'ocr.trf'
Dateiendung: `.trf`, `.otr`

▷ **Password** (input_control) . . . . . string(-array) ⇝ *string*
Passworte für geschützte Trainingsdateien.

▷ **Epsilon** (input_control) . . . . . real ⇝ *real*
Abbruchparameter für das Training.
Defaultwert: 0.001
Wertevorschläge: `Epsilon` ∈ {0.00001, 0.0001, 0.001, 0.01, 0.1}

▷ **TrainMode** (input_control) . . . . . number ⇝ *string* / integer
Modus des Trainings.
Defaultwert: 'default'
Werteliste: `TrainMode` ∈ {'default', 'add_sv_to_train_set'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `trainf_ocr_class_svm_protected` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Falls `trainf_ocr_class_svm` den Fehler 9211 (Matrix ist nicht positiv definit) bei `Preprocessing` = *'canonical_variates'* zurückliefert, bedeutet dies typischerweise, dass für die verschiedenen Klassen zu wenige Trainingsmuster gespeichert worden sind. Für diesen Fall ist es sinnvoll, `Preprocessing` auf *'normalization'* zu setzen. Ansonsten ist es auch möglich, mehr Trainingsmuster hinzuzufügen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- OCRHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_ocr_class_svm`, `write_ocr_trainf`, `append_ocr_trainf`, `write_ocr_trainf_image`, `protect_ocr_trainf`

Nachfolger

`do_ocr_single_class_svm`, `do_ocr_multi_class_svm`, `write_ocr_class_svm`

Alternativen

`read_ocr_class_svm`

Siehe auch

`trainf_ocr_class_svm`, `train_class_svm`

Modul

OCR/OCV

`write_ocr_class_svm` (: : OCRHandle, FileName :)

Abspeichern eines OCR-Klassifikators in Datei.

`write_ocr_class_svm` speichert den OCR-Klassifikator `OCRHandle` in der Datei `FileName` ab. Falls keine Datei-Extension in `FileName` angegeben wird, wird die Standard-Extension *'.osc'* an `FileName` angehängt. `write_ocr_class_svm` wird typischerweise aufgerufen, nachdem der Klassifikator mit `trainf_ocr_class_svm` trainiert wurde. Der Klassifikator kann mit `read_ocr_class_svm` wieder eingelesen werden.

Parameter

▷ **`OCRHandle`** (input_control) . . . . . ocr_svm ⇝ *handle*
Handle des OCR-Klassifikators.

▷ **`FileName`** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.osc`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_ocr_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

trainf_ocr_class_svm

Nachfolger

clear_ocr_class_svm

Siehe auch

create_ocr_class_svm, read_ocr_class_svm, write_class_svm, read_class_svm

Modul

OCR/OCV

20.8 Trainingsdateien

append_ocr_trainf (Character, Image : : Class, TrainingFile :)

Anhängen von Trainingszeichen an eine Datei.

append_ocr_trainf dient zum Vorbereiten des Trainings mit trainf_ocr_class_mlp oder trainf_ocr_class_svm. Hierzu werden Regionen, die Buchstaben darstellen, mit ihren Grauwerten (Region und Pixel) und dem zugehörigen Klassennamen auf Datei geschrieben. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in Character muss der zugehörige Name (Klasse) in Class übergeben werden. Die Grauwerte übergibt man in Image. Im Gegensatz zu write_ocr_trainf werden die Zeichen im gleichen Dateiformat an die bestehende Datei angefügt. Falls die Datei zum Zeitpunkt des Aufrufs noch nicht existiert, wird die Datei neu angelegt. In diesem Fall kann das Dateiformat durch den Parameter *'ocr_trainf_version'* des Operators set_system gewählt werden. Besitzt der Dateiname in TrainingFile keine Erweiterung, wird dem Namen die Erweiterung '.trf' angefügt.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu trainierende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte für die Zeichen.
- ▷ **Class** (input_control) ..... string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
- ▷ **TrainingFile** (input_control) ..... filename.write ⇝ *string*
  Name der Trainingsdatei.
  Defaultwert: 'train_ocr'
  Dateiendung: .trf, .otr

Beispiel

```
char       name[128];
char       class[128];

read_image(&Image,"character.tiff");
binary_threshold(Image,&Dark, 'otsu', 'dark', &UsedThreshold);
connection(Dark,&Character);
count_obj(Character,&num);
create_tuple(&Class,num);
open_window(0,0,-1,-1,0,"","",&WindowHandle);
set_color(WindowHandle,"red");
for (i=0; i<num; i++) {
  select_obj(Character,&SingleCharacter,i);
  disp_region(SingleCharacter,WindowHandle);
  printf("class of character %d ?\n",i);
  scanf("%s\n",class);
  append_ocr_trainf(Character,Image,name,class);
}
```

Ergebnis

Sind die Parameter korrekt, dann liefert `append_ocr_trainf` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `connection`, `read_ocr_trainf`

Nachfolger

`trainf_ocr_class_mlp`, `trainf_ocr_class_svm`, `write_ocr_trainf`

Alternativen

`write_ocr_trainf`, `write_ocr_trainf_image`

Modul

OCR/OCV

concat_ocr_trainf (: : SingleFiles, ComposedFile :)

Zusammenfügen mehrerer Trainingsdateien.

`concat_ocr_trainf` speichert alle Zeichen, die in den Dateien von `SingleFiles` abgelegt sind, in einer neuen Datei mit Namen `ComposedFile` ab. Das Dateiformat kann mit Hilfe des Parameters *'ocr_trainf_version'* des Operators `set_system` gewählt werden. Besitzt der Dateiname in `ComposedFile` keine Erweiterung, wird dem Namen die Erweiterung '.trf' angefügt. Es ist nicht möglich, eine der Dateien von `SingleFiles` als neue Datei `ComposedFile` zu verwenden.

Parameter

▷ **SingleFiles** (input_control) ..... filename.read(-array) ⇝ *string*
Namen der einzelnen Trainingsdateien.
Defaultwert: ''
Dateiendung: `.trf`, `.otr`

▷ **ComposedFile** (input_control) ..... filename.write ⇝ *string*
Name der zusammengesetzten Trainingsdatei.
Defaultwert: 'all_characters'
Dateiendung: `.trf`

Ergebnis

Sind die Parameter korrekt, dann liefert `concat_ocr_trainf` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`write_ocr_trainf`, `append_ocr_trainf`

Nachfolger

`trainf_ocr_class_mlp`, `trainf_ocr_class_svm`, `write_ocr_trainf`

Modul

OCR/OCV

protect_ocr_trainf (: : TrainingFile, Password, TrainingFileProtected :)

Schützen von Trainingsdaten.

Das Erlangen von Trainingsdaten für einen OCR-Klassifikator kann ein komplizierter und langwieriger Vorgang sein. Trainingsdaten sind demnach wertvoll und es kann notwendig sein, deren Verwendung zu schützen. Das grundlegende Konzept hierfür ist, dass geschützte OCR-Trainingsdaten weder sichtbar sind, noch modifiziert werden können und dass sichergestellt ist, dass deren Verwendung, z.B. für das Trainieren eines Klassifikators, durch ein Benutzerpasswort geschützt ist.

Das Vorgehen in HALCON ist, dass OCR-Trainingsdaten zunächst auf die übliche Weise erlangt und in eine ungeschützte OCR-Trainingsdatei geschrieben werden. Anschließend wird eine neue Datei erzeugt, die die mit einem Benutzerpasswort geschützten Trainingsdaten enthält. Geschützte Trainingsdaten können mit speziellen Operatoren verarbeitet werden, wobei für das Verwenden der Daten das korrekte Benutzerpasswort bekannt sein muss.

Dabei ist zu beachten, dass Daten in geschützten OCR-Trainingsdateien weder sichtbar sind, noch in irgendeiner Art und Weise modifiziert werden können, d.h. es ist wichtig, dass die ungeschützten Trainingsdateien mit den Originaldaten auch nach dem Schützen vorhanden bleiben.

Geschützte OCR-Trainingsdateien können für das Trainieren von MLP-Klassifikatoren und SVM-Klassifikatoren verwendet werden.

protect_ocr_trainf schützt die OCR-Trainingsdaten in der Datei TrainingFile und schreibt diese in die geschützte Trainingsdatei TrainingFileProtected. Das Schützen der Daten erfolgt über das Benutzerpasswort Password. Dasselbe Passwort muss später bei der Verwendung der geschützten Trainingsdaten angegeben werden, z.B. bei den Operatoren trainf_ocr_class_mlp_protected oder read_ocr_trainf_names_protected. Leere Passworte (oder Passworte, die den leeren String beinhalten) sind nicht erlaubt. Falls genau ein Passwort Password übergeben wird, werden alle Ausgabedateien TrainingFileProtected mit diesem Passwort geschützt, ansonsten muss die Anzahl der Passworte gleich der Anzahl der Ein- und Ausgabedateien sein, und die Ausgabedatei an Position n wird mit dem Passwort an Position n geschützt.

Parameter

▷ **TrainingFile** (input_control) ..... filename.read ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: ”
Dateiendung: .trf, .otr

▷ **Password** (input_control) ..... string(-array) ⇝ *string*
Passworte zum Schützen der Trainingsdateien.

▷ **TrainingFileProtected** (input_control) ..... filename.write ⇝ *string*
Namen der geschützten Trainingsdateien.
Dateiendung: .trf

Ergebnis

Sind die Parameter korrekt, dann liefert protect_ocr_trainf den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

write_ocr_trainf, append_ocr_trainf, concat_ocr_trainf

Nachfolger

read_ocr_trainf_names_protected, trainf_ocr_class_mlp_protected, trainf_ocr_class_svm_protected, select_feature_set_trainf_mlp_protected, select_feature_set_trainf_svm_protected

Siehe auch

select_feature_set_trainf_mlp

Modul

OCR/OCV

read_ocr_trainf (: Characters : TrainingFile : CharacterNames)

Einlesen von Trainingszeichen für OCR als Bilder.

read_ocr_trainf liest alle Zeichen aus den angegebenen Trainings-Dateien ein und wandelt sie in Bilder um. Die Dateiendung für die Trainings-Datei ist in HALCON 'trf'. Die Definitionsbereiche werden auf die Vordergrundregion der Zeichen (gemäß write_ocr_trainf) beschränkt. Die Namen der Zeichen werden im Parameter CharacterNames zurückgegeben. Es kann mehr als eine Datei eingelesen werden, dabei werden die Dateien entsprechend der Ordnung der Dateinamen abgearbeitet.

Parameter

- ▷ **Characters** (output_object) ..... image-array ⇝ *object :* byte / uint2
  Eingelesene Zeichen.
- ▷ **TrainingFile** (input_control) ..... filename.read(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert: ''
  Dateiendung: .trf, .otr
- ▷ **CharacterNames** (output_control) ..... string-array ⇝ *string*
  Bezeichnung der eingelesenen Zeichen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_ocr_trainf den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

write_ocr_trainf

Nachfolger

disp_image, select_obj, zoom_image_size

Alternativen

read_ocr_trainf_select

Modul

OCR/OCV

read_ocr_trainf_names (: : TrainingFile : CharacterNames, CharacterCount)

Abfragen der Zeichen, die in Trainings-Dateien abgelegt sind.

read_ocr_trainf_names bestimmt die Namen aller Zeichen und deren Anzahl in den angegebenen Trainings-Dateien.

Parameter

- ▷ **TrainingFile** (input_control) . . . . . filename.read(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert: ”
  Dateiendung: `.trf`, `.otr`
- ▷ **CharacterNames** (output_control) . . . . . string(-array) ⇝ *string*
  Bezeichnung der eingelesenen Zeichen.
- ▷ **CharacterCount** (output_control) . . . . . integer(-array) ⇝ *integer*
  Anzahl der jeweiligen Zeichen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_ocr_trainf_names` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`write_ocr_trainf`

Siehe auch

`trainf_ocr_class_svm`, `trainf_ocr_class_mlp`

Modul

OCR/OCV

read_ocr_trainf_names_protected (: : TrainingFile,
Password : CharacterNames, CharacterCount)

Abfragen der Zeichen, die in (geschützten) Trainings-Dateien abgelegt sind.

`read_ocr_trainf_names_protected` bestimmt die Namen aller Zeichen und deren Anzahl in den angegebenen Trainings-Dateien. Die Funktionalität des Operators entspricht der Funktionalität von `read_ocr_trainf_names`, mit der Erweiterung, dass `read_ocr_trainf_names_protected` sowohl ungeschützte als auch geschützte Trainingsdateien verarbeiten kann. Geschützte Trainingsdateien können nur mit dem korrekten Benutzerpasswort `Password` verwendet werden. Falls genau ein Passwort `Password` übergeben wird, werden alle Eingabedateien `TrainingFile` mit diesem Passwort überprüft, ansonsten muss die Anzahl der Passworte gleich der Anzahl der Eingabedateien sein, und die Eingabedatei an Position n wird mit dem Passwort an Position n überprüft. Für ungeschützte Trainingsdateien werden die übergebenen Passworte ignoriert.

Für eine genauere Beschreibung der Operatorfunktionalität siehe `read_ocr_trainf_names`. Das Konzept des Schützens von OCR-Trainingsdaten in HALCON ist in `protect_ocr_trainf` beschrieben.

Parameter

- ▷ **TrainingFile** (input_control) . . . . . filename.read(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert: ”
  Dateiendung: `.trf`, `.otr`
- ▷ **Password** (input_control) . . . . . string(-array) ⇝ *string*
  Passworte für geschützte Trainingsdateien.
- ▷ **CharacterNames** (output_control) . . . . . string(-array) ⇝ *string*
  Bezeichnung der eingelesenen Zeichen.
- ▷ **CharacterCount** (output_control) . . . . . integer(-array) ⇝ *integer*
  Anzahl der jeweiligen Zeichen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_ocr_trainf_names_protected` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

protect_ocr_trainf

Siehe auch

read_ocr_trainf_names

Modul

OCR/OCV

read_ocr_trainf_select (: Characters : TrainingFile, SearchNames : FoundNames)

Einlesen von ausgewählten Trainingszeichen für OCR als Bilder.

read_ocr_trainf_select liest die in SearchNames angegebenen Zeichen aus den Trainings-Dateien ein und wandelt sie in Bilder um. Es arbeitet analog zu read_ocr_trainf nur können die zu extrahierenden Zeichen vorgegeben werden.

Parameter

▷ **Characters** (output_object) ..... image-array ⇝ *object :* byte / uint2
Eingelesene Zeichen.

▷ **TrainingFile** (input_control) ..... filename.read(-array) ⇝ *string*
Namen der Trainingsdateien.
Defaultwert: ”
Dateiendung: .trf, .otr

▷ **SearchNames** (input_control) ..... string(-array) ⇝ *string*
Namen der zu extrahierenden Zeichen.
Defaultwert: ’0’

▷ **FoundNames** (output_control) ..... string(-array) ⇝ *string*
Bezeichnung der eingelesenen Zeichen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_ocr_trainf_select den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

write_ocr_trainf

Nachfolger

disp_image, select_obj, zoom_image_size

Alternativen

read_ocr_trainf

Modul

OCR/OCV

write_ocr_trainf (Character, Image : : Class, TrainingFile :)

Abspeichern von Trainingszeichen in eine Datei.

write_ocr_trainf dient zum Vorbereiten des Trainings mit trainf_ocr_class_mlp, trainf_ocr_class_svm oder trainf_ocr_class_box. Hierzu werden Regionen, die Buchstaben darstellen, mit ihren Grauwerten (Region und Pixel) und dem zugehörigen Klassennamen auf Datei geschrieben. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in Character muss der zugehörige Name (Klasse) in Class übergeben werden. Die Grauwerte übergibt man in Image. Besitzt der Dateiname in TrainingFile keine Erweiterung wird dem Namen die Erweiterung '.trf' angefügt. Die Version des Dateiformats, mit welcher die Daten geschrieben werden, kann mit Hilfe des Parameters *'ocr_trainf_version'* des Operators set_system gewählt werden.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *object*
  Zu trainierende Zeichen.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Grauwerte für die Zeichen.
- ▷ **Class** (input_control) ..... string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
- ▷ **TrainingFile** (input_control) ..... filename.write ⇝ *string*
  Name der Trainingsdatei.
  Defaultwert: 'train_ocr'
  Dateiendung: .trf

Ergebnis

Sind die Parameter korrekt, dann liefert write_ocr_trainf den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, connection, read_ocr_trainf

Nachfolger

trainf_ocr_class_mlp, trainf_ocr_class_svm

Modul

OCR/OCV

write_ocr_trainf_image (Character : : Class, TrainingFile :)

Speichern von Zeichen in eine Trainingsdatei.

write_ocr_trainf_image dient zum Vorbereiten des Trainings mit trainf_ocr_class_mlp, trainf_ocr_class_svm oder trainf_ocr_class_box. Hierzu werden Regionen, die Buchstaben darstellen, mit ihren Grauwerten (Region und Pixel) und dem zugehörigen Klassennamen in eine Datei geschrieben. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in Character muss der zugehörige Name (Klasse) in Class übergeben werden. Besitzt der Dateiname in TrainingFile keine Erweiterung, wird dem Namen die Erweiterung '.trf' angefügt. Im Gegensatz zu write_ocr_trainf wird pro Zeichen ein Bild übergeben, dessen Definitionsbereich den Vordergrund des Zeichens angibt. Das Dateiformat kann mit Hilfe des Parameters *'ocr_trainf_version'* des Operators set_system gewählt werden.

Parameter

▷ **Character** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2
Zu trainierende Zeichen.

▷ **Class** (input_control) . . . . . string(-array) ⇝ *string*
Klasse (Name) der Zeichen.

▷ **TrainingFile** (input_control) . . . . . filename.write ⇝ *string*
Name der Trainingsdatei.
Defaultwert: 'train_ocr'
Dateiendung: `.trf`

Ergebnis

Sind die Parameter korrekt, dann liefert `write_ocr_trainf_image` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `connection`, `read_ocr_trainf`

Nachfolger

`trainf_ocr_class_mlp`, `trainf_ocr_class_svm`

Alternativen

`write_ocr_trainf`, `append_ocr_trainf`

Modul

OCR/OCV

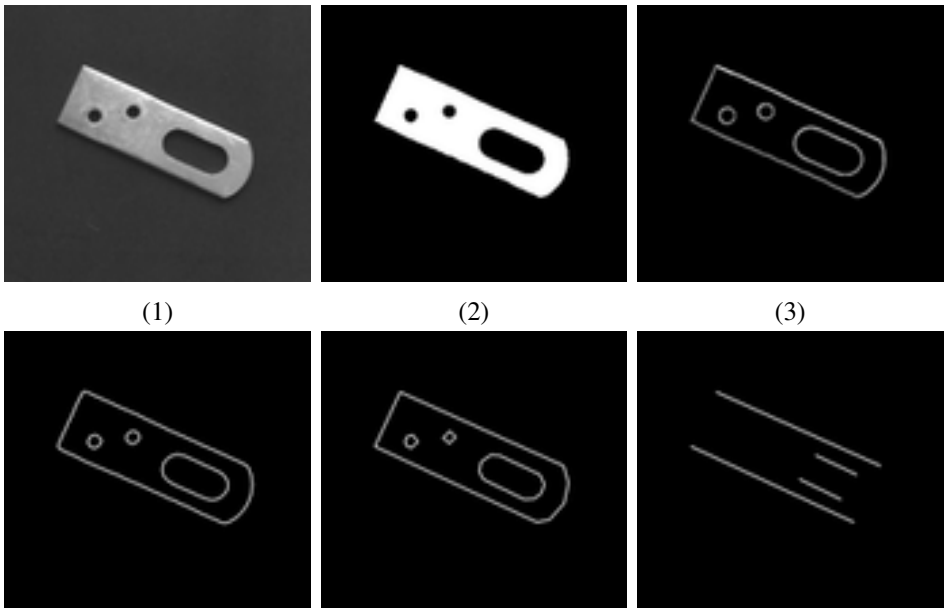
Kapitel 21

Objekt

Dieses Kapitel beschreibt Operatoren, die benutzt werden, um zum einen Informationen zu ikonischen Objekten abzurufen und zum anderen ikonische Objekte zu manipulieren.

Verfügbare Ikonische Objekte

In HALCON stehen verschiedene ikonische Objekte zur Verfügung, die für unterschiedliche Zwecke geeignet sind:



(1) (2) (3)

(4) (5) (6)

In dem Bild eines Metallstücks (1) werden eine Region (2), eine ROI, die unter Verwendung des dilatierten Regionenrands erzeugt wird (3), Konturen (4), Polygone (5), und Parallelen (6) extrahiert.

Bilder: Ein Bild besteht aus ein oder mehreren Kanälen, d.h., Matrizen gleicher Größe, die Grauwerte verschiedener Pixeltypen enthalten. Bilder stellen den Ausgangspunkt der meisten Aufgaben im Bereich der industriellen Bildverarbeitung dar.

Regionen: Eine Region ist als ein Set von Pixeln definiert. Die Pixel einer Region müssen dabei nicht zusammenhängend sein. D.h., eine beliebige Auswahl von Pixeln kann als eine einzige Region behandelt werden. Die Regionenverarbeitung ist z.B. für Blob-Analyse-Aufgaben geeignet. Vor allem eignet sie sich aber auch, um eine 'Region of Interest' (ROI) für nachfolgende subpixel-genaue Operationen festzulegen.

XLDs: XLD ist die Abkürzung für eXtended Line Description und umfasst alle kontur- und polygonbasierten Daten. XLD Konturen werden dabei von sehr vielen Operatoren zurückgegeben oder genutzt, während XLD Polygone und XLD Parallelen nur in besonderen Fällen benötigt werden. Eine Kontur ist eine Sequenz von

subpixel-genauen 2D Kontrollpunkten, die durch Linien miteinander verbunden sind. Typischerweise entspricht der Abstand zwischen den Kontrollpunkten etwa einem Pixel. XLD Objekte enthalten neben den Kontrollpunkten sogenannte lokale und globale Attribute. Typische Beispiele für diese sind die Kantenamplitude eines Kontrollpunktes oder die Regressionsparameter eines Kontursegments. Die Konturverarbeitung ist z.B. für subpixel-genaue Messungen geeignet. Subpixel-genaue Operationen können beschleunigt werden, wenn sie in einer geeigneten ROI ausgeführt werden.

Weiterführende Information

Weitere Details zu den in HALCON verfügbaren Datenstrukturen finden sich im „Quick Guide".

21.1 Information

compare_obj (Objects1, Objects2 : : Epsilon : IsEqual)

Vergleich von ikonischen Objekten hinsichtlich ihrer Gleichheit.

Der Operator compare_obj vergleicht ikonische Objekte. Die ikonischen Objekte werden in den beiden Eingabeparametern Objects1 und Objects2 in Form von (möglicherweise gemischten) Tupeln von Bildern, Regionen oder XLDs übergeben. Dabei wird (für alle n) das n-te Objekt in Objects1 mit dem n-ten Objekt in Objects2 verglichen. Wenn die beiden übergebenen Objekttupel gleich lang und alle Objekte identisch sind, wird der Parameter IsEqual auf 1 gesetzt, andernfalls auf 0.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Abhängig vom Typ der Eingabeobjekte müssen verschiedene Bedingungen erfüllt werden, damit zwei Objekte als gleich gelten. Im Folgenden sind die Bedingungen für die möglichen Objekte aufgelistet:

Bilder: Für alle Kanäle dürfen sich die Grauwerte der entsprechenden Pixel um nicht mehr als Epsilon unterscheiden. Dabei ist zu beachten, dass Bilder nur innerhalb ihres Definitionsbereich (ROI) miteinander verglichen werden und sie nur dann identisch sind, wenn sie auch den gleichen Definitionsbereich(ROI) aufweisen.

Regionen: Die Regionen müssen identisch sein. Der Parameter Epsilon hat keine Auswirkung auf den Vergleich von Regionen.

XLD Konturen: XLD Konturen müssen die gleiche Anzahl an Punkten sowie die gleichen Attribute haben. Die Koordinaten der Punkte und die Attributwerte dürfen sich um nicht mehr als Epsilon unterscheiden.

XLD Polygone: XLD Polygone müssen die gleiche Anzahl an Liniensegmenten aufweisen. Die Länge und die Orientierung dieser Segmente sowie die Koordinaten der Kontrollpunkte dürfen sich um nicht mehr als Epsilon unterscheiden.

XLD Parallelen und erweiterte XLD Parallelen: XLD Parallelen müssen einen identischen Index für das Start- und Endliniensegment der Parallelen des ersten Polygons (P1) und des zweiten Polygons (P2) sowie identische Zeiger zu der zugrundeliegenden Kontur aufweisen.

Modifizierte XLD Parallelen: Für modifizierte XLD Parallelen gelten dieselben Bedingungen wie für XLD Parallelen und erweiterte XLD Parallelen. Zusätzlich muss auch der Abstand zwischen den Liniensegmenten der parallelen Polygone identisch sein.

Im Gegensatz zu test_equal_obj, vergleicht compare_obj den tatsächlichen Inhalt der Objekte. Bei test_equal_obj wird hingegen nur bei Regionen der tatsächlichen Inhalt verglichen, während bei allen anderen Objekten nur geprüft wird, ob sie denselben Speicherort besitzen.

Parameter

▷ **Objects1** (input_object) ..... object(-array) ⇝ *object*
Referenzobjekte.

▷ **Objects2** (input_object) ..... object(-array) ⇝ *object*
Vergleichsobjekte.

▷ **Epsilon** (input_control) ..... number ⇝ *real* / integer
Maximal erlaubte Differenz zwischen zwei Grauwerten oder Koordinaten etc.
Defaultwert: 0.0
Wertevorschläge: Epsilon ∈ {0.0, 1.e-5}

▷ **IsEqual** (output_control) ..... integer ⇝ *integer*
Boolescher Ergebniswert.

Ergebnis

`compare_obj` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Unterscheidet sich die Anzahl der Objekte, wird eine Fehlerbehandlung durchgeführt. Ansonsten liefert `compare_obj` den Wert TRUE

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`test_equal_obj`, `test_equal_region`

Modul

Foundation

count_obj (Objects : : : Number)

Anzahl von Objekten in einem Tupel.

`count_obj` bestimmt für den Objektparameter `Objects` die Anzahl der Objekte, die dieser enthält. Dabei ist zu beachten, dass Objekt nicht gleichzusetzen ist mit Zusammenhangskomponente (siehe `connection`). Somit ist beispielsweise die Anzahl der Objekte einer Region, die aus drei nicht zusammenhängenden Teilen besteht, gleich 1.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

▷ **Objects** (input_object) ..... object-array ⇝ *object*
Zu untersuchende Objekte.

▷ **Number** (output_control) ..... integer ⇝ *integer*
Anzahl der Objekte im Tupel `Objects`.

Komplexität

Laufzeitkomplexität: $O(|\texttt{Objects}|)$.

Ergebnis

`count_obj` liefert den Wert TRUE.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`copy_obj`, `obj_to_integer`, `connection`, `set_system`

Modul

Foundation

get_channel_info (Object : : Request, Channel : Information)

Informationen über die Komponenten eines Bildobjektes.

get_channel_info liefert Informationen über die Komponenten eines Bildobjektes. Folgende Abfragen (Request) sind derzeit möglich:

'creator' Ausgabe der Namen der Prozeduren, die die Komponenten (nicht das Objekt) ursprünglich angelegt haben.

'type' Ausgabe des Typs der Bildkomponente ('byte', 'int1', 'int2', 'uint2' 'int4', 'real', 'direction', 'cyclic', 'complex', 'vector_field'). Die Komponente 0 ist vom Typ 'region' oder 'xld'.

Im Tupel Channel werden die Nummern der Komponenten angegeben, über die Informationen gewünscht werden. Information enthält nach Ausführung von get_channel_info ein Tupel von Strings (je ein String pro Eintrag in Channel) mit den gewünschten Informationen.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

▷ **Object** (input_object) ..... object ⇝ *object*
Zu untersuchendes Bildobjekt.

▷ **Request** (input_control) ..... string ⇝ *string*
Gewünschte Information über Objektkomponenten.
Defaultwert: 'creator'
Werteliste: Request ∈ {'creator', 'type'}

▷ **Channel** (input_control) ..... channel(-array) ⇝ *integer*
Zu untersuchende Komponenten (0 für Region/XLD).
Defaultwert: 0
Wertevorschläge: Channel ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8}

▷ **Information** (output_control) ..... string(-array) ⇝ *string*
Angeforderte Information.

Ergebnis

Sind die Parameter korrekt, liefert get_channel_info den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_image

Siehe auch

count_relation

Modul

Foundation

get_obj_class (Object : : : Class)

Name der Klasse eines Bildobjektes.

get_obj_class liefert zu jedem Objekt den Namen der zugehörigen Klasse. Folgende Klassen sind möglich:

'image' Objekt mit Region (Definitionsbereich) und mindestens einem Kanal.

'region' Objekt mit einer Region ohne Grauwerte.

'xld_cont' XLD-Objekt als Kontur

'xld_poly' XLD-Objekt als Polygon

'xld_parallel' XLD-Objekt mit parallelen Polygonen

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

▷ **Object** (input_object) ..... object(-array) ⇝ *object*
Zu untersuchende Bildobjekte.

▷ **Class** (output_control) ..... string(-array) ⇝ *string*
Name der Klasse.

Ergebnis

Sind die Parameter korrekt, liefert `get_obj_class` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`disp_image`, `disp_region`, `disp_xld`

Siehe auch

`get_channel_info`, `count_relation`

Modul

Foundation

test_equal_obj (Objects1, Objects2 : : : IsEqual)

Vergleich von Bildobjekten auf Gleichheit.

`test_equal_obj` vergleicht die Regionen und Grauwertkomponenten aller Objekte der beiden Eingabeparameter. Verglichen wird das n-te Objekt in `Objects1` mit dem n-ten Objekt in `Objects2` (für alle n). Falls alle entsprechenden Regionen gleich sind und auch die Anzahl der Regionen identisch ist, erhält `IsEqual` den Wert TRUE, ansonsten FALSE.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Achtung

Bildmatrizen und XLDs werden nicht auf ihren Inhalt verglichen. Zwei Bilder resp. XLDs sind daher „gleich“, wenn sie an der gleichen Stelle im Speicher stehen. Im Gegensatz dazu wird bei Regionen, die nicht an der gleichen Stelle im Speicher stehen, der tatsächliche Inhalt verglichen. Falls die Eingabeparameter leer sind und das Verhalten mit `set_system(::'no_object_result','true':)` gesetzt wurde, wird `IsEqual` auf TRUE gesetzt, da alle Eingaben (= leere Menge) gleich sind.

Parameter

▷ **Objects1** (input_object) ..... object-array ⇝ *object*
Testobjekte.

▷ **Objects2** (input_object) ..... object-array ⇝ *object*
Vergleichsobjekte.

▷ **IsEqual** (output_control) ..... integer ⇝ *integer*
Boolescher Ergebniswert.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität bei Ergebnis TRUE $O(1)$ oder $O(sqrt(F))$ und bei Ergebnis FALSE $O(sqrt(F))$.

Ergebnis

`test_equal_obj` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system(::`

'no_object_result',<Result>:) festlegen. Unterscheidet sich die Anzahl der Objekte, wird eine Fehlerbehandlung durchgeführt. Ansonsten liefert test_equal_obj den Wert TRUE

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

compare_obj, test_equal_region

Modul

Foundation

21.2 Manipulation

clear_obj (Objects : : :)

Löschen von Bildobjekten aus der HALCON-Datenbank.

clear_obj löscht ikonische Objekte in der Datenbank, die nicht mehr benötigt werden. Es ist zu beachten, dass clear_obj in HALCON/C die einzige Möglichkeit ist, Objekte in der Datenbank zu löschen und damit den Speicherplatz freizugeben. In allen anderen HALCON-Sprachschnittstellen darf clear_obj **nicht** verwendet werden, da die Objekte automatisch durch geeignete Destruktoren freigegeben werden.

Bildmatrizen und Regionen werden von mehreren Bildobjekten gemeinsam verwendet (Speicherplatz!). Dies hat Konsequenzen für das Löschen: Eine Region oder ein Bild werden erst gelöscht, wenn alle Bildobjekte, die sie verwenden gelöscht wurden.

Mit reset_obj_db kann das System zurückgesetzt und alle verbliebenen Bildobjekte gelöscht werden.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Achtung

Zur Verwendung von lokalen Variablen in HALCON/C: Da beim Verlassen einer Prozedur nur die lokalen Variablen beseitigt werden, die Datenbank von HALCON dabei aber nicht modifiziert wird, ist es nötig, vor dem Verlassen der Prozedur lokale Objekte zu löschen.

Parameter

▷ **Objects** (input_object) ..... object(-array) ⇝ *object*
Zu löschende Objekte.

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte in der HALCON-Datenbank vorhanden, dann liefert clear_obj den Wert TRUE. Sind nicht alle Surrogate zulässig (z.B. schon gelöscht), so führt dies zu einer Fehlerbehandlung, die zur Folge hat, dass auch korrekte Surrogate nicht gelöscht werden. Mit dem Aufruf set_check(::'~clear':) wird erreicht, dass bei fehlerhaften Surrogaten keine Fehlerbehandlung erfolgt und alle möglichen Löschungen ausgeführt werden. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

reset_obj_db

Siehe auch

set_check

Modul

Foundation

concat_obj (Objects1, Objects2 : ObjectsConcat : :)

Konkatenieren von zwei Objekttupeln zu einem Tupel von Objekten.

concat_obj konkateniert die Objekttupel Objects1 und Objects2 zu einem neuen Tupel ObjectsConcat. Dieses enthält also alle ikonischen Objekte, die in den beiden Eingabelisten enthalten sind:

ObjectsConcat = [Objects1,Objects2]

In ObjectsConcat werden erst alle Objekte aus Objects1 und dann die Objekte von Objects2 abgelegt. Die Reihenfolge der Objekte wird dabei nicht verändert. Zu beachten ist, dass dabei Referenzen auf die Bilder und Regionen angelegt werden, aber nicht die Bilder und Regionen selber kopiert werden. Das bedeutet, dass Änderungen an Eingabebildern, die z.B. mit set_grayval, overpaint_gray oder overpaint_region durchgeführt werden, auch die Bilder des Ausgabetupels ObjectsConcat betreffen und umgekehrt.

concat_obj ist nicht mit union1 oder union2 zu verwechseln, bei denen die Regionen vereinigt werden, also die Anzahl der Objekte verändert wird.

Mit concat_obj ist es möglich, Objekte verschiedener Bildobjekttypen (z.B. Bilder und XLD-Konturen) zu einem Objekt zu konkatenieren. Dies ist ausschließlich dann sinnvoll, wenn es unumgänglich ist, in einer Objektvariable z.B. alle Ergebnisse einer Bildverarbeitungssequenz aufzusammeln. Dabei ist zu beachten, dass die einzigen Operatoren, die diese Art von gemischten Objekttupeln verarbeiten können, concat_obj, copy_obj, select_obj und disp_obj sind.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

- ▷ **Objects1** (input_object) ..... object(-array) ⇝ *object*
  Objekttupel 1.
- ▷ **Objects2** (input_object) ..... object(-array) ⇝ *object*
  Objekttupel 2.
- ▷ **ObjectsConcat** (output_object) ..... object-array ⇝ *object*
  Enthält (Kopien aller) Objekte aus Objects1 und Objects2.

Beispiel

```
/* generate a tuple of a circle and a rectangle */

gen_circle(&Circle,200.0,400.0,23.0);
gen_rectangle1(&Rectangle,23.0,44.0,203.0,201.0);
concat_obj(Circle,Rectangle,&CirclAndRectangle);
disp_region(CircleAndRectangle,WindowHandle);
```

Komplexität

Laufzeitkomplexität: $O(|\texttt{Objects1}| + |\texttt{Objects2}|)$;

Speicherplatzkomplexität der Ergebnisdaten: $O(|\texttt{Objects1}| + |\texttt{Objects2}|)$

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte in der HALCON-Datenbank vorhanden, dann liefert concat_obj den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`count_obj`, `copy_obj`, `select_obj`, `disp_obj`

Modul

Foundation

`copy_obj` (`Objects` : `ObjectsSelected` : `Index`, `NumObj` :)

Kopieren von Bildobjekten in der HALCON-Datenbank.

`copy_obj` kopiert `NumObj` Bildobjekte ab Position `Index` (beginnend mit 1) aus dem Tupel von Eingabeobjekten in `Objects`. Das resultierende Tupel von Ausgabeobjekten wird in `ObjectsSelected` abgelegt. Wird für `NumObj` der Wert -1 verwendet, dann werden alle Objekte ab `Index` kopiert. Beim Kopieren werden die Region und die Bilder nicht dupliziert. Es werden neue Objekte angelegt, die Verweise auf die Regionen und Bilder enthalten. Die Anzahl von Objekten in einem Tupel kann mit `count_obj` abgefragt werden.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

▷ **`Objects`** (input_object) ..... object(-array) ⇝ *object*
Objekte, die kopiert werden sollen.

▷ **`ObjectsSelected`** (output_object) ..... object(-array) ⇝ *object*
Kopierte Objekte.

▷ **`Index`** (input_control) ..... integer ⇝ *integer*
Ab dieser Position in dem Tupel Object werden Bildobjekte kopiert.
Defaultwert: 1
Wertevorschläge: `Index` ∈ {1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000}
Typischer Wertebereich: 1 ≤ Index
Restriktion: Index <= number(Objects)

▷ **`NumObj`** (input_control) ..... integer ⇝ *integer*
Anzahl der zu kopierenden Bildobjekte oder -1.
Defaultwert: 1
Wertevorschläge: `NumObj` ∈ {-1, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000}
Typischer Wertebereich: -1 ≤ NumObj
Restriktion: NumObj + Index - 1 <= number(Objects) && NumObj != 0

Beispiel

```
gen_circle (Circles, [100,200,400], [200,100,400], [100,100,50])
copy_obj (Circles, CircleCopy, 2, 3)
erosion_circle (CircleCopy, CircleErosion, 15.5)
dev_set_color ('red')
dev_set_draw ('fill')
dev_display(Circles)
dev_set_color ('white')
dev_set_draw ('margin')
dev_display (CircleCopy)
dev_set_color ('green')
dev_display (CircleErosion)
```

Komplexität

Laufzeitkomplexität: $O(|\texttt{Objects}| + \texttt{NumObj})$;

Speicherkomplexität der Ausgabedaten: $O(\texttt{NumObj})$

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte in der HALCON-Datenbank vorhanden und die Parameter korrekt, dann liefert `copy_obj` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`count_obj`

Alternativen

`select_obj`

Siehe auch

`count_obj`, `concat_obj`, `obj_to_integer`, `copy_image`

Modul

Foundation

gen_empty_obj (: EmptyObject : :)

Erzeugen eines leere Objekttupels.

`gen_empty_obj` erzeugt eine leeres Tupel. Dies bedeutet, dass der Ausgabeparameter keine Objekte enthält. `count_obj` liefert also 0. Es kann aber `clear_obj` für die Ausgabe aufgerufen werden. Es ist zu beachten, dass keine Objekte nicht mit einer leeren Region zu verwechseln sind. Bei einer leeren Region, d.h. einer Region mit 0 Bildpunkten liefert `count_obj` den Wert 1.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

▷ **EmptyObject** (output_object) ..... object ⇝ *object*
Keine Objekte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

insert_obj (Objects, ObjectsInsert : ObjectsExtended : Index :)

Einfügen von Bildobjekten in ein ikonisches Objekt-Tupel.

`insert_obj` fügt die Elemente von `ObjectsInsert` in das Object-Tupel `Objects` an der Stelle `Index` ein und gibt das Ergebnis in `ObjectsExtended` zurück.

`Index` muss einen einelementigen Integerwert enthalten. Dabei ist zu beachten, dass die Indizes anders als bei Steuerungs-Tupeln bei Objekt-Tupeln mit 1 beginnen. Die Länge von `ObjectsExtended` ist die Summe der Länge von `Objects` und `ObjectsInsert`. Bilder, Regionen und XLDs können in `Objects` und `ObjectsInsert` gemischt verwendet werden.

Parameter

▷ **Objects** (input_object) ..... object(-array) ⇝ *object*
Eingabe Objekt-Tupel.

▷ **ObjectsInsert** (input_object) ..... object(-array) ⇝ *object*
Einzufügendes Objekt-Tupel.

▷ **ObjectsExtended** (output_object) ..... object(-array) ⇝ *object*
Erweitertes Ausgabe Objekt-Tupel.

▷ **Index** (input_control) ..... number ⇝ *integer*
Einfüge-Index.

Beispiel

```
gen_image_const (Image1, 'byte', 1, 1)
gen_image_const (Image3, 'byte', 3, 3)
concat_obj (Image1, Image3, Images)
gen_image_const (Image, 'byte', 2, 2)
insert_obj (Images, Image, Images, 2)
```

Komplexität

Typische Komplexität der Laufzeit: $O(|\texttt{Objects}| + |\texttt{ObjectsInsert}|)$.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`count_obj`

Nachfolger

`count_obj`

Alternativen

`concat_obj`

Siehe auch

`count_obj`, `select_obj`, `copy_obj`, `remove_obj`

Modul

Foundation

integer_to_obj (: Objects : SurrogateTuple :)

Konversion von „ganzen Zahlen“ zu Surrogaten (Bildobjekten).

`integer_to_obj` arbeitet invers zu `obj_to_integer`. Alle Surrogate auf Steuerparameterpositionen werden als Objekte abgelegt. Es ist zu beachten, dass im Gegensatz zu `obj_to_integer` die Objekte dupliziert werden. `integer_to_obj` ist speziell für HALCON/C, da hier Bildobjekte und Steuerparameter unterschiedlich behandelt werden. Bitte beachten Sie, dass das Program abstürzt, wenn in `SurrogateTuple` der Wert *0* übergeben wird, da dies kein gültiger Zeiger ist.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Achtung

Die Objekte werden in der Datenbank dupliziert.

Parameter

▷ **Objects** (output_object) . . . . . object(-array) ⇝ *object*
Surrogate auf Bildobjektposition.

▷ **SurrogateTuple** (input_control) . . . . . pointer(-array) ⇝ *integer*
Tupel von Surrogaten.

Ergebnis

`integer_to_obj` liefert den Wert TRUE, falls die Parameterwerte korrekt sind, es sich also um Objektschlüssel handelt. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`obj_to_integer`

Modul

Foundation

obj_diff (Objects, ObjectsSub : ObjectsDiff : :)

Mengendifferenz zweier Objekttupel.

`obj_diff` berechnet die Mengendifferenz von Objekttupeln:

(Objekte aus `Objects`) − (Objekte aus `ObjectsSub`)

Das Ergebnistupel `ObjectsDiff` berechnet sich aus dem Eingabetupel `Objects` minus aller Objekte aus `ObjectsSub`.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Achtung

Bildmatrizen und XLDs werden nicht auf ihren Inhalt verglichen. Zwei Bilder resp. XLDs sind daher „gleich“, wenn sie an der gleichen Stelle im Speicher stehen. Im Gegensatz dazu wird bei Regionen, die nicht an der gleichen Stelle im Speicher stehen, der tatsächliche Inhalt verglichen.

Parameter

▷ **Objects** (input_object) . . . . . object(-array) ⇝ *object*
Objekttupel 1.

▷ **ObjectsSub** (input_object) . . . . . object(-array) ⇝ *object*
Objekttupel 2.

▷ **ObjectsDiff** (output_object) . . . . . object(-array) ⇝ *object*
Enthält (Kopien aller) Objekte aus `Objects`, die nicht in `ObjectsSub` enthalten sind.

Ergebnis

`obj_diff` liefert immer den Wert TRUE zurück.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`test_equal_obj`, `count_obj`, `copy_obj`, `select_obj`

Modul

Foundation

obj_to_integer (Objects : : Index, Number : SurrogateTuple)

Konversion von Surrogaten (Bildobjekten) zu „ganzen Zahlen“.

`obj_to_integer` speichert ab der angegebenen Stelle (`Index`) die gewünschte Anzahl von Surrogaten der Objekte (Objektschlüssel) aus `Objects` als Integer-Werte im Ausgabeparameter `SurrogateTuple`. Wird für `Number` der Wert -1 verwendet, dann werden alle Objekte ab `Index` kopiert. Damit wird ein Direktzugriff auf ein beliebiges Element von `Objects` möglich. Im Zusammenhang mit `count_obj` (liefert die Anzahl der in `Objects` enthaltenen Objekte) kann man so eine sukzessive Abarbeitung der einzelnen Objekte in `Objects` durchführen. Die Objekte werden von `obj_to_integer` nicht dupliziert und dürfen deshalb nicht gelöscht (`clear_obj`) werden.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Achtung

Die Daten der Objekte werden nicht dupliziert.

Parameter

- ▷ **Objects** (input_object) ..... object(-array) ⇝ *object*
  Objekte, deren Surrogate ausgegeben werden.
- ▷ **Index** (input_control) ..... integer ⇝ *integer*
  Ab diesem Element werden die Surrogate der Objekte ausgegeben.
  Defaultwert: 1
  Typischer Wertebereich: 1 ≤ Index
- ▷ **Number** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu übergebenden Surrogate.
  Defaultwert: -1
  Restriktion: Number == -1 || Number + Index <= number(Objects)
- ▷ **SurrogateTuple** (output_control) ..... pointer(-array) ⇝ *integer*
  Tupel der Surrogate

Beispiel

```
* Access the i-th element:
obj_to_integer(Objects,i,1,Surrogat)
```

Komplexität

Laufzeitkomplexität: $O(|\texttt{Objects}| + \texttt{Number})$

Ergebnis

`obj_to_integer` liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

copy_obj, select_obj, copy_image, gen_image_proto

Siehe auch

integer_to_obj, count_obj

Modul

Foundation

remove_obj (Objects : ObjectsReduced : Index :)

Entferne Objekte aus einem ikonischen Objekt-Tupel.

remove_obj entfernt die von Index referenzierten Objekte dem Objekt-Tupel Objects und gibt die verbleibenden Objekte in dem Tupel ObjectsReduced zurück. Index bestimmt die Indizes der Elemente, die entfernt werden sollen. Dabei ist zu beachten, dass die Indizes anders als bei Steuerungs-Tupeln bei Objekt-Tupeln mit 1 beginnen. Duplikate und Indizes außerhalb des Bereiches werden ignoriert.

Parameter

- ▷ **Objects** (input_object) ..... object(-array) ⇝ *object*
  Eingabe Objekt-Tupel.
- ▷ **ObjectsReduced** (output_object) ..... object(-array) ⇝ *object*
  Verbleibende Objekt-Tupel.
- ▷ **Index** (input_control) ..... number(-array) ⇝ *integer*
  Indizes der zu entfernenden Objekte.

Beispiel

```
gen_empty_obj (Images)
for Index := 1 to 10 by 1
  gen_image_const (Image, 'byte', Index, Index)
  concat_obj (Images, Image, Images)
endfor

remove_obj (Images, Images, [7, 4])
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

select_obj

Siehe auch

count_obj, select_obj, copy_obj, insert_obj

Modul

Foundation

replace_obj (Objects, ObjectsReplace : Replaced : Index :)

Ersetzen einzelner Elemente aus einem ikonischen Objekt-Tupel.

replace_obj ersetzt einzelne Elemente aus einem ikonischen Objekt-Tupel Objects und gibt das veränderte Tupel mit dem Parameter Replaced zurück. Der Parameter Index enthält hierbei die Indizes der Elemente und ObjectsReplace die entsprechenden ikonischen Elementwerte, die ersetzt werden. Folglich darf Index

ausschließlich ganze Zahlen beinhalten (falls Index eine Gleitkommazahl enthält, so muss diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Ikonische Tupelindizes werden beginnend mit 1 gezählt, das heißt das erste Tupelelement wird über den Index 1 ausgewählt. Falls ObjectsReplace nur ein Objekt enthält, dann wird dieses an allen in Index angegebenen Indizes eingesetzt. Falls ein Indexwert größer als die Tupellänge des Eingabeobjektes ist, so wird das Ergebnistupel Replaced entsprechend verlängert und mit leeren Regionen aufgefüllt.

Sonderfall: Leere Eingabewerte

Ist einer der beiden Eingabeparameter Index und ObjectsReplace leer und das andere nicht, wird eine Fehlermeldung angezeigt. Sind beide leer, entspricht das Ausgabeobjekt Replaced dem Eingabeobjekt Objects. Sind beide nicht leer, aber das Eingabeobjekt Objects ist leer, wird das leere Objekt entsprechend der oben beschriebenen Vorgehensweise erweitert.

Parameter

- ▷ **Objects** (input_object) ..... object(-array) ⇝ *object*
  Ikonisches Eingabeobjekt.
- ▷ **ObjectsReplace** (input_object) ..... object(-array) ⇝ *object*
  Ikonisches Objekt-Tupel mit den einzusetzenden Elementen.
- ▷ **Replaced** (output_object) ..... object(-array) ⇝ *object*
  Ikonisches Ausgabe-Tupel mit den ersetzten Elementen.
- ▷ **Index** (input_control) ..... number(-array) ⇝ *integer*
  Indizes der zu ersetzenden Elemente.

Beispiel

```
gen_empty_obj (Images)
for Index := 1 to 10 by 1
  gen_image_const (Image, 'byte', Index, Index)
  concat_obj (Images, Image, Images)
endfor
gen_empty_obj (Replace)

replace_obj (Images, Replace, Images, 1)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

select_obj

Siehe auch

count_obj, select_obj, copy_obj, insert_obj

Modul

Foundation

select_obj (Objects : ObjectSelected : Index :)

Selektieren von Bildobjekten.

select_obj kopiert die Bildobjekte an den Positionen, die durch Index (beginnend mit 1) gegeben sind, aus dem Tupel von Eingabeobjekten in Objects. Beim Kopieren werden die Region und die Bilder nicht dupliziert. Es werden neue Objekte angelegt, die Verweise auf die Regionen und Bilder enthalten. Die Anzahl von Objekten in einem Tupel kann mit count_obj abgefragt werden.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

▷ **Objects** (input_object) ..... object(-array) ⇝ *object*
Eingabeobjekte.

▷ **ObjectSelected** (output_object) ..... object(-array) ⇝ *object*
Selektierte Objekte.

▷ **Index** (input_control) ..... integer(-array) ⇝ *integer*
Indizes der Objekte, die selektiert werden sollen.
Defaultwert: 1
Wertevorschläge: `Index` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 50, 100, 200, 500, 1000, 2000, 5000}
Restriktion: Index >= 1

Beispiel

```
/* Access to all Regions */

count_obj(Regions,&Num);
for (i=1; i<=Num; i++)
{
  select_obj(Regions,&Single,i);
  T_get_region_polygon(Single,5.0,&Row,&Column);
  T_disp_polygon(WindowHandleTuple,Row,Column);
  destroy_tuple(Row);
  destroy_tuple(Column);
}
```

Komplexität

Laufzeitkomplexität: $O(|\texttt{Objects}|)$

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte in der HALCON-Datenbank vorhanden und die Parameter korrekt, dann liefert `select_obj` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Unterstützt Objekte auf Compute Devices.
- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`count_obj`

Alternativen

`copy_obj`

Siehe auch

`count_obj`, `concat_obj`, `obj_to_integer`

Modul

Foundation

Kapitel 22

Regionen

22.1 Generierung

gen_checker_region (: RegionChecker : WidthRegion, HeightRegion, WidthPattern, HeightPattern :)

Erzeugen einer schachbrettartigen Region.

gen_checker_region liefert eine schachbrettartige Region. Dabei gehören jeweils die schwarzen Felder des Schachbretts zur Region. Die horizontale und vertikale Ausdehnung der Region wird durch WidthRegion bzw. HeightRegion begrenzt, die Größe der Felder des Schachbrettes durch WidthPattern × HeightPattern.

Achtung

Wird das Muster sehr klein gewählt (WidthPattern < 4) und die Begrenzung groß, so benötigt die erzeugte Region viel Speicher.

Parameter

- ▷ **RegionChecker** (output_object) ..... region ⇝ *object*
  Erzeugte Schachbrett-Region.
- ▷ **WidthRegion** (input_control) ..... extent.x ⇝ *integer*
  Größter auftretender x-Wert der Region.
  Defaultwert: 511
  Wertevorschläge: WidthRegion ∈ {10, 20, 31, 63, 127, 255, 300, 400, 511}
  Typischer Wertebereich: $1 \leq$ WidthRegion ≤ 1024 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: WidthRegion >= 1
- ▷ **HeightRegion** (input_control) ..... extent.y ⇝ *integer*
  Größter auftretender y-Wert der Region.
  Defaultwert: 511
  Wertevorschläge: HeightRegion ∈ {10, 20, 31, 63, 127, 255, 300, 400, 511}
  Typischer Wertebereich: $1 \leq$ HeightRegion ≤ 1024 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: HeightRegion >= 1
- ▷ **WidthPattern** (input_control) ..... extent.x ⇝ *integer*
  Breite eines Feldes des Schachbrettes.
  Defaultwert: 64
  Wertevorschläge: WidthPattern ∈ {1, 2, 4, 8, 16, 20, 32, 64, 100, 128, 200, 300, 500}
  Typischer Wertebereich: $1 \leq$ WidthPattern ≤ 1024 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: WidthPattern > 0 && WidthPattern < WidthRegion

▷ **HeightPattern** (input_control) . . . . . extent.y ⇝ *integer*
Höhe eines Feldes des Schachbrettes.
Defaultwert: 64
Wertevorschläge: HeightPattern ∈ {1, 2, 4, 8, 16, 20, 32, 64, 100, 128, 200, 300, 500}
Typischer Wertebereich: 1 ≤ HeightPattern ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: HeightPattern > 0 && HeightPattern < HeightRegion

Beispiel

```
gen_checker_region(Checker,512,512,32,64)
dev_set_draw('fill')
dev_display (Checker)
```

Komplexität

Der benötigte Speicher (in Byte) für die Region ist:

$O((\texttt{WidthRegion} * \texttt{HeightRegion})/\texttt{WidthPattern})$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_checker_region den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'2 (H_MSG_TRUE) festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

paint_region

Alternativen

gen_grid_region, gen_region_polygon_filled, gen_region_points, gen_region_runs, gen_rectangle1, concat_obj, gen_random_region, gen_random_regions

Siehe auch

reduce_domain

Modul

Foundation

gen_circle (: Circle : Row, Column, Radius :)

Erzeugen eines Kreises.

gen_circle generiert einen oder mehrere Kreise, die durch Mittelpunkt und Radius beschrieben werden. Sollen mehrere Kreise erzeugt werden, sind die Koordinaten in Form von Tupeln zu übergeben.

gen_circle erzeugt ausschließlich symmetrische Kreise. Dazu wird intern der Radius auf ein Vielfaches von 0.5 abgerundet. Wird für den Radius ein ganzzahliger Wert (1, 2, 3, ...) angegeben, so ergibt sich ein geradzahliger Durchmesser und damit ein Kreis, der nur bezüglich eines Mittelpunktes mit Nachkommaanteil 0.5 symmetrisch sein kann. Intern werden also in diesem Fall die Koordinaten des Mittelpunktes auf die nächstgelegenen Koordinaten mit einem Nachkommaanteil von 0.5 angepasst. Hierbei werden ganzzahlige Koordinaten auf die nächstkleineren Koordinaten mit einem Nachkommaanteil von 0.5 abgerundet. Bei ungeradzahligem Durchmesser (Radius = 1.5, 2.5, 3.5, ...) erhält man einen Kreis, der nur bezüglich eines Mittelpunktes mit ganzzahligen Koordinaten symmetrisch sein kann. Intern werden also in diesem Fall die Koordinaten des Mittelpunktes auf die nächstgelegenen ganzzahligen Koordinaten gerundet. Es ist zu beachten, dass durch den obigen Algorithmus Kreise mit einem geradzahliger Durchmesser *nicht* in Kreisen mit dem nächstgrößeren ungeradzahligem Durchmesser enthalten sein müssen, selbst wenn die in Row und Column angegebenen Koordinaten identisch sind.

Falls der Kreis über den Bildrand reicht, wird je nach Wert des Systemflags 'clip_region' (set_system) der Kreis auf das aktuelle Bildformat beschnitten.

Achtung

Aus Geschwindigkeitsgründen kann die Ergebnisregion zusätzliche Pixel am Rand enthalten, und einige einzelne Pixel am Rand können fehlen. Dies kann zu einer Inkonsistenz zwischen den Operatoren smallest_circle und gen_circle führen.

Parameter

- ▷ **Circle** (output_object) ..... region(-array) ⇝ *object*
  Erzeugter Kreis.
- ▷ **Row** (input_control) ..... circle.center.y(-array) ⇝ *real* / integer
  Zeilenindex des Schwerpunktes.
  Defaultwert: 200.0
  Wertevorschläge: Row ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}
  Typischer Wertebereich: 1.0 ≤ Row ≤ 1024.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 10.0
- ▷ **Column** (input_control) ..... circle.center.x(-array) ⇝ *real* / integer
  Spaltenindex des Schwerpunktes.
  Defaultwert: 200.0
  Wertevorschläge: Column ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}
  Typischer Wertebereich: 1.0 ≤ Column ≤ 1024.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 10.0
- ▷ **Radius** (input_control) ..... circle.radius(-array) ⇝ *real* / integer
  Radius des Kreises.
  Defaultwert: 100.5
  Wertevorschläge: Radius ∈ {1.0, 1.5, 2.0, 2.5, 3, 3.5, 4, 4.5, 5.5, 6.5, 7.5, 9.5, 11.5, 15.5, 20.5, 25.5, 31.5, 50.5}
  Typischer Wertebereich: 1.0 ≤ Radius ≤ 1024.0 (lin)
  Restriktion: Radius > 0.0

Beispiel

```
read_image(Image,'fabrik')
gen_circle(Circle,300.0,200.0,150.5)
reduce_domain(Image,Circle,Mask)
dev_clear_window ()
dev_display (Mask)
```

Komplexität

Laufzeitkomplexität: $O(\text{Radius} * 2)$

Speicherplatzkomplexität (Byte): $O(\text{Radius} * 8)$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_circle den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'2 (H_MSG_TRUE) festgelegt. Entsteht durch Clipping (der Kreis liegt völlig außerhalb des Bildformats) eine leere Region, dann legt set_system('store_empty_region',<'2 (H_MSG_TRUE) fest, ob die leere Region ausgegeben wird.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

paint_region, reduce_domain

Alternativen

gen_ellipse, gen_region_polygon_filled, gen_region_points, gen_region_runs, draw_circle

Siehe auch

disp_circle, set_shape, smallest_circle, reduce_domain

Modul

Foundation

gen_circle_sector (: CircleSector : Row, Column, Radius, StartAngle, EndAngle :)

Erzeugen eines Kreisausschnitts.

gen_circle_sector generiert einen oder mehrere Kreisschnitte, die durch Mittelpunkt und Radius beschrieben werden. Sollen mehrere Kreise erzeugt werden, sind die Koordinaten in Form von Tupeln zu übergeben.

gen_circle_sector erzeugt ausschließlich symmetrische Kreise. Dazu wird intern der Radius auf ein Vielfaches von 0.5 abgerundet. Wird für den Radius ein ganzzahliger Wert (1, 2, 3, ...) angegeben, so ergibt sich ein geradzahliger Durchmesser und damit ein Kreis, der nur bezüglich eines Mittelpunktes mit Nachkommaanteil 0.5 symmetrisch sein kann. Intern werden also in diesem Fall die Koordinaten des Mittelpunktes auf die nächstgelegenen Koordinaten mit einem Nachkommaanteil von 0.5 angepasst. Hierbei werden ganzzahlige Koordinaten auf die nächstkleineren Koordinaten mit einem Nachkommaanteil von 0.5 abgerundet. Bei ungeradzahligem Durchmesser (Radius = 1.5, 2.5, 3.5, ...) erhält man einen Kreis, der nur bezüglich eines Mittelpunktes mit ganzzahligen Koordinaten symmetrisch sein kann. Intern werden also in diesem Fall die Koordinaten des Mittelpunktes auf die nächstgelegenen ganzzahligen Koordinaten gerundet. Es ist zu beachten, dass durch den obigen Algorithmus Kreise mit einem geradzahliger Durchmesser *nicht* in Kreisen mit dem nächstgrößeren ungeradzahligem Durchmesser enthalten sein müssen, selbst wenn die in Row und Column angegebenen Koordinaten identisch sind.
Die Winkel werden in Bogenmaß in mathematisch positivem Drehsinn angegeben. Dies wird in der unten folgenden Illustration veranschaulicht. Der Wert *'rad(360)'* ist äquivalent zu *0*. Entsprechend ergibt ein Sektor mit StartAngle = *0* und EndAngle = *'rad(360)'* eine leere Region.

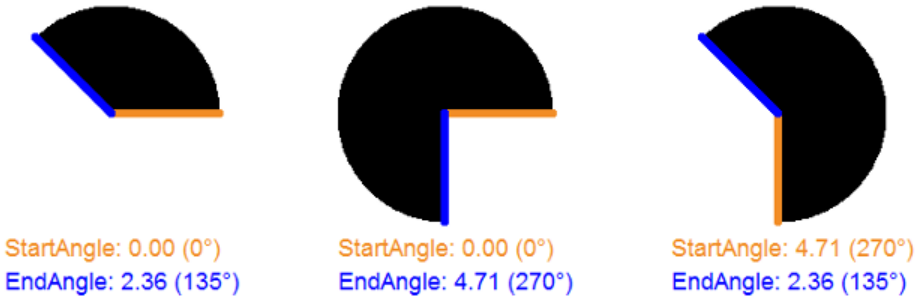


Illustration verschiedener Winkelkombinationen.

Falls der Kreis über den Bildrand reicht, wird je nach Wert des Systemflags 'clip_region' (set_system) der Kreis auf das aktuelle Bildformat beschnitten.

Parameter

▷ **CircleSector** (output_object) ..... region(-array) ⇝ *object*
Erzeugter Kreisausschnitt.

▷ **Row** (input_control) ..... coordinates.y(-array) ⇝ *real* / integer
Zeilenindex des Schwerpunktes.
Defaultwert: 200.0
Wertevorschläge: Row ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}
Typischer Wertebereich: 1.0 ≤ Row ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Column** (input_control) . . . . . coordinates.x(-array) ⇝ *real* / integer
Spaltenindex des Schwerpunktes.
Defaultwert: 200.0
Wertevorschläge: Column ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}
Typischer Wertebereich: 1.0 ≤ Column ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Radius** (input_control) . . . . . number(-array) ⇝ *real* / integer
Radius des Kreises.
Defaultwert: 100.5
Wertevorschläge: Radius ∈ {1.0, 1.5, 2.0, 2.5, 3, 3.5, 4, 4.5, 5.5, 6.5, 7.5, 9.5, 11.5, 15.5, 20.5, 25.5, 31.5, 50.5}
Typischer Wertebereich: 1.0 ≤ Radius ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Radius > 0.0

▷ **StartAngle** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Startwinkel des Kreisausschnittes.
Defaultwert: 0.0
Wertevorschläge: StartAngle ∈ {0.0, 0.785398, 1.5708, 2.35619, 3.14159, 3.92699, 4.71239, 5.49779, 6.28318}
Typischer Wertebereich: 0 ≤ StartAngle ≤ 6.28318 (lin)
Restriktion: 0 <= StartAngle && StartAngle <= 2 * pi

▷ **EndAngle** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Endwinkel des Kreisausschnittes.
Defaultwert: 3.14159
Wertevorschläge: EndAngle ∈ {0.0, 0.785398, 1.5708, 2.35619, 3.14159, 3.92699, 4.71239, 5.49779, 6.28318}
Typischer Wertebereich: 0 ≤ EndAngle ≤ 6.28318 (lin)
Restriktion: 0 <= EndAngle && EndAngle <= 2 * pi

Beispiel

```
read_image(Image,'fabrik')
gen_circle_sector(CircleSector,300.0,200.0,150.5,0,rad(120))
reduce_domain(Image,CircleSector,Mask)
dev_clear_window ()
dev_display (Mask)
```

Komplexität

Laufzeitkomplexität: $O(\texttt{Radius} * 2)$

Speicherplatzkomplexität (Byte): $O(\texttt{Radius} * 8)$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_circle den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'2 (H_MSG_TRUE) festgelegt. Entsteht durch Clipping (der Kreis liegt völlig außerhalb des Bildformats) eine leere Region, dann legt set_system('store_empty_region',<'2 (H_MSG_TRUE) fest, ob die leere Region ausgegeben wird.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

paint_region, reduce_domain

Alternativen

gen_ellipse, gen_ellipse_sector, gen_region_polygon_filled, gen_region_points, gen_region_runs, draw_circle

Siehe auch

disp_circle, disp_region, set_shape, smallest_circle, reduce_domain

Modul

Foundation

gen_ellipse (: Ellipse : Row, Column, Phi, Radius1, Radius2 :)

Erzeugen einer Ellipse.

gen_ellipse generiert eine oder mehrere Ellipsen mit dem Schwerpunkt (Row, Column), der Orientierung Phi und den Halbradien Radius1 und Radius2. Der Winkel wird in Bogenmaß bzgl. der x-Achse in mathematisch positivem Drehsinn angegeben. Es kann mehr als eine Region erzeugt werden, indem Tupel von Parameterwerten übergeben werden.

Der Schwerpunkt muss innerhalb der Bildkoordinaten liegen. Das Koordinatensystem läuft von (0,0) (linkes oberes Eck) bis (Width-1,Height-1). Siehe hierzu auch get_system und reset_obj_db. Falls die Ellipse über den Bildrand reicht, wird je nach dem Wert des Systemflags 'clip_region' (set_system) die Ellipse auf das aktuelle Bildformat beschnitten.

Parameter

▷ **Ellipse** (output_object) . . . . . region(-array) ⇝ *object*
Erzeugte Ellipse(n).

▷ **Row** (input_control) . . . . . ellipse.center.y(-array) ⇝ *real* / integer
Zeilenindex des Schwerpunktes.
Defaultwert: 200.0
Wertevorschläge: Row ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich: 1.0 ≤ Row ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Column** (input_control) . . . . . ellipse.center.x(-array) ⇝ *real* / integer
Spaltenindex des Schwerpunktes.
Defaultwert: 200.0
Wertevorschläge: Column ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich: 1.0 ≤ Column ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Phi** (input_control) . . . . . ellipse.angle.rad(-array) ⇝ *real* / integer
Orientierung des längeren Radius (Radius1).
Defaultwert: 0.0
Wertevorschläge: Phi ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
Typischer Wertebereich: -1.178097 ≤ Phi ≤ 1.178097 (lin)

▷ **Radius1** (input_control) . . . . . ellipse.radius1(-array) ⇝ *real* / integer
Längerer Radius.
Defaultwert: 100.0
Wertevorschläge: Radius1 ∈ {2.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich: 1.0 ≤ Radius1 ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Radius1 > 0

▷ **Radius2** (input_control) . . . . . ellipse.radius2(-array) ⇝ *real* / integer
Kürzerer Radius.
Defaultwert: 60.0
Wertevorschläge: Radius2 ∈ {1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich: 1.0 ≤ Radius2 ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Radius2 > 0 && Radius2 <= Radius1

Beispiel

```
read_image (Image, 'fabrik')
gen_ellipse (Ellipse, 200, 200, rad(10), 160, 90)
reduce_domain (Image, Ellipse, ImageReduced)
dev_clear_window ()
dev_display (ImageReduced)
```

Komplexität

Laufzeitkomplexität: $O(\text{Radius1} * 2)$

Speicherplatzkomplexität (Byte): $O(\text{Radius1} * 8)$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_ellipse den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'2 (H_MSG_TRUE) festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

paint_region, reduce_domain

Alternativen

gen_circle, gen_region_polygon_filled, draw_ellipse

Siehe auch

disp_ellipse, set_shape, smallest_circle, reduce_domain

Modul

Foundation

gen_ellipse_sector (: EllipseSector : Row, Column, Phi, Radius1, Radius2, StartAngle, EndAngle :)

Erzeugen eines Ellipsenausschnittes.

gen_ellipse_sector generiert eine oder mehrere Ellipsen mit dem Schwerpunkt (Row, Column), der Orientierung Phi, den Halbradien Radius1 und Radius2 und den Start- und End-winkeln StartAngle and EndAngle. Der Winkel wird in Bogenmaß in mathematisch positivem Drehsinn angegeben. Eine Illustration zu den Winkeln kann bei dem Operator gen_circle_sector gefunden werden. Der Wert *'rad(360)'* ist äquivalent zu *0*. Entsprechend ergibt ein Sektor mit StartAngle = *0* und EndAngle = *'rad(360)'* eine leere Region. Es kann mehr als eine Region erzeugt werden, indem Tupel von Parameterwerten übergeben werden.

Der Schwerpunkt muss innerhalb der Bildkoordinaten liegen. Das Koordinatensystem läuft von (0,0) (linkes oberes Eck) bis (Width-1,Height-1). Siehe hierzu auch get_system und reset_obj_db. Falls die Ellipse über den Bildrand reicht, wird je nach dem Wert des Systemflags 'clip_region' (set_system) die Ellipse auf das aktuelle Bildformat beschnitten.

Parameter

▷ **EllipseSector** (output_object) . . . . . region(-array) ⇝ *object*
Erzeugte Ellipse(n).

▷ **Row** (input_control) . . . . . coordinates.y(-array) ⇝ *real* / integer
Zeilenindex des Schwerpunktes.
Defaultwert: 200.0
Wertevorschläge: `Row` ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich: 1.0 ≤ `Row` ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Column** (input_control) . . . . . coordinates.x(-array) ⇝ *real* / integer
Spaltenindex des Schwerpunktes.
Defaultwert: 200.0
Wertevorschläge: `Column` ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich: 1.0 ≤ `Column` ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Phi** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Orientierung des längeren Radius (Radius1).
Defaultwert: 0.0
Wertevorschläge: `Phi` ∈ {-1.5707, -1.1781, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.1781, 1.5707}
Typischer Wertebereich: -1.570796 ≤ `Phi` ≤ 1.570796 (lin)
Restriktion: - pi / 2 <= Phi <= pi / 2

▷ **Radius1** (input_control) . . . . . number(-array) ⇝ *real* / integer
Längerer Radius.
Defaultwert: 100.0
Wertevorschläge: `Radius1` ∈ {2.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich: 1.0 ≤ `Radius1` ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Radius1 > 0

▷ **Radius2** (input_control) . . . . . number(-array) ⇝ *real* / integer
Kürzerer Radius.
Defaultwert: 60.0
Wertevorschläge: `Radius2` ∈ {1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich: 1.0 ≤ `Radius2` ≤ 1024.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Radius2 > 0 && Radius2 <= Radius1

▷ **StartAngle** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Startwinkel des Ellipsenausschnittes.
Defaultwert: 0.0
Wertevorschläge: `StartAngle` ∈ {0.0, 0.785398, 1.5708, 2.35619, 3.14159, 3.92699, 4.71239, 5.49779, 6.28318}
Typischer Wertebereich: 0 ≤ `StartAngle` ≤ 6.28318 (lin)
Restriktion: 0 <= StartAngle && StartAngle <= 2 * pi

▷ **EndAngle** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Endwinkel des Ellipsenausschnittes.
Defaultwert: 3.14159
Wertevorschläge: `EndAngle` ∈ {0.0, 0.785398, 1.5708, 2.35619, 3.14159, 3.92699, 4.71239, 5.49779, 6.28318}
Typischer Wertebereich: 0 ≤ `EndAngle` ≤ 6.28318 (lin)
Restriktion: 0 <= EndAngle && EndAngle <= 2 * pi

Beispiel

```
read_image (Image, 'fabrik')
gen_ellipse_sector (EllipseSector, 200, 200, rad(30),\
```

```
                    150, 90, rad(45), rad(280))
reduce_domain (Image, EllipseSector, ImageReduced)
dev_clear_window ()
dev_display (ImageReduced)
```

Komplexität

Laufzeitkomplexität: $O(\texttt{Radius1} * 2)$

Speicherplatzkomplexität (Byte): $O(\texttt{Radius1} * 8)$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_ellipse den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'2 (H_MSG_TRUE) festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

paint_region, reduce_domain

Alternativen

gen_circle, gen_circle_sector, gen_region_polygon_filled, draw_ellipse, gen_ellipse

Siehe auch

disp_ellipse, set_shape, smallest_circle, reduce_domain

Modul

Foundation

gen_empty_region (: EmptyRegion : :)

Erzeugen eine leere Region.

gen_empty_region erzeugt eine leere Region. Dies bedeutet, dass der Ausgabeparameter ein Objekt enthält. count_obj liefert also 1. Die Fläche der Region ist 0. Die meisten Formmerkmale sind undefiniert (0). Es ist zu beachten, dass eine leere Region nicht mit dem leeren Tupel zu verwechseln ist.

Parameter

▷ **EmptyRegion** (output_object) ..... region ⇝ *object*
Leere Region (keine Punkte).

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

gen_grid_region (: RegionGrid : RowSteps, ColumnSteps, Type, Width, Height :)

Erzeugen einer Region aus Linien oder Punkten.

gen_grid_region erzeugt ein Raster, das aus Linien (Type = 'lines') oder Punkten (Type = 'points') aufgebaut ist. Bei 'lines' werden durchgezogene Linien bei 'points' nur die Schnittpunkte der Linien ausgegeben. Ausgehend von dem Punkt (0,0) bis zum Punkt (Height-1,Width-1) wird das Raster in der Schrittweite RowSteps in Zeilenrichtung und ColumnSteps in Spaltenrichtung aufgebaut. Die Schrittweiten müssen größer als *1* sein. Ausnahmen sind im Modus 'lines', in dem RowSteps bzw. ColumnSteps auf *0* gesetzt werden kann. In diesem Fall werden nur Spalten bzw. Zeilen erzeugt.

Achtung

Wird das Muster sehr klein gewählt (RowSteps $<$ 4 or ColumnSteps $<$ 4), so benötigt die erzeugte Region viel Speicher.

Im Modus 'points' dürfen RowSteps und ColumnSteps nicht auf *0* gesetzt werden.

Parameter

▷ **RegionGrid** (output_object) . . . . . region ⇝ *object*
Erzeugte Linien/Punkte-Region.

▷ **RowSteps** (input_control) . . . . . extent.y ⇝ *integer* / real
Schrittweite in Zeilenrichtung oder Null.
Defaultwert: 10
Wertevorschläge: RowSteps ∈ {0, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 100}
Typischer Wertebereich: 0 ≤ RowSteps ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: RowSteps > 1 || RowSteps == 0

▷ **ColumnSteps** (input_control) . . . . . extent.x ⇝ *integer* / real
Schrittweite in Spaltenrichtung oder Null.
Defaultwert: 10
Wertevorschläge: ColumnSteps ∈ {0, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 100}
Typischer Wertebereich: 0 ≤ ColumnSteps ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: ColumnSteps > 1 || ColumnSteps == 0

▷ **Type** (input_control) . . . . . string ⇝ *string*
Art des erzeugten Musters.
Defaultwert: 'lines'
Werteliste: Type ∈ {'lines', 'points'}

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Maximale Breite des Musters.
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width >= 1

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Maximale Höhe des Musters.
Defaultwert: 512
Wertevorschläge: Height ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height >= 1

Beispiel

```
read_image(Image,'fabrik')
gen_grid_region(Raster,10,10,'lines',512,512)
reduce_domain(Image,Raster,Mask)
sobel_amp(Mask,GridSobel,'sum_abs',3)
dev_display(GridSobel)
```

Komplexität

Der benötigte Speicher (in Byte) für die Region ist:

$O((ImageWidth/ColumnSteps) * (ImageHeight/RowSteps))$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_grid_region` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch `set_system ('clip_region',<'2 (H_MSG_TRUE)` festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`reduce_domain`, `paint_region`

Alternativen

`gen_region_line`, `gen_region_polygon`, `gen_region_points`, `gen_region_runs`

Siehe auch

`gen_checker_region`, `reduce_domain`

Modul

Foundation

gen_random_region (: RegionRandom : Width, Height :)

Erzeugen einer Zufallsregion.

`gen_random_region` liefert eine Zufallsregion. Dabei wird im Bildbereich $[0 \ldots$ `Width` $-1][0 \ldots$ `Height` $-1]$ jeder Punkt mit Wahrscheinlichkeit 0.5 in die Region aufgenommen. Man kann sich die erzeugte Region als die Schwellenwertbildung in einem Bild mit Rauschen vorstellen.

Diese Prozedur ist insbesondere wichtig für die Erzeugung von unkorrelierten Binärmustern. Das Zufallsmuster wird mit der C-Funktion „nrand48()“ erzeugt.

Achtung

Wenn `Width` und `Height` groß gewählt werden (> 100), dann kann die erzeugte Region durch die intern verwendete Lauflängenkodierung viel Speicherplatz benötigen. Die Grauwerte der Ausgaberegion sind undefiniert.

Parameter

▷ **RegionRandom** (output_object) . . . . . region ⇝ *object*
Erzeugte Zufallsregion mit Ausdehnung `Width` x `Height`.

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Maximale horizontale Ausdehnung der Zufallsregion.
Defaultwert: 128
Wertevorschläge: `Width` ∈ {16, 32, 50, 64, 100, 128, 256, 300, 400, 512}
Typischer Wertebereich: $1 \leq$ `Width` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width > 0

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Maximale vertikale Ausdehnung der Zufallsregion.
Defaultwert: 128
Wertevorschläge: `Height` ∈ {16, 32, 50, 64, 100, 128, 256, 300, 400, 512}
Typischer Wertebereich: $1 \leq$ `Height` ≤ 512 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height > 0

Komplexität

Der schlimmste Fall für die Speicherplatzkomplexität für die erzeugte Region (in Byte) ist: $O(Width*Height*2)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_random_region` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch `set_system ('clip_region',<'2 (H_MSG_TRUE)` festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`paint_region`, `reduce_domain`

Siehe auch

`gen_checker_region`, `add_noise_distribution`, `add_noise_white`, `reduce_domain`

Modul

Foundation

gen_random_regions (: Regions : Type, WidthMin, WidthMax, HeightMin, HeightMax, PhiMin, PhiMax, NumRegions, Width, Height :)

Erzeugen zufälliger Regionen wie Kreise, Rechtecke und Ellipsen.

`gen_random_regions` dient zum Erzeugen von Kreisen, Rechtecken und Ellipsen, deren Parameter zufällig bestimmt werden. Es wird jeweils nur eine Unter- bzw. Obergrenze angegeben. Die Position ist immer zufällig und kann nicht über Parameter bestimmt werden. Der Parameter `NumRegions` gibt an wie viele Regionen erzeugt werden sollen.

Parameter

▷ **Regions** (output_object) . . . . . region-array ⇝ *object*
Erzeugte Regionen.

▷ **Type** (input_control) . . . . . string ⇝ *string*
Art der zu erzeugenden Regionen.
Defaultwert: 'circle'
Werteliste: `Type` ∈ {'circle', 'ring', 'ellipse', 'rectangle1', 'rectangle2'}

▷ **WidthMin** (input_control) . . . . . number ⇝ *real* / integer
Minimale Breite der Region.
Defaultwert: 10.0
Wertevorschläge: `WidthMin` ∈ {1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0}
Typischer Wertebereich: 1.0 ≤ `WidthMin` ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: WidthMin > 0

▷ **WidthMax** (input_control) . . . . . number ⇝ *real* / integer
Maximale Breite der Region.
Defaultwert: 20.0
Wertevorschläge: `WidthMax` ∈ {1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0}
Typischer Wertebereich: 1.0 ≤ `WidthMax` ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: WidthMax > 0

▷ **HeightMin** (input_control) . . . . . number ⇝ *real* / integer
Minimale Höhe der Region.
Defaultwert: 10.0
Wertevorschläge: HeightMin ∈ {1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0}
Typischer Wertebereich: 1.0 ≤ HeightMin ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: HeightMin > 0

▷ **HeightMax** (input_control) . . . . . number ⇝ *real* / integer
Maximale Höhe der Region.
Defaultwert: 30.0
Wertevorschläge: HeightMax ∈ {1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0}
Typischer Wertebereich: 1.0 ≤ HeightMax ≤ 511.0 (lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: HeightMax > 0

▷ **PhiMin** (input_control) . . . . . number ⇝ *real* / integer
Minimaler Rotationswinkel der Region.
Defaultwert: -0.7854
Wertevorschläge: PhiMin ∈ {0.0, 0.1, 0.3, 0.6, 0.9, 1.2, 1.5}
Typischer Wertebereich: 0.0 ≤ PhiMin ≤ 6.28 (lin)
Restriktion: PhiMin > 0

▷ **PhiMax** (input_control) . . . . . number ⇝ *real* / integer
Maximaler Rotationswinkel der Region.
Defaultwert: 0.7854
Wertevorschläge: PhiMax ∈ {0.0, 0.1, 0.3, 0.6, 0.9, 1.2, 1.5}
Typischer Wertebereich: 0.0 ≤ PhiMax ≤ 6.28 (lin)
Restriktion: PhiMax > 0

▷ **NumRegions** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Regionen.
Defaultwert: 100
Wertevorschläge: NumRegions ∈ {1, 5, 20, 100, 200, 500, 1000, 2000}
Typischer Wertebereich: 1 ≤ NumRegions ≤ 2000 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: NumRegions > 0

▷ **Width** (input_control) . . . . . integer ⇝ *integer*
Maximale horizontale Ausdehnung.
Defaultwert: 512
Wertevorschläge: Width ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Width > 0

▷ **Height** (input_control) . . . . . integer ⇝ *integer*
Maximale vertikale Ausdehnung.
Defaultwert: 512
Wertevorschläge: Height ∈ {128, 256, 512, 1024}
Typischer Wertebereich: 1 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: Height > 0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_random_regions den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'2 (H_MSG_TRUE) festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`paint_region`

Modul

Foundation

gen_rectangle1 (: Rectangle : Row1, Column1, Row2, Column2 :)

Erzeugen eines Rechtecks parallel zu den Koordinatenachsen.

`gen_rectangle1` generiert ein oder mehrere Rechtecke parallel zu den Koordinatenachsen, welche durch das linke obere Eck (`Row1`, `Column1`) und das rechte untere Eck (`Row2`, `Column2`) beschrieben werden. Es kann mehr als eine Region erzeugt werden, indem ein Tupel von Eckpunkten übergeben wird. Das Koordinatensystem läuft von (0,0) (linkes oberes Eck) bis (Width-1,Height-1). Siehe hierzu auch `get_system` und `reset_obj_db`.

Parameter

▷ **Rectangle** (output_object) ..... region(-array) ⇝ *object*
Erzeugtes Rechteck.

▷ **Row1** (input_control) ..... rectangle.origin.y(-array) ⇝ *real* / integer
Zeile des linken oberen Eckpunkts.
Defaultwert: 30.0
Wertevorschläge: `Row1` ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 200.0}
(lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Column1** (input_control) ..... rectangle.origin.x(-array) ⇝ *real* / integer
Spalte des linken oberen Eckpunkts.
Defaultwert: 20.0
Wertevorschläge: `Column1` ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 200.0}
(lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Row2** (input_control) ..... rectangle.corner.y(-array) ⇝ *real* / integer
Zeile des rechten unteren Eckpunkts.
Defaultwert: 100.0
Wertevorschläge: `Row2` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0, 511.0}
(lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Row2 >= Row1

▷ **Column2** (input_control) ..... rectangle.corner.x(-array) ⇝ *real* / integer
Spalte des rechten unteren Eckpunkts.
Defaultwert: 200.0
Wertevorschläge: `Column2` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0, 511.0}
(lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0
Restriktion: Column2 >= Column1

Beispiel

```
* Contrast improvement in a rectangular region of interest
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
```

```
read_image(Image,'mreut')
dev_display (Image)
draw_rectangle1(WindowHandle,Row1,Column1,Row2,Column2)
gen_rectangle1(Rectangle,Row1,Column1,Row2,Column2)
reduce_domain(Image,Rectangle,Mask)
emphasize(Mask,Emphasize,9,9,1.0)
dev_display (Emphasize)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_rectangle1 den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'2 (H_MSG_TRUE) festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

paint_region, reduce_domain

Alternativen

gen_rectangle2, gen_region_polygon, fill_up, gen_region_runs, gen_region_points, gen_region_line

Siehe auch

draw_rectangle1, reduce_domain, smallest_rectangle1

Modul

Foundation

gen_rectangle2 (: Rectangle : Row, Column, Phi, Length1, Length2 :)

Erzeugen eines beliebig orientierten Rechtecks.

gen_rectangle2 generiert ein oder mehrere Rechtecke mit dem Schwerpunkt (Row, Column) , der Orientierung Phi und den halben Kantenlängen Length1 und Length2. Die Orientierung wird in Bogenmaß angegeben und gibt den Winkel zwischen der horizontalen Achse und der ersten Kante (mit Länge Length1) an (mathematisch positiv). Das Koordinatensystem läuft von (0,0) (linkes oberes Eck) bis (Width-1,Height-1). Siehe hierzu auch get_system und reset_obj_db. Es kann mehr als eine Region erzeugt werden, indem ein Tupel von Eckpunkten übergeben wird.

Achtung

Die Grauwerte der Ausgabeobjekte sind undefiniert. Aus Geschwindigkeitsgründen kann die Ergebnisregion zusätzliche Pixel am Rand enthalten, und einige einzelne Pixel am Rand können fehlen. Dies kann zu einer Inkonsistenz zwischen den Operatoren smallest_rectangle2 und gen_rectangle2 führen.

Parameter

▷ **Rectangle** (output_object) ..... region(-array) ⇝ *object*
Erzeugtes Rechteck.

▷ **Row** (input_control) ..... rectangle2.center.y(-array) ⇝ *real* / integer
Zeilenindex des Schwerpunktes.
Defaultwert: 300.0
Wertevorschläge: Row ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
(lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Column** (input_control) . . . . . rectangle2.center.x(-array) ⇝ *real* / integer
Spaltenindex des Schwerpunktes.
Defaultwert: 200.0
Wertevorschläge: Column ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
(lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Phi** (input_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real* / integer
Winkel der ersten Kante zur Horizontalen (Bogenmaß).
Defaultwert: 0.0
Wertevorschläge: Phi ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
Typischer Wertebereich: -1.178097 ≤ Phi ≤ 1.178097 (lin)
Restriktion: - pi / 2 < Phi && Phi <= pi / 2

▷ **Length1** (input_control) . . . . . rectangle2.hwidth(-array) ⇝ *real* / integer
Halbe Breite.
Defaultwert: 100.0
Wertevorschläge: Length1 ∈ {3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0}
(lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

▷ **Length2** (input_control) . . . . . rectangle2.hheight(-array) ⇝ *real* / integer
Halbe Höhe.
Defaultwert: 20.0
Wertevorschläge: Length2 ∈ {1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0}
(lin)
Minimale Schrittweite: 1.0
Empfohlene Schrittweite: 10.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_rectangle2 den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'2 (H_MSG_TRUE) festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

paint_region, reduce_domain

Alternativen

gen_rectangle1, gen_region_polygon_filled, gen_region_polygon, gen_region_points, fill_up

Siehe auch

draw_rectangle2, reduce_domain, smallest_rectangle2, gen_ellipse

Modul

Foundation

gen_region_contour_xld (Contour : Region : Mode :)

Generierung einer Region aus einer XLD-Kontur.

gen_region_contour_xld erzeugt eine pixelgenaue Region Region aus einer subpixelgenauen XLD-Kontur Contour. Die örtliche Diskretisierung der Kontur erfolgt nach dem Bresenham-Algorithmus und berücksichtigt die Nachbarschaftsbeziehung, welche sich mit Hilfe des Wertes des Parameters *'neighborhood'* des

Operators set_system einstellen lässt. Offene Konturen werden geschlossen, bevor sie in eine Region gewandelt werden. Der Parameter Mode legt schließlich fest, ob die Region gefüllt (*'filled'*) oder nur ihr Rand (*'margin'*) zurückgegeben wird.

Zu beachten ist, dass die Koordinaten der Konturpunkte während der Umwandlung auf die nächsten Integer-Pixelkoordinaten gerundet werden. Dies kann zu unerwarteten Ergebnissen führen, wenn man die aus dem Operator gen_contour_region_xld erhaltene Kontur als Eingabe an gen_region_contour_xld übergibt: Setzt man den Parameter Mode von gen_contour_region_xld auf *'border'*, dann sind die Eingaberegion von gen_contour_region_xld und die Ergebnisregion von gen_region_contour_xld unterschiedlich. Nehmen wir z.B. an, dass die Eingaberegion von gen_contour_region_xld lediglich aus dem Pixel (1,1) besteht. Dann enthält die Ergebniskontur von gen_contour_region_xld (wenn Mode auf *'border'* gesetzt ist) die fünf Punkte (0.5,0.5), (0.5,1.5), (1.5,1.5), (1.5,0.5) und (0.5,0.5). Übergibt man diese Kontur wiederum an gen_region_contour_xld, erhält man folglich eine Ergebnisregion, die aus den Punkten (1,1), (1,2), (2,2) und (2,1) besteht.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabe-Kontur(en).
- ▷ **Region** (output_object) ..... region(-array) ⇝ *object*
  Erzeugte Region(en).
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Füllmodus der Region(en).
  Defaultwert: 'filled'
  Wertevorschläge: Mode ∈ {'filled', 'margin'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contour_polygon_xld, gen_contour_polygon_rounded_xld

Alternativen

gen_region_polygon, gen_region_polygon_xld

Siehe auch

set_system

Modul

Foundation

gen_region_histo (: Region : Histogram, Row, Column, Scale :)

Umwandeln eines Histogramms in eine Region.

gen_region_histo wandelt ein Histogramm, das mit gray_histo, gray_histo_range oder tuple_histo_range erzeugt wurde, in eine Region um. Die Parameter Row und Column geben die Zeile bzw. Spalte des Mittelpunktes der Grafik (für die Positionierung) an. Der Skalierungsfaktor Scale legt die Größe fest. Dabei bedeutet ein Faktor von *1*, dass 256 Werte dargestellt werden; bei *2* werden 128 Werte dargestellt, bei *3* werden 64 Werte dargestellt, usw.

Parameter

- ▷ **Region** (output_object) ..... region ⇝ *object*
  Darstellung des Histogramms als Region.
- ▷ **Histogram** (input_control) ..... histogram-array ⇝ *integer*
  Eingabehistogramm.

▷ **Row** (input_control) . . . . . point.y ⇝ *integer*
Zeilenkoordinate des Mittelpunktes.
Defaultwert: 255
Wertevorschläge: Row ∈ {100, 200, 255, 300, 400}
Typischer Wertebereich: 0 ≤ Row ≤ 511

▷ **Column** (input_control) . . . . . point.x ⇝ *integer*
Spaltenkoordinate des Mittelpunktes.
Defaultwert: 255
Wertevorschläge: Column ∈ {100, 200, 255, 300, 400}
Typischer Wertebereich: 0 ≤ Column ≤ 511

▷ **Scale** (input_control) . . . . . integer ⇝ *integer*
Verkleinerungsfaktor.
Defaultwert: 1
Wertevorschläge: Scale ∈ {1, 2, 3, 4, 5, 6, 7}
Typischer Wertebereich: 1 ≤ Scale ≤ 10 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Ergebnis

gen_region_histo liefert den Wert TRUE, falls die Parameter korrekt sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gray_histo, gray_histo_range, tuple_histo_range

Siehe auch

disp_channel

Modul

Foundation

gen_region_hline (: Regions : Orientation, Distance :)

Abspeichern von in Hessescher Normalform beschriebenen Eingabelinien als Regionen.

gen_region_hline legt die in Hessescher Normalform beschriebenen Linien als Regionen ab. Eine Linie wird bestimmt durch den Abstand der Linie vom Ursprung (Distance, entspricht der Länge des Normalenvektors) und der Richtung des Normalenvektors (Orientation, entspricht der Orientierung der Linie $\pm\pi/2$). Die Richtungen wurden so definiert, dass bei Orientation = 0 der Normalenvektor in Richtung der X-Achse liegt, was einer senkrechten Linie entspricht. Bei Orientation = $\pi/2$ zeigt der Normalenvektor in Richtung der Y-Achse, d.h. es wird eine waagerechte Linie beschrieben.

Achtung

Die Linien werden am derzeitigen maximalem Bildformat geclippt.

Parameter

▷ **Regions** (output_object) . . . . . region(-array) ⇝ *object*
Erzeugte Regionen (für jede Linie eine), beschnitten auf maximales Bildformat.
Parameteranzahl: Regions == Distance

▷ **Orientation** (input_control) . . . . . hesseline.angle.rad(-array) ⇝ *real* / integer
Orientierung des Normalenvektors in Bogenmaß.
Parameteranzahl: Orientation == Distance
Defaultwert: 0.0
Wertevorschläge: `Orientation` ∈ {-0.78, 0.0, 0.78, 1.57}
(lin)
Empfohlene Schrittweite: 0.02

▷ **Distance** (input_control) . . . . . hesseline.distance(-array) ⇝ *real* / integer
Abstand der Linie vom Koordinatenursprung (0,0).
Defaultwert: 200
Wertevorschläge: `Distance` ∈ {10, 50, 100, 200, 300, 400}
(lin)
Empfohlene Schrittweite: 1

Ergebnis

`gen_region_hline` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`gen_region_line`

Siehe auch

`hough_lines`

Modul

Foundation

gen_region_line (: RegionLines : BeginRow, BeginCol, EndRow, EndCol :)

Abspeichern von Eingabelinien als Regionen.

`gen_region_line` legt die übergebenen Linien (mit Anfangspunkt [`BeginRow`,`BeginCol`] und Endpunkt [`EndRow`, `EndCol`]) als Region ab.

Parameter

▷ **RegionLines** (output_object) . . . . . region(-array) ⇝ *object*
Erzeugte Regionen.

▷ **BeginRow** (input_control) . . . . . line.begin.y(-array) ⇝ *integer* / real
Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
Defaultwert: 100
Wertevorschläge: `BeginRow` ∈ {10, 50, 100, 200, 300, 400}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **BeginCol** (input_control) . . . . . line.begin.x(-array) ⇝ *integer* / real
Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
Defaultwert: 50
Wertevorschläge: `BeginCol` ∈ {10, 50, 100, 200, 300, 400}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **EndRow** (input_control) . . . . . line.end.y(-array) ⇝ *integer* / real
Zeilenkoordinaten der Endpunkte der Eingabelinien.
Defaultwert: 150
Wertevorschläge: `EndRow` ∈ {50, 100, 200, 300, 400, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **EndCol** (input_control) . . . . . line.end.x(-array) ⇝ *integer* / real
Spaltenkoordinaten der Endpunkte der Eingabelinien.
Defaultwert: 250
Wertevorschläge: `EndCol` ∈ {50, 100, 200, 300, 400, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Ergebnis

`gen_region_line` liefert immer den Wert TRUE. Das Clipping am aktuellen Bildformat wird durch `set_system('clip_region',<'2 (H_MSG_TRUE)` festgelegt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`split_skeleton_lines`

Alternativen

`gen_region_hline`

Modul

Foundation

gen_region_points (: Region : Rows, Columns :)

Abspeichern einzelner Punkte als Bildregion.

`gen_region_points` erzeugt eine Region, die durch eine Anzahl von Punkten beschrieben wird. Die Punkte müssen in keiner festen Reihenfolge abgespeichert werden, doch ergibt sich das beste Laufzeitverhalten, wenn die Punkte aufsteigend sortiert sind. Die Ordnung lautet:

$$(l_1, c_1) \leq (l_2, c_2) := (l_1 < l_2) \quad \vee \quad (l_1 = l_2) \wedge (c_1 \leq c_2)$$

Die angegebenen Koordinaten stehen für zwei aufeinanderfolgende Punkte in dem Tupel.

Parameter

▷ **Region** (output_object) . . . . . region ⇝ *object*
Erzeugte Region.

▷ **Rows** (input_control) . . . . . coordinates.y(-array) ⇝ *integer* / real
Zeilen der Punkte in der Region.
Defaultwert: 100
Wertevorschläge: `Rows` ∈ {0, 10, 30, 50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Columns** (input_control) . . . . . coordinates.x(-array) ⇝ *integer / real*
Spalten der Punkte in der Region.
Parameteranzahl: Columns == Rows
Defaultwert: 100
Wertevorschläge: `Columns` ∈ {0, 10, 30, 50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei F die Anzahl der Punkte. Falls die Punkte aufsteigend sortiert sind, dann ist die Laufzeitkomplexität: $O(F)$, sonst $O(\log(F) * F)$.

Ergebnis

`gen_region_points` liefert den Wert TRUE, falls die Punkte innerhalb des Bildformats liegen. Ansonsten wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch `set_system ('clip_region',<'2 (H_MSG_TRUE)` festgelegt. Wird eine leere Region erzeugt (durch das Clipping oder eine leere Eingabe), dann legt `set_system('store_empty_region',<'2 (H_MSG_TRUE)` fest, ob die Region ausgegeben wird.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`get_region_points`

Nachfolger

`paint_region`, `reduce_domain`

Alternativen

`gen_region_polygon`, `gen_region_runs`, `gen_region_line`

Siehe auch

`reduce_domain`

Modul

Foundation

gen_region_polygon (: Region : Rows, Columns :)

Abspeichern eines Polygons als Region.

`gen_region_polygon` erzeugt eine Region aus einem Polygonzug, der durch eine Folge von Zeilen- und Spaltenkoordinaten beschrieben wird. Die erzeugte Region besteht aus den Punkten der dadurch definierten Strecken, wobei zwischen den Stützpunkten linear interpoliert wird.

Achtung

Die Region wird nicht automatisch geschlossen und nicht „aufgefüllt“.

Parameter

▷ **Region** (output_object) . . . . . region ⇝ *object*
Erzeugte Region.

▷ **Rows** (input_control) . . . . . polygon.y-array ⇝ *integer*
Zeilenindizes der Stützpunkte der Regionenkontur.
Defaultwert: 100
Wertevorschläge: `Rows` ∈ {0, 10, 30, 50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Columns** (input_control) . . . . . polygon.x-array ⇝ *integer*
Spaltenindizes der Stützpunkte der Regionenkontur.
Parameteranzahl: Columns == Rows
Defaultwert: 100
Wertevorschläge: Columns ∈ {0, 10, 30, 50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Beispiel

```
* Polygon-approximation
get_region_polygon(Region,7,Row,Column)
* store it as a region
gen_region_polygon(Pol,Row,Column)
* fill up the hole
fill_up(Pol,Filled)
```

Ergebnis

Falls die Stützpunkte korrekt sind, liefert gen_region_polygon den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'2 (H_MSG_TRUE) festgelegt. Wird eine leere Region erzeugt (durch das Clipping oder eine leere Eingabe), dann legt set_system('store_empty_region',<'2 (H_MSG_TRUE) fest, ob die Region ausgegeben wird.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

get_region_polygon, draw_polygon

Alternativen

gen_region_polygon_filled, gen_region_points, gen_region_runs

Siehe auch

fill_up, reduce_domain, get_region_polygon, draw_polygon

Modul

Foundation

gen_region_polygon_filled (: Region : Rows, Columns :)

Abspeichern eines Polygons als eine „aufgefüllte“ Region.

gen_region_polygon_filled erzeugt eine Region aus einem Polygon, das die Eckpunkte der Region (Zeilen- und Spaltenkoordinaten) entweder im oder gegen den Uhrzeigersinn enthält. Im Gegensatz zu gen_region_polygon wird hier eine „aufgefüllte“ Region zurückgeliefert.

Parameter

▷ **Region** (output_object) . . . . . region ⇝ *object*
Erzeugte Region.

▷ **Rows** (input_control) . . . . . polygon.y-array ⇝ *integer* / real
Zeilenindizes der Stützpunkte der Regionenkontur.
Defaultwert: 100
Wertevorschläge: Rows ∈ {0, 10, 30, 50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Columns** (input_control) . . . . . polygon.x-array ⇝ *integer* / real
Spaltenindizes der Stützpunkte der Regionenkontur.
Parameteranzahl: Columns == Rows
Defaultwert: 100
Wertevorschläge: `Columns` ∈ {0, 10, 30, 50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Beispiel

```
/* Polygon approximation */
T_get_region_polygon(Region,7,&Row,&Column);
T_gen_region_polygon_filled(&Pol,Row,Column);
/* fill up with original gray value */
reduce_domain(Image,Pol,&New);
```

Ergebnis

Falls die Stützpunkte korrekt sind, liefert `gen_region_polygon_filled` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch `set_system ('clip_region',<'2 (H_MSG_TRUE)` festgelegt. Wird eine leere Region erzeugt (durch das Clipping oder eine leere Eingabe), dann legt `set_system('store_empty_region',<'2 (H_MSG_TRUE)` fest, ob die Region ausgegeben wird.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_region_polygon`, `draw_polygon`

Alternativen

`gen_region_polygon`, `gen_region_points`, `draw_polygon`

Siehe auch

`gen_region_polygon`, `reduce_domain`, `get_region_polygon`, `gen_region_runs`

Modul

Foundation

gen_region_polygon_xld (Polygon : Region : Mode :)

Generierung einer Region aus einem XLD-Polygon.

`gen_region_polygon_xld` erzeugt eine pixelgenaue Region `Region` aus einem subpixelgenauen XLD-Polygon `Polygon`. Die örtliche Diskretisierung des Polygons erfolgt nach dem Bresenham-Algorithmus und berücksichtigt die Nachbarschaftsbeziehung, welche sich mit Hilfe des Parameters *'neighborhood'* des Operators `set_system` einstellen lässt. Offene Konturen werden geschlossen, bevor sie in eine Region gewandelt werden. Der Parameter `Mode` legt schließlich fest, ob die Region gefüllt (*'filled'*) oder nur ihr Rand (*'margin'*) zurückgegeben wird.

Parameter

▷ **Polygon** (input_object) . . . . . xld_poly(-array) ⇝ *object*
Eingabe-Polygon(e).

▷ **Region** (output_object) . . . . . region(-array) ⇝ *object*
Erzeugte Region(en).

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Füllmodus der Region(en).
Defaultwert: 'filled'
Wertevorschläge: Mode ∈ {'filled', 'margin'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_polygons_xld

Alternativen

gen_region_polygon, gen_region_contour_xld

Siehe auch

set_system

Modul

Foundation

gen_region_runs (: Region : Row, ColumnBegin, ColumnEnd :)

Erzeugen einer Region aus einer Lauflängenkodierung.

gen_region_runs erzeugt eine Region, die über die eingegebene Sehnenstruktur beschrieben ist. Die Sehnendarstellung entsteht, indem man eine Region zeilenweise mit aufsteigender Zeilenzahl (= von „oben“ nach „unten“) untersucht. Jede Zeile wird von links nach rechts durchlaufen (aufsteigende Spaltenzahl). Dabei werden alle Anfangs- und Endpunkte von Regionenabschnitten (=Sehnen) gespeichert. Eine Region lässt sich somit durch eine Folge von Sehnen beschreiben, wobei eine Sehne durch Zeilennummer, sowie Anfangs- und Endpunkte (Spaltennummer) definiert ist.

Die Abspeicherung erfolgt am schnellsten, wenn die Sehnen sortiert sind. Die Ordnung lautet:

$$(l_1, b_1, e_1) \leq (l_2, b_2, e_2) := (l_1 < l_2) \quad \vee \quad (l_1 = l_2) \wedge (b_1 \leq b_2)$$

.

Achtung

Aus Laufzeitgründen wird nicht überprüft, ob die Restriktion ColumnEnd >= ColumnBegin eingehalten wird. Wird diese Einschränkung verletzt, so kann dies bei nachfolgenden Operatoraufrufen zu Fehlern und unerwartetem Verhalten führen.

Parameter

▷ **Region** (output_object) . . . . . region ⇝ *object*
Erzeugte Region.

▷ **Row** (input_control) . . . . . chord.y(-array) ⇝ *integer*
Zeilen der Sehnen.
Defaultwert: 100
Wertevorschläge: Row ∈ {0, 50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **ColumnBegin** (input_control) . . . . . chord.x1(-array) ⇝ *integer*
Spalten der Anfangspunkte der Sehnen.
Parameteranzahl: ColumnBegin == Row
Defaultwert: 50
Wertevorschläge: ColumnBegin ∈ {0, 50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **ColumnEnd** (input_control) . . . . . chord.x2(-array) ⇝ *integer*
Spalten der Endpunkte der Sehnen.
Parameteranzahl: ColumnEnd == Row
Defaultwert: 200
Wertevorschläge: ColumnEnd ∈ {50, 100, 200, 300, 500}
(lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: ColumnEnd >= ColumnBegin

Komplexität

Sei F die Anzahl der Punkte. Falls die Punkte aufsteigend sortiert sind, dann ist die Laufzeitkomplexität: $O(F)$, sonst $O(log(F) * F)$.

Ergebnis

gen_region_runs liefert den Wert TRUE, falls die Daten korrekt sind, ansonsten wird eine Fehlerbehandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'2 (H_MSG_TRUE) festgelegt. Wird eine leere Region erzeugt (durch das Clipping oder eine leere Eingabe), dann legt set_system('store_empty_region',<'2 (H_MSG_TRUE) fest, ob die Region ausgegeben wird.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

get_region_runs

Alternativen

gen_region_points, gen_region_polygon, gen_region_line, gen_region_polygon_filled

Siehe auch

reduce_domain

Modul

Foundation

label_to_region (LabelImage : Regions : :)

Suchen von Regionen mit gleichen Grauwerten.

label_to_region segmentiert Bilder in Regionen gleichen Grauwerts. Dabei wird für jeden Grauwert im Bild genau eine Ausgaberegion erzeugt. Dies entspricht einem mehrfachen Aufruf des Operators threshold, gefolgt von der Konkatenation der entstandenen Regionen (concat_obj). Verwandt ist die Routine auch mit regiongrowing. label_to_region führt allerdings kein connection durch, d.h. die Ausgaberegionen müssen nicht zusammenhängen. Eine typische Anwendung von label_to_region ist die Segmentierung „gelabelter“ Bilder, daher auch der Name.

Die Anzahl der Ausgaberegionen wird durch den Systemparameter *'max_outp_obj_par'* begrenzt, der mittels

get_system(::'max_outp_obj_par':<Anzahl>)

abgefragt werden kann.

Achtung

label_to_region ist nicht für Real-Bilder implementiert. Die Eingabebilder dürfen keine negativen Grauwerte enthalten.

Parameter

▷ **LabelImage** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / int4 / int8
„Gelabeltes“ Bild.

▷ **Regions** (output_object) . . . . . region-array ⇝ *object*
Regionen mit konstantem Grauwert.

Komplexität

Sei $x1$ die minimale X-Koordinate, $x2$ die maximale X-Koordinate, $y1$ die minimale Y-Koordinate und $y2$ die maximale Y-Koordinate eines auftretenden Grauwertes und N die Anzahl der verschiedenen Grauwerte, dann ist die Laufzeitkomplexität $O(N * (x2 - x1 + 1) * (y2 - y1 + 1))$

Ergebnis

Sind die Grauwerte in einem zulässigen Bereich, dann liefert `label_to_region` den Wert TRUE. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`min_max_gray`, `sobel_amp`, `binomial_filter`, `gauss_filter`, `reduce_domain`, `diff_of_gauss`

Nachfolger

`connection`, `dilation1`, `erosion1`, `opening`, `closing`, `rank_region`, `shape_trans`, `skeleton`

Siehe auch

`threshold`, `concat_obj`, `regiongrowing`, `region_to_label`

Modul

Foundation

22.2 Geometrische Transformationen

```
affine_trans_region ( Region : RegionAffineTrans : HomMat2D,
    Interpolate : )
```

Anwenden einer beliebigen affinen 2D-Transformation auf Regionen.

`affine_trans_region` wendet eine beliebige affine 2D-Transformation (Skalierung, Drehung, Verschiebung, Scherung) auf die in `Region` übergebenen Regionen an und liefert die transformierten Regionen in `RegionAffineTrans` zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in `HomMat2D` übergeben wird. Diese kann mit Hilfe der Operatoren `hom_mat2d_identity`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_translate` etc. aufgebaut werden oder das Resultat von Operatoren wie `vector_angle_to_rigid` sein.

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, dass die *Zeilen*-Koordinate des Bildes der x-Koordinate und die *Spalten*-Koordinate des Bildes der y-Koordinate des Koordinatensystems entspricht, in dem die Transformationsmatrix definiert wurde. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Der Parameter `Interpolate` gibt dabei an, ob die Transformation mit einer internen Interpolation ausgeführt werden soll oder nicht. Es sind die in `affine_trans_image` beschriebenen Optionen *'nearest_neighbor'* und *'constant'* verfügbar. Eine Interpolation kann insbesondere bei Vergrößerungen zu glatteren Regionenrändern führen. Allerdings steigt dadurch die Laufzeit erheblich.

Achtung

`affine_trans_region` ist i.a. nicht reversibel (Clipping und Rasterung bei Rotation und Streckung).

`affine_trans_region` verwendet nicht das HALCON-Standard-Koordinatensystem (mit dem Ursprung im Mittelpunkt des linken oberen Pixels) sondern das gleiche Koordinatensystem wie `affine_trans_pixel`, d.h. der Ursprung liegt in der linken oberen Ecke des linken oberen Pixels. Eine Transformation mit `affine_trans_region` entspricht für jeden Punkt der Region einer Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt, siehe `affine_trans_pixel`). Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von der Region mit Operatoren wie z.B. `area_center` abgeleitet wurden. Wenn z.B. eine rotationssymmetrische Region unter Verwendung von `hom_mat2d_rotate` um ihren mit `area_center` bestimmten Schwerpunkt rotiert wird, so liegt die transformierte Region nicht auf der Originalregion. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von `affine_trans_region` die Transformationsmatrix `HomMat2D` durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
affine_trans_region(Region, RegionAffineTrans, HomMat2DAdapted,
'nearest_neighbor')
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Zu transformierende Region(en).
- ▷ **`RegionAffineTrans`** (output_object) ..... region(-array) ⇝ *object*
  Region(en), die transformiert wurde(n).
  Parameteranzahl: RegionAffineTrans == Region
- ▷ **`HomMat2D`** (input_control) ..... hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **`Interpolate`** (input_control) ..... string ⇝ *string*
  Soll die Transformation mit Interpolation ausgeführt werden?
  Defaultwert: 'nearest_neighbor'
  Werteliste: `Interpolate` ∈ {'constant', 'nearest_neighbor'}

Ergebnis

Falls die Matrix `HomMat2D` eine affine Abbildung darstellt (d.h. keine projektive Abbildung), liefert `affine_trans_region` den Wert TRUE zurück. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<'2 (H_MSG_TRUE)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`hom_mat2d_identity`, `hom_mat2d_scale`, `hom_mat2d_translate`, `hom_mat2d_invert`, `hom_mat2d_rotate`, `hom_mat2d_reflect`

Nachfolger

`select_shape`

Alternativen

`move_region`, `mirror_region`, `zoom_region`

Siehe auch

`affine_trans_image`

Modul

Foundation

mirror_region (Region : RegionMirror : Mode, WidthHeight :)

Spiegeln von Regionen an einer Achse.

mirror_region spiegelt die Eingaberegion(en) an einer von drei Achsen. Wenn Mode *'row'* ist, wird die Eingaberegion vertikal gespiegelt, wenn er *'column'* ist, wird sie horizontal gespiegelt, und wenn er *'diagonal'* ist, wird die Eingaberegion an der Winkelhalbierenden $x = y$ gespiegelt.

Für Mode = *'row'* oder *'column'* entspricht WidthHeight dabei dem Index der Spiegelachse * 2. Für den Fall, dass Region aus einem Bild extrahiert wurde und so gespiegelt werden soll, als ob es aus der Spiegelung des Bildes extrahiert worden wäre, entspricht WidthHeight einer der Dimensionen dieses Bildes (abhängig von Mode). Der Parameter WidthHeight wird bei Mode = *'diagonal'* nicht genutzt.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Zu transformierende Region(en).
- ▷ **RegionMirror** (output_object) ..... region(-array) ⇝ *object*
  Gespiegelte Region(en).
  Parameteranzahl: RegionMirror == Region
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Spiegelachse.
  Defaultwert: 'row'
  Werteliste: Mode ∈ {'column', 'row', 'diagonal'}
- ▷ **WidthHeight** (input_control) ..... integer ⇝ *integer*
  Doppelte Koordinate der Spiegelachse.
  Defaultwert: 512
  Wertevorschläge: WidthHeight ∈ {128, 256, 512, 525, 768, 1024}
  Typischer Wertebereich: $1 \leq$ WidthHeight ≤ 1024 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
  Restriktion: WidthHeight > 0

Beispiel

```
read_image(&Image,"monkey");
threshold(Image,&Seg,128.0,255.0);
mirror_region(Seg,&Mirror,"row",512);
disp_region(Mirror,WindowHandle);
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection, regiongrowing, pouring

Nachfolger

select_shape, disp_region

Alternativen

hom_mat2d_reflect, affine_trans_region

Siehe auch

zoom_region

Modul

Foundation

move_region (Region : RegionMoved : Row, Column :)

Verschieben einer Region.

move_region verschiebt alle Eingaberegionen um den Vektor (Row, Column). Dabei erfolgt gegebenenfalls ein Clipping der Ergebnisregionen an dem aktuellen Bildformat.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Region(en), die verschoben werden soll(en).
- ▷ **RegionMoved** (output_object) ..... region(-array) ⇝ *object*
  Modifizierte Region(en).
  Parameteranzahl: RegionMoved == Region
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Um diesen Wert werden alle Regionen vertikal verschoben.
  Defaultwert: 30
  Wertevorschläge: Row ∈ {-128, -64, -32, -16, -10, -8, -4, -2, -1, 0, 1, 2, 4, 5, 8, 10, 16, 32, 64, 128}
  Typischer Wertebereich: -512 ≤ Row ≤ 512 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Um diesen Wert werden alle Regionen horizontal verschoben.
  Defaultwert: 30
  Wertevorschläge: Column ∈ {-128, -64, -32, -16, -10, -8, -4, -2, -1, 0, 1, 2, 4, 5, 8, 10, 16, 32, 64, 128}
  Typischer Wertebereich: -512 ≤ Column ≤ 512 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10

Komplexität

Sei F die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität: $O(sqrt(F))$.

Ergebnis

move_region liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), das bei leerer Ergebnisregion mit set_system ('store_empty_region',<'2 (H_MSG_TRUE) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection, regiongrowing, pouring

Nachfolger

select_shape, disp_region

Siehe auch

affine_trans_image, mirror_region, zoom_region

Modul

Foundation

polar_trans_region (Region : PolarTransRegion : Row, Column, AngleStart, AngleEnd, RadiusStart, RadiusEnd, Width, Height, Interpolation :)

Anwenden einer Polarkoordinatentransformation auf eine Region innerhalb eines Kreisrings.

polar_trans_region transformiert eine Region, die innerhalb des durch den Mittelpunkt (Row, Column), die Radien RadiusStart und RadiusEnd und die Winkel AngleStart und AngleEnd definierten Kreisrings liegt, auf dessen Polarkoordinatendarstellung in der Region PolarTransRegion in einem virtuellen Bild mit den Dimensionen Width × Height.

Die Polartransformation entspricht einem Wechsel des Koordinatensystems. Anstatt durch Zeilen- und Spaltenkoordinate wird die Position eines Punktes durch dessen Radius (Abstand zum Mittelpunkt Row,Column) und den Winkel zwischen der Spaltenachse (durch den Mittelpunkt) und der Strecke vom Mittelpunkt zum Punkt beschrieben. Diese Transformation ist nicht affin.

Die Koordinate $(0,0)$ in der Ausgaberegion entspricht immer dem Punkt in der Eingaberegion, der durch RadiusStart und AngleStart spezifiziert wird. Analog entspricht die Koordinate $(\texttt{Height}-1,\texttt{Width}-1)$ immer dem Punkt in der Eingaberegion, der durch RadiusEnd und AngleEnd spezifiziert wird. Im Normalfall ($\texttt{AngleStart} < \texttt{AngleEnd}$ und $\texttt{RadiusStart} < \texttt{RadiusEnd}$) wird die Polarkoordinatentransformation also in mathematisch positiver Durchlaufrichtung (entgegen dem Uhrzeigersinn) durchgeführt. Dabei liegen die Punkte mit kleineren Radien in der Ausgaberegion weiter oben. Durch geeignetes Vertauschen der Werte der Parameter (z.B. $\texttt{AngleStart} > \texttt{AngleEnd}$ oder $\texttt{RadiusStart} > \texttt{RadiusEnd}$) kann jede beliebige Orientierung der Ausgaberegion erreicht werden.

Die Winkel sind frei als reelle Zahlen wählbar, Mittelpunkt und Radien können ebenfalls frei als reelle Zahlen gewählt werden. Sind diese jedoch ganze Zahlen und entspricht die Differenz von RadiusEnd und RadiusStart dem Wert $\texttt{Height} - 1$, so wird die Berechnung durch eine optimierte Routine erheblich schneller durchgeführt.

Die Radien und Winkel sind inklusive, d.h. die erste Zeile des virtuellen Zielbildes entspricht dem Kreis mit dem Radius RadiusStart, die letzte Zeile dem mit dem Radius RadiusEnd. Für Vollkreise, bei denen die Differenz AngleStart - AngleEnd 2π (360 Grad) beträgt, bedeutet dies insbesondere, dass die erste Spalte des virtuellen Zielbildes gleich der letzten ist.

Um diesem Verhalten vorzubeugen, sollte diese Differenz $2\pi(1 - \frac{1}{\texttt{Width}})$ statt 2π betragen.

Über den Parameter Interpolation kann dabei zwischen den Interpolationsverfahren *'bilinear'* und *'nearest_neighbor'* gewählt werden. Die Wahl $\texttt{Interpolation} = $ *'bilinear'* kann insbesondere bei Vergrößerungen zu glatteren Regionenrändern führen. Allerdings steigt dadurch die Laufzeit signifikant an.

Werden mehrere Regionen in Region übergeben, werden deren Polartransformationen individuell berechnet und als Tupel in PolarTransRegion abgespeichert. Dabei entsprechen die Indizes von Ein- und Ausgaberegionen nur dann einander, wenn die Systemvariable *'store_empty_regions'* auf *'true'* gesetzt wird (siehe auch set_system). Sonst werden leere Regionen nicht zurückgegeben und deshalb entspricht die Länge des Ausgabetupels PolarTransRegion im Allgemeinen nicht der Länge des Eingabetupels Region.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Achtung

Werden die Dimensionen des virtuellen Ausgabebildes Width und Height größer als die Dimensionen des aktuellen Bildes gewählt, sollte die Systemvariable *'clip_region'* auf *'false'* gesetzt werden (siehe auch set_system). Andernfalls kann eine Ausgaberegion, die nicht innerhalb der Dimensionen des aktuellen Bildes liegt, zu einer Fehlermeldung führen.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegion.
- ▷ **PolarTransRegion** (output_object) ..... region(-array) ⇝ *object*
  Ergebnisregion.
- ▷ **Row** (input_control) ..... number ⇝ *real* / integer
  Zeilenkoordinate des Kreismittelpunktes.
  Defaultwert: 256
  Wertevorschläge: Row ∈ {0, 16, 32, 64, 128, 240, 256, 480, 512}
  Typischer Wertebereich: $0 \leq \texttt{Row} \leq 32767$
  Restriktion: Row >= -131068 && Row <= 131068

▷ **Column** (input_control) . . . . . number ⇝ *real* / integer
Spaltenkoordinate des Kreismittelpunktes.
Defaultwert: 256
Wertevorschläge: Column ∈ {0, 16, 32, 64, 128, 256, 320, 512, 640}
Typischer Wertebereich: 0 ≤ Column ≤ 32767
Restriktion: Column >= -131068 && Column <= 131068

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Winkel, der auf die Spaltenkoordinate *0* in PolarTransRegion abgebildet wird.
Defaultwert: 0.0
Wertevorschläge: AngleStart ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853, 12.566370616}
Typischer Wertebereich: -6.2831853 ≤ AngleStart ≤ 6.2831853

▷ **AngleEnd** (input_control) . . . . . angle.rad ⇝ *real*
Winkel, der auf die Spaltenkoordinate Width − 1 in PolarTransRegion abgebildet wird.
Defaultwert: 6.2831853
Wertevorschläge: AngleEnd ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853, 12.566370616}
Typischer Wertebereich: -6.2831853 ≤ AngleEnd ≤ 6.2831853

▷ **RadiusStart** (input_control) . . . . . number ⇝ *real* / integer
Kreisradius, der auf die Zeilenkoordinate *0* in PolarTransRegion abgebildet wird.
Defaultwert: 0
Wertevorschläge: RadiusStart ∈ {0, 16, 32, 64, 100, 128, 256, 512}
Typischer Wertebereich: 0 ≤ RadiusStart ≤ 32767

▷ **RadiusEnd** (input_control) . . . . . number ⇝ *real* / integer
Kreisradius, der auf die Zeilenkoordinate Height − 1 abgebildet wird.
Defaultwert: 100
Wertevorschläge: RadiusEnd ∈ {0, 16, 32, 64, 100, 128, 256, 512}
Typischer Wertebereich: 0 ≤ RadiusEnd ≤ 32767

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des virtuellen Zielbildes.
Defaultwert: 512
Wertevorschläge: Width ∈ {256, 320, 512, 640, 800, 1024}
Typischer Wertebereich: 2 ≤ Width ≤ 32767

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des virtuellen Zielbildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {240, 256, 480, 512, 600, 1024}
Typischer Wertebereich: 2 ≤ Height ≤ 32767

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Interpolationsverfahren für die Transformation.
Defaultwert: 'nearest_neighbor'
Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Siehe auch

polar_trans_image_ext, polar_trans_image_inv, polar_trans_region_inv, polar_trans_contour_xld, polar_trans_contour_xld_inv

Modul

Foundation

polar_trans_region_inv (PolarRegion : XYTransRegion : Row, Column, AngleStart, AngleEnd, RadiusStart, RadiusEnd, WidthIn, HeightIn, Width, Height, Interpolation :)

Rücktransformieren einer Region in Polarkoordinaten.

polar_trans_region_inv transformiert die Polarkoordinatendarstellung einer Region, gespeichert in PolarRegion, zurück auf einen Kreisring in kartesischen Koordinaten, der durch die Radien RadiusStart und RadiusEnd sowie durch die Winkel AngleStart und AngleEnd beschrieben ist und dessen Mittelpunkt in (Row, Column) liegt. Alle diese Werte können frei aus den reellen Zahlen gewählt werden. Zusätzlich müssen die Dimensionen des virtuellen Eingabebildes, welches die Region PolarRegion enthält, in WidthIn und HeightIn übergeben werden. WidthIn $-$ 1 ist die Spaltenkoordinate, die AngleEnd entspricht, und HeightIn $-$ 1 ist die Zeilenkoordinate, die RadiusEnd entspricht. AngleStart bzw. RadiusStart entspricht der Spalten- bzw. Zeilenkoordinate *0*. Außerdem müssen die Dimensionen des virtuellen Ausgabebildes, welches die Region XYTransRegion enthält, in Width und Height übergeben werden.

Die Winkel und Radien sind inklusive, die Zeile mit der Koordinate *0* der Eingaberegion wird also auf einen Kreis im Abstand RadiusStart vom angegebenen Mittelpunkt abgebildet, die Zeile mit der Koordinate HeightIn $-$ 1 auf einen Kreis im Abstand RadiusEnd. Das Gleiche gilt für AngleStart, AngleEnd und WidthIn. Entspricht die Breite der Eingaberegion einem Winkelintervall, dessen Länge größer als $2pi$ ist, wird die Eingaberegion so beschnitten, dass die Länge dieses Intervalls $2pi$ ist.

Über den Parameter Interpolation kann dabei zwischen den Interpolationsverfahren *'bilinear'* und *'nearest_neighbor'* gewählt werden. Die Wahl Interpolation $=$ *'bilinear'* kann insbesondere bei Vergrößerungen zu glatteren Regionenrändern führen. Allerdings steigt dadurch die Laufzeit signifikant an.

polar_trans_region_inv ist die exakte Umkehrfunktion zu polar_trans_region.

Die Aufrufsequenz:

```
polar_trans_region(Region, PolarRegion, Row, Column, rad(360), 0, 0,
Radius, Width, Height, 'nearest_neighbor')
polar_trans_region_inv(PolarRegion, XYTransRegion, Row, Column, rad(360),
0, 0, Radius, Width, Height, Width, Height, 'nearest_neighbor')
```

liefert in der Ausgaberegion XYTransRegion die Region Region eingeschränkt auf den Kreis um (Row,Column) mit Radius Radius zurück.

Werden mehrere Regionen in PolarRegion übergeben, werden deren Transformationen individuell berechnet und als Tupel in XYTransRegion abgespeichert. Dabei entsprechen die Indizes von Ein- und Ausgaberegionen nur dann einander, wenn die Systemvariable *'store_empty_regions'* auf *'true'* gesetzt wird (siehe auch set_system). Sonst werden leere Regionen nicht zurückgegeben und deshalb entspricht die Länge des Ausgabetupels XYTransRegion im Allgemeinen nicht der Länge des Eingabetupels PolarRegion.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Achtung

Werden die Dimensionen des virtuellen Ausgabebildes Width und Height größer als die Dimensionen des aktuellen Bildes gewählt, sollte die Systemvariable *'clip_region'* auf *'false'* gesetzt werden (siehe auch set_system). Andernfalls kann eine Ausgaberegion, die nicht innerhalb der Dimensionen des aktuellen Bildes liegt, zu einer Fehlermeldung führen.

Parameter

- ▷ **PolarRegion** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegion.
- ▷ **XYTransRegion** (output_object) ..... region(-array) ⇝ *object*
  Ergebnisregion.
- ▷ **Row** (input_control) ..... number ⇝ *real* / integer
  Zeilenkoordinate des Kreismittelpunktes.
  Defaultwert: 256
  Wertevorschläge: Row ∈ {0, 16, 32, 64, 128, 240, 256, 480, 512}
  Typischer Wertebereich: 0 ≤ Row ≤ 32767
  Restriktion: Row >= -131068 && Row <= 131068

▷ **Column** (input_control) ..... number ⇝ *real* / integer
Spaltenkoordinate des Kreismittelpunktes.
Defaultwert: 256
Wertevorschläge: Column ∈ {0, 16, 32, 64, 128, 256, 320, 512, 640}
Typischer Wertebereich: 0 ≤ Column ≤ 32767
Restriktion: Column >= -131068 && Column <= 131068

▷ **AngleStart** (input_control) ..... angle.rad ⇝ *real*
Winkel, auf den die Spaltenkoordinate *0* von PolarRegion abgebildet wird.
Defaultwert: 0.0
Wertevorschläge: AngleStart ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853}
Typischer Wertebereich: -6.2831853 ≤ AngleStart ≤ 6.2831853

▷ **AngleEnd** (input_control) ..... angle.rad ⇝ *real*
Winkel, auf den die Spaltenkoordinate WidthIn − 1 in PolarRegion abgebildet wird.
Defaultwert: 6.2831853
Wertevorschläge: AngleEnd ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853}
Typischer Wertebereich: -6.2831853 ≤ AngleEnd ≤ 6.2831853

▷ **RadiusStart** (input_control) ..... number ⇝ *real* / integer
Kreisradius, auf den die Zeilenkoordinate *0* in PolarRegion abgebildet wird.
Defaultwert: 0
Wertevorschläge: RadiusStart ∈ {0, 16, 32, 64, 100, 128, 256, 512}
Typischer Wertebereich: 0 ≤ RadiusStart ≤ 32767

▷ **RadiusEnd** (input_control) ..... number ⇝ *real* / integer
Kreisradius, auf den die Zeilenkoordinate HeightIn − 1 in PolarRegion abgebildet wird.
Defaultwert: 100
Wertevorschläge: RadiusEnd ∈ {0, 16, 32, 64, 100, 128, 256, 512}
Typischer Wertebereich: 0 ≤ RadiusEnd ≤ 32767

▷ **WidthIn** (input_control) ..... extent.x ⇝ *integer*
Breite des virtuellen Eingabebildes.
Defaultwert: 512
Wertevorschläge: WidthIn ∈ {256, 320, 512, 640, 800, 1024}
Typischer Wertebereich: 2 ≤ WidthIn ≤ 32767

▷ **HeightIn** (input_control) ..... extent.y ⇝ *integer*
Höhe des virtuellen Eingabebildes.
Defaultwert: 512
Wertevorschläge: HeightIn ∈ {240, 256, 480, 512, 600, 1024}
Typischer Wertebereich: 2 ≤ HeightIn ≤ 32767

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des virtuellen Zielbildes.
Defaultwert: 512
Wertevorschläge: Width ∈ {256, 320, 512, 640, 800, 1024}
Typischer Wertebereich: 1 ≤ Width ≤ 32767

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des virtuellen Zielbildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {240, 256, 480, 512, 600, 1024}
Typischer Wertebereich: 1 ≤ Height ≤ 32767

▷ **Interpolation** (input_control) ..... string ⇝ *string*
Interpolationsverfahren für die Transformation.
Defaultwert: 'nearest_neighbor'
Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Siehe auch

polar_trans_image_ext, polar_trans_image_inv, polar_trans_region, polar_trans_contour_xld, polar_trans_contour_xld_inv

Modul

Foundation

projective_trans_region (Regions : TransRegions : HomMat2D, Interpolation :)

Anwenden einer projektiven Transformation auf eine Region.

projective_trans_region wendet die durch die homogene Matrix HomMat2D beschriebene projektive Transformation auf die Regionen in Regions an und liefert die Ergebnisregionen in TransRegions zurück.

Zur Berechnung und Interpretation der Matrix siehe projective_trans_image.

Wird *'clip_region'* durch set_system('clip_region', 'true') auf seinen Grundzustand *'true'* gesetzt, oder ist die Transformation entartet, erzeugt also unendlich große Regionen, so wird die Ausgaberegion durch das Rechteck mit linker oberer Ecke (0, 0) und rechter unterer Ecke (*'width'*, *'height'*) beschnitten, wobei *'width'* und *'height'* Systemvariablen sind (siehe get_system). Falls *'clip_region'* auf *'false'* gesetzt wird, so wird die Ausgaberegion nur durch die maximale unterstützte Koordinatengröße *'MAX_FORMAT'* beschränkt. Dies kann zu sehr speicher- und zeitintensiven Berechnungen führen, ist also mit Vorsicht zu benutzen.

Achtung

Der Transformation liegt das gleiche Koordinatensystem zugrunde, das auch bei dem Operator affine_trans_pixel verwendet wird. Das bedeutet, dass die Transformation nicht mit der Transformationsmatrix HomMat2D, sondern mit einer modifizierte Version davon durchgeführt wird. Eine Transformation mit projective_trans_region entspricht für jeden Punkt (Row_i, Col_i) der Region der folgenden Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt):

$$\begin{pmatrix} RowTrans_i \\ ColTrans_i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & -0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D} \cdot \begin{bmatrix} 1 & 0 & +0.5 \\ 0 & 1 & +0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} Row_i \\ Col_i \\ 1 \end{pmatrix}$$

Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von der Region mit Operatoren wie z.B. area_center abgeleitet wurden. Wenn z.B. eine rotationssymmetrische Region unter Verwendung von hom_mat2d_rotate um ihren mit area_center bestimmten Schwerpunkt rotiert wird, so liegt die transformierte Region nicht auf der Originalregion. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von projective_trans_region die Transformationsmatrix HomMat2D durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
projective_trans_region(Region, TransRegion, HomMat2DAdapted, 'bilinear')
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegionen.
- ▷ **TransRegions** (output_object) ..... region(-array) ⇝ *object*
  Ergebnisregionen.
- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **Interpolation** (input_control) ..... string ⇝ *string*
  Interpolationsmethode für die Transformation.
  Defaultwert: 'bilinear'
  Werteliste: Interpolation ∈ {'nearest_neighbor', 'bilinear'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`vector_to_proj_hom_mat2d`, `hom_vector_to_proj_hom_mat2d`, `proj_match_points_ransac`, `proj_match_points_ransac_guided`, `hom_mat3d_project`

Siehe auch

`projective_trans_image`, `projective_trans_image_size`, `projective_trans_contour_xld`, `projective_trans_point_2d`, `projective_trans_pixel`

Modul

Foundation

`transpose_region` (Region : Transposed : Row, Column :)

Ausführen einer Punktspiegelung einer Region.

`transpose_region` führt eine Punktspiegelung der Eingaberegionen durch. Der Symmetriepunkt S wird durch `Column` und `Row` bestimmt. Der Bildpunkt P' eines Punktes P ist durch folgende Forderung festgelegt:

Ist $P = S$, dann ist $P' = S$. Der Punkt S ist also ein Bezugspunkt. Ist $P \neq S$, dann ist S der Mittelpunkt der Strecke PP'. Die Abbildungsgleichungen der dieser Punktspiegelung mit dem Zentrum $S =$ `Column`, `Row` ermittelt man aus den Beziehungen:

$$\begin{aligned} \texttt{Column} &= \frac{x + x'}{2} \\ \texttt{Row} &= \frac{y + y'}{2}. \end{aligned}$$

Werden `Row` und `Column` auf den Ursprung gesetzt, so erhält man die in der Morphologie übliche Punktspiegelung. Typischerweise wird `transpose_region` zur Punktspiegelung eines strukturierenden Elementes verwendet.

Parameter

▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
Regionen die gespiegelt werden.

▷ **`Transposed`** (output_object) ..... region(-array) ⇝ *object*
Transponierte Regionen

▷ **`Row`** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Bezugspunktes.
Defaultwert: 0
Wertevorschläge: `Row` ∈ {0, 64, 128, 256, 512}
Typischer Wertebereich: 0 ≤ `Row` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **`Column`** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Bezugspunktes.
Defaultwert: 0
Wertevorschläge: `Column` ∈ {0, 64, 128, 256, 512}
Typischer Wertebereich: 0 ≤ `Column` ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `transpose_region` den Wert TRUE. Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion lässt sich wie folgt kontrollieren:

- keine Region:
  `set_system('no_object_result',<RegionResult>)`
- leere Region:
  `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`reduce_domain`, `select_shape`, `area_center`, `connection`

Siehe auch

`dilation1`, `opening`, `closing`

Modul

Foundation

`zoom_region` (`Region` : `RegionZoom` : `ScaleWidth`, `ScaleHeight` :)

Zoomen von Regionen.

`zoom_region` vergrößert oder verkleinert die Regionen `Region` in x- und y-Richtung um die angegebenen Skalierungsfaktoren `ScaleWidth` und `ScaleHeight`.

Parameter

▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
Zu transformierende Region(en).

▷ **`RegionZoom`** (output_object) ..... region(-array) ⇝ *object*
Gezoomte Region(en).
Parameteranzahl: RegionZoom == Region

▷ **`ScaleWidth`** (input_control) ..... extent.x ⇝ *real*
Skalierungsfaktor in x-Richtung.
Defaultwert: 2.0
Wertevorschläge: `ScaleWidth` ∈ {0.25, 0.5, 1.0, 2.0, 3.0}
Typischer Wertebereich: 0.0 ≤ `ScaleWidth` ≤ 100.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.5

▷ **`ScaleHeight`** (input_control) ..... extent.y ⇝ *real*
Skalierungsfaktor in y-Richtung.
Defaultwert: 2.0
Wertevorschläge: `ScaleHeight` ∈ {0.25, 0.5, 1.0, 2.0, 3.0}
Typischer Wertebereich: 0.0 ≤ `ScaleHeight` ≤ 100.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.5

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `connection`, `regiongrowing`, `pouring`

Nachfolger

`select_shape`, `disp_region`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Foundation

22.3 Mengen-Operationen

`complement` `( Region : RegionComplement : : )`

Berechnen des Komplements einer Region.

`complement` berechnet das Komplement der eingegebenen Region(en).

Wenn das Systemflag *'clip_region'* auf *'true'* gesetzt ist, was der Normalfall ist, wird die Differenz aus dem größten aktuellen Bild (siehe `reset_obj_db`) und der Region berechnet.

Wenn das Systemflag *'clip_region'* auf *'false'* steht (siehe `set_system`), würde die resultierende Region unendlich groß werden. Um dies zu verhindern, wird das Komplement nur virtuell ausgeführt, d.h., es wird der Eintrag (Flag) für das Komplement in `Region` auf TRUE gesetzt. Für die Berechnung von Folgeoperationen werden dann die Gesetze von de Morgan verwendet. Sinnvoll ist die Ausführung von `complement` mit dem Systemflag auf *'false'* nur zur Vermeidung von Randeffekten, z.B., wenn der zu bearbeitende Ausschnitt über die Bildgröße hinausgeht oder wenn er kleiner als die Bildgröße ist. In letzterem Fall müsste dann das gewünschte Clipping explizit gesetzt werden. Hat man keinen Grund, den Operator mit *'clip_region'='false'* aufzurufen, braucht diese Einstellung aber für andere Operationen, empfiehlt es sich, das Systemflag kurzzeitig auf *'true'* und nach Anwendung von `complement` wieder auf *'false'* zu setzen, da ansonsten negative Regionen aus Folgeoperationen resultieren können

Parameter

▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
Eingaberegion(en).

▷ **`RegionComplement`** (output_object) ..... region(-array) ⇝ *object*
Berechnete Komplement(e).
Parameteranzahl: RegionComplement == Region

Ergebnis

`complement` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection, regiongrowing, pouring, class_ndim_norm

Nachfolger

select_shape

Siehe auch

difference, union1, union2, intersection, reset_obj_db, set_system

Modul

Foundation

difference (Region, Sub : RegionDifference : :)

Berechnen der Differenz von Mengen von Regionen.

difference berechnet die flächenmäßige Differenz von Regionen:

(Regionen aus Region) − (Regionen aus Sub)

Eine Ergebnisregion berechnet sich aus der Eingaberegion (Region) minus aller Punkte der Regionen aus Sub. Intern werden alle Regionen aus Sub zunächst vereinigt, bevor die Differenz jeder einzelnen Region aus Region mit der vereinigten Region berechnet wird.

Achtung

Leere Eingaberegionen sind bei beiden Parametern zulässig. Es ist zu beachten, dass leere Regionen entstehen können. Hierbei ist der Wert von *'store_empty_region'* zu beachten.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Zu bearbeitende Region(en).
- ▷ **Sub** (input_object) ..... region(-array) ⇝ *object*
  Diese Region(en) (ihre Vereinigung) werden von Region abgezogen.
- ▷ **RegionDifference** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Differenz.

Beispiel

```
* provides the region X without the points in Y
difference(X,Y,RegionDifference)
```

Komplexität

Sei N die Anzahl der Region und F_1 deren mittlere Fläche und F_2 die Gesamtfläche aller Regionen aus Sub, dann ist die Laufzeitkomplexität: $O(F_1 * log(F_1) + N * (sqrt(F_1) + sqrt(F_2)))$.

Ergebnis

difference liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, connection, regiongrowing, pouring, class_ndim_norm

Nachfolger

select_shape, disp_region

Siehe auch

intersection, union1, union2, complement, symm_difference

Modul

Foundation

intersection (Region1, Region2 : RegionIntersection : :)

Berechnen des Durchschnitts von Mengen von Regionen.

intersection bildet den Durchschnitt aller Regionen aus Region1 mit den Regionen aus Region2. Jede Region aus Region1 wird mit allen Regionen aus Region2 geschnitten. D.h., intern werden alle Regionen aus Region2 zunächst vereinigt, bevor der Durchschnitt jeder einzelnen Region aus Region1 mit der vereinigten Region berechnet wird. Die Reihenfolge der Regionen aus Region1 ist identisch mit der Reihenfolge der bearbeiteten Regionen in RegionIntersection (d.h. erste Eingaberegion = erste geschnittene Region, usw.).

Achtung

Leere Eingaberegionen sind zulässig. Da leere Regionen auftreten können ist das Flag *'store_empty_region'* zu beachten.

Parameter

- ▷ **Region1** (input_object) ..... region(-array) ⇝ *object*
  Jede Region aus Region1 wird mit der Vereinigung der Regionen aus Region2 geschnitten.
- ▷ **Region2** (input_object) ..... region(-array) ⇝ *object*
  Regionen, die für die Berechnung vereinigt werden.
- ▷ **RegionIntersection** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Durchschnittsbildung.
  Parameteranzahl: RegionIntersection <= Region1

Komplexität

Sei N die Anzahl der Regionen in Region1 und F_1 deren mittlere Fläche und F_2 die Gesamtfläche aller Regionen aus Region2, dann ist die Laufzeitkomplexität: $O(F_1 \log(F_1) + N * (\sqrt{F_1} + \sqrt{F_2}))$.

Ergebnis

intersection liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, connection, regiongrowing, pouring

Nachfolger

select_shape, disp_region

Siehe auch

union1, union2, complement

Modul

Foundation

symm_difference (Region1, Region2 : RegionDifference : :)

Berechnen der symmetrischen Mengendifferenz von Regionen.

symm_difference berechnet die symmetrische Mengendifferenz von Regionen. Intern werden alle Regionen aus Region2 zunächst vereinigt, bevor die Differenz jeder einzelnen Region aus Region1 mit der vereinigten Region berechnet wird. Zwei mögliche Definitionen der symmetrischen Mengendifferenz sind aus dem Beispiel unten zu entnehmen. Eine dritte Definition ist, die symmetrische Mengendifferenz als das exklusive Oder der beiden Regionen zu betrachten.

Achtung

Leere Eingaberegionen sind bei beiden Parametern zulässig. Es ist zu beachten, dass leere Regionen entstehen können. Hierbei ist der Wert von *'store_empty_region'* zu beachten.

Parameter

- ▷ **Region1** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegion 1.
- ▷ **Region2** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegion 2.
- ▷ **RegionDifference** (output_object) ..... region(-array) ⇝ *object*
  Ergebnis der Differenz.

Beispiel

```
* Simulate the symmetric difference of Region1 and Region2 with
* difference and union:
difference(Region1, Region2, Diff1)
difference(Region2, Region1, Diff2)
union2(Diff1, Diff2, Difference)

* Simulate the symmetric difference of Region1 and Region2 with
* union, intersection, and difference:
union2(Region1, Region2, Union)
intersection(Region1, Region2, Intersection)
difference(Union, Intersection, Difference)
```

Ergebnis

symm_difference liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

select_shape, disp_region

Siehe auch

intersection, union1, union2, complement, difference

Modul

Foundation

union1 (Region : RegionUnion : :)

Vereinigen von Regionen.

union1 bestimmt die Vereinigung der Eingaberegionen und erzeugt daraus eine Ausgaberegion, die alle Punkte der Eingaberegionen umfasst.

Parameter

▷ **Region** (input_object) . . . . . region-array ⇝ *object*
Regionen, die vereinigt werden sollen.

▷ **RegionUnion** (output_object) . . . . . region ⇝ *object*
Ergebnisregion, die die Vereinigung aller Eingaberegionen ist.
Parameteranzahl: RegionUnion <= Region

Beispiel

```
* Union of segmentation results:
threshold(Image,Region1,128,255)
dyn_threshold(Image,Mean,Region2,5,'light')
concat_obj(Region1,Region2,Regions)
union1(Regions,RegionUnion)
```

Komplexität

Sei F die Summe aller Flächen der Eingaberegionen, dann ist die Laufzeitkomplexität: $O(log(sqrt(F))*sqrt(F))$.

Ergebnis

`union1` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `connection`, `regiongrowing`, `pouring`

Nachfolger

`select_shape`, `disp_region`

Alternativen

`union2`

Siehe auch

`intersection`, `complement`

Modul

Foundation

union2 (Region1, Region2 : RegionUnion : :)

Vereinigen von Regionen mit zwei Eingabeparametern.

`union2` vereinigt jede Region aus `Region1` mit der Vereinigung aller Regionen aus `Region2`. D.h., intern werden alle Regionen aus `Region2` zunächst vereinigt, bevor die Vereinigung jeder einzelnen Region aus `Region1` mit der vereinigten Region berechnet wird. `union2` ist also nicht kommutativ!

Parameter

▷ **Region1** (input_object) . . . . . region(-array) ⇝ *object*
Jede Region aus Region1 wird mit der Vereinigung aller Regionen aus Region2 vereinigt.

▷ **Region2** (input_object) . . . . . region(-array) ⇝ *object*
Die Vereinigung der Regionen aus Region2 wird mit jeder Region aus Region1 vereinigt.

▷ **RegionUnion** (output_object) . . . . . region(-array) ⇝ *object*
Ergebnisregionen, die die Vereinigung mit den Eingaberegionen aus Region2 sind.
Parameteranzahl: RegionUnion == Region1

Komplexität

Sei F die Summe aller Flächen der Eingaberegionen, dann ist die Laufzeitkomplexität: $O(log(sqrt(F))*sqrt(F))$.

Ergebnis

`union2` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `connection`, `regiongrowing`, `pouring`

Nachfolger

`select_shape`, `disp_region`

Alternativen

`union1`

Siehe auch

`intersection`, `complement`

Modul

Foundation

22.4 Merkmale

Dieses Kapitel beschreibt Operatoren, mit denen auf die unterschiedlichen Features von Regionen zugegriffen werden kann.

Liste von Features

Im Folgenden werden die Features kurz illustriert.

'area': Fläche des Objektes



'row': Zeilenindex des Schwerpunkts

'column': Spaltenindex des Schwerpunkts

'row1': Zeilenindex der linken oberen Ecke

'column1': Spaltenindex der linken oberen Ecke

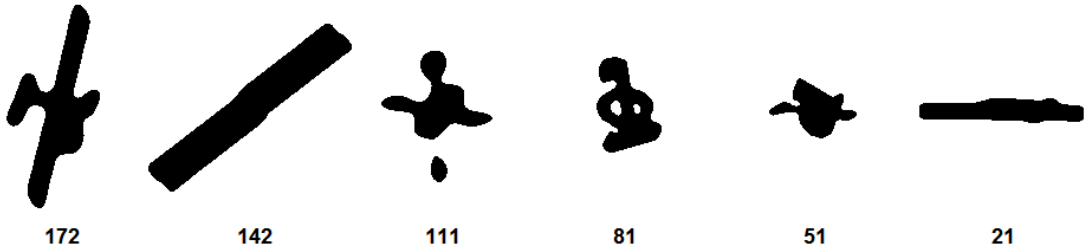
'row2': Zeilenindex der rechten unteren Ecke

'column2': Spaltenindex der rechten unteren Ecke

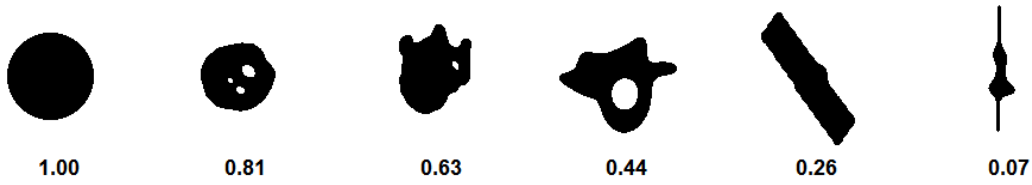
'width': Breite der Region (parallel zur Koordinatenachse)



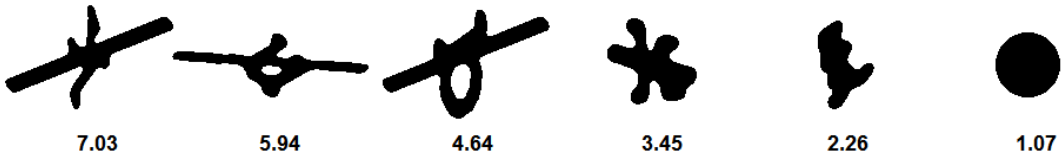
'height': Höhe der Region (parallel zur Koordinatenachse)



'circularity': Kreisförmigkeit (vgl. `circularity`)



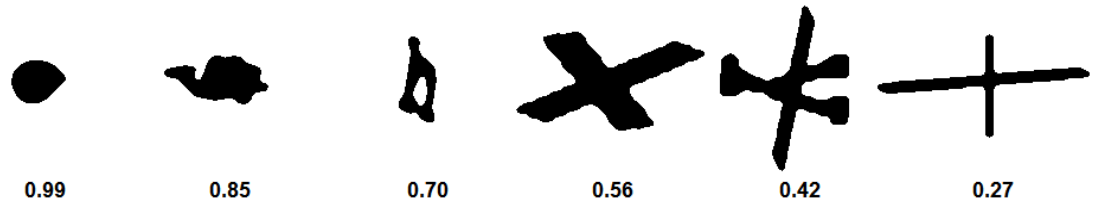
'compactness': Kompaktheit (vgl. `compactness`)



'contlength': Gesamtlänge der Kontur (vgl. `contlength`)



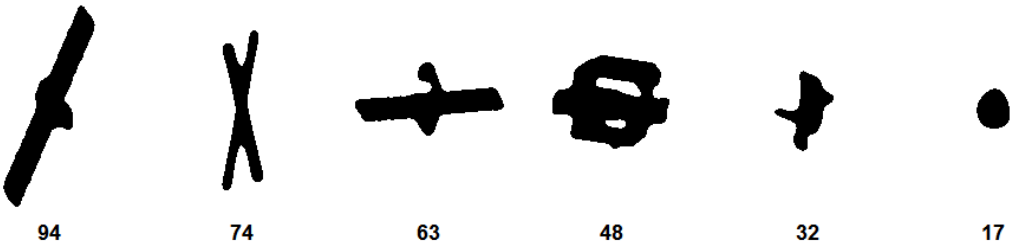
'convexity': Konvexität (vgl. `convexity`)



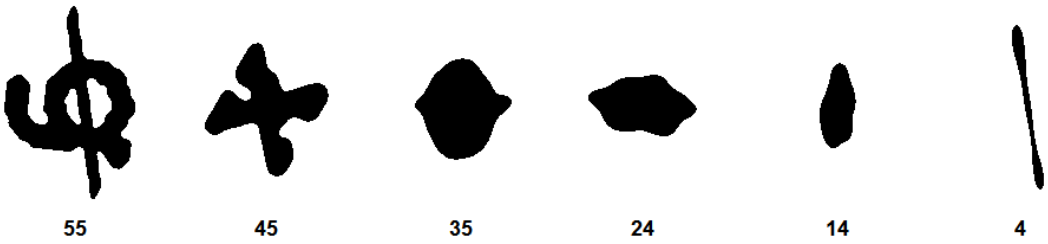
'rectangularity': Rechteckigkeit (vgl. `rectangularity`)



'ra': Hauptradius der äquivalenten Ellipse (vgl. `elliptic_axis`)



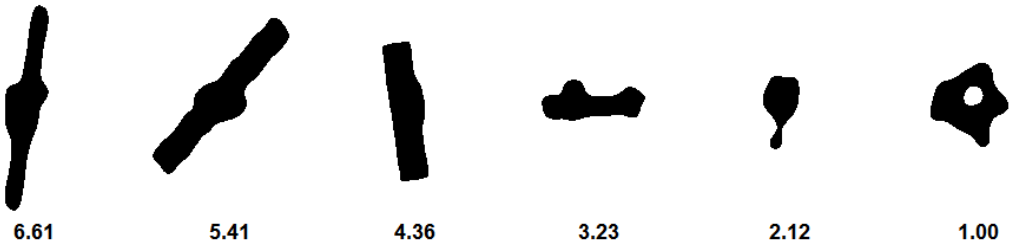
'rb': Nebenradius der äquivalenten Ellipse (vgl. `elliptic_axis`)



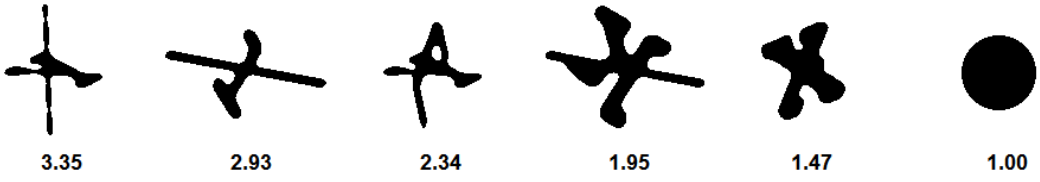
'phi': Orientierung der äquivalenten Ellipse (vgl. `elliptic_axis`)



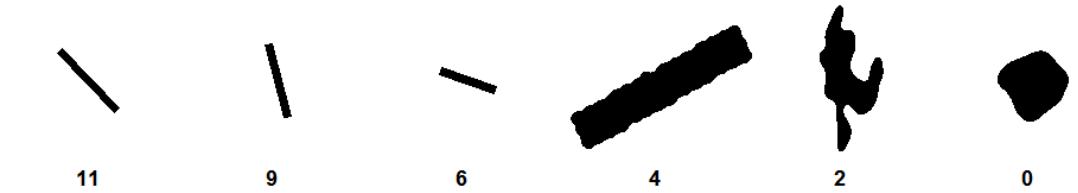
'anisometry' Anisometrie (vgl. `eccentricity`)



'bulkiness' Bulkiness (vgl. Operator `eccentricity`)



'struct_factor' Struktur Faktor (vgl. Operator `eccentricity`)



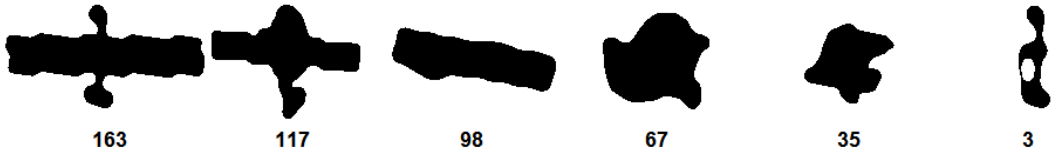
'outer_radius': Radius des kleinsten umschließenden Kreises (vgl. `smallest_circle`)



'inner_radius': Radius des größten Inkreises (vgl. `inner_circle`)



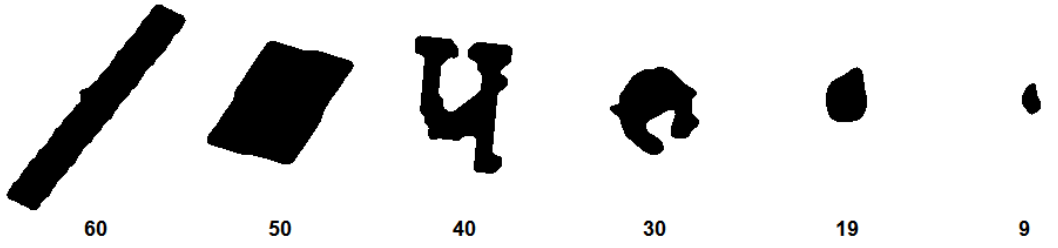
'inner_width': Breite des größten enthaltenen achsenparallelen Rechtecks (vgl. `inner_rectangle1`)



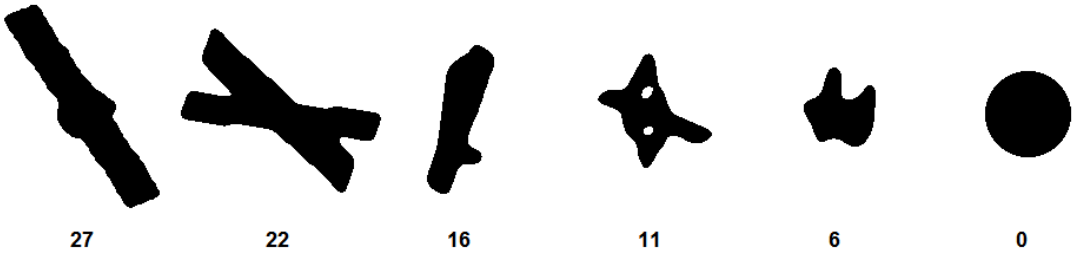
'inner_height': Höhe des größten enthaltenen achsenparallelen Rechtecks (vgl. `inner_rectangle1`)



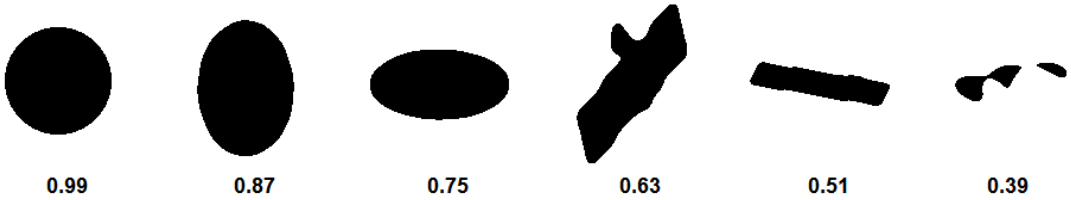
'dist_mean': Mittlerer Abstand zwischen dem Rand der Region und ihrem Schwerpunkt (vgl. Operator `roundness`)



'dist_deviation': Standardabweichung des Abstands vom Regionenrand zum Schwerpunkt (vgl. Operator `roundness`)



'roundness': Rundheit (vgl. Operator `roundness`)



'num_sides': Anzahl Polygonseiten (vgl. Operator `roundness`)



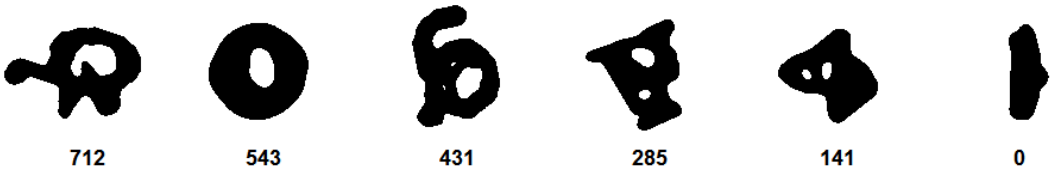
'connect_num': Anzahl der Zusammenhangskomponenten (vgl. Operator `connect_and_holes`)



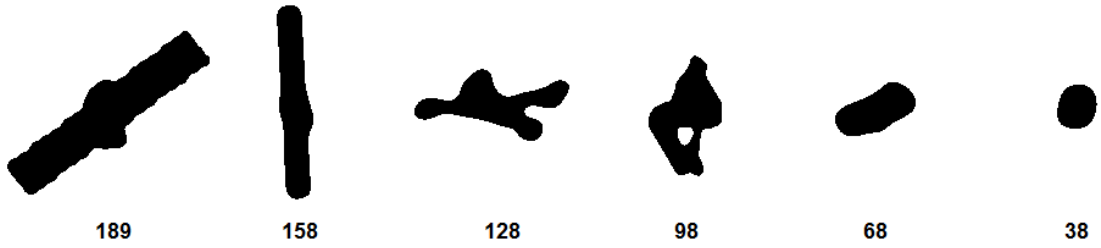
'holes_num': Anzahl der Löcher (vgl. Operator `connect_and_holes`)



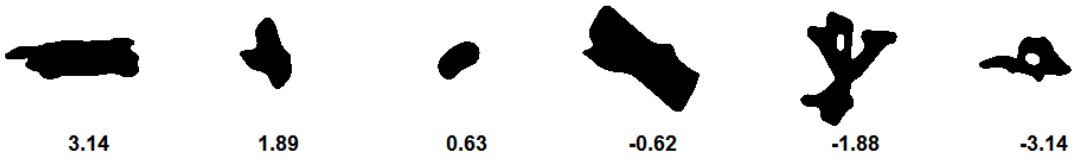
'area_holes:' Fläche der Löcher des Objektes (vgl. Operator `area_holes`)



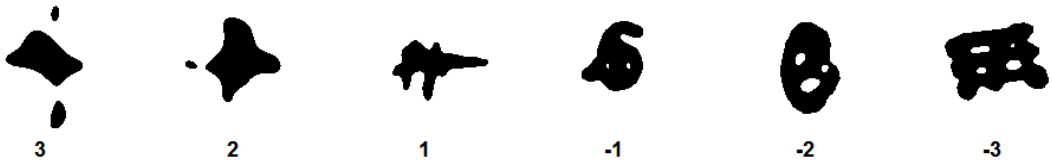
'max_diameter': Maximale Ausdehnung der Region (vgl. Operator `diameter_region`)



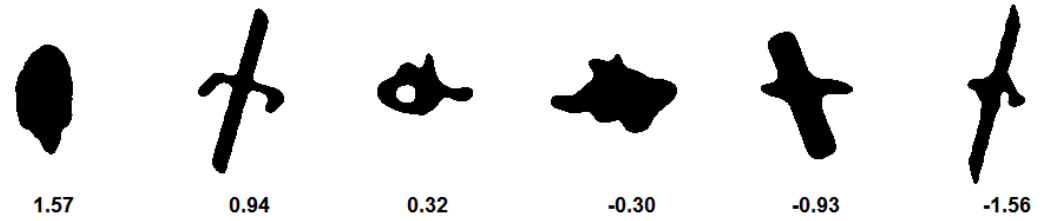
'orientation': Orientierung der Region (vgl. Operator `orientation_region`)



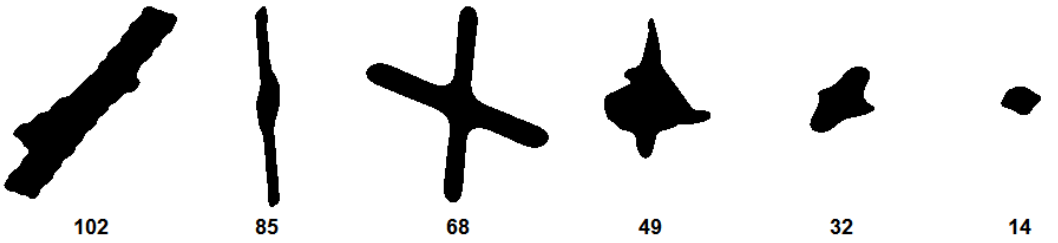
'euler_number': Eulerzahl (vgl. Operator `euler_number`)



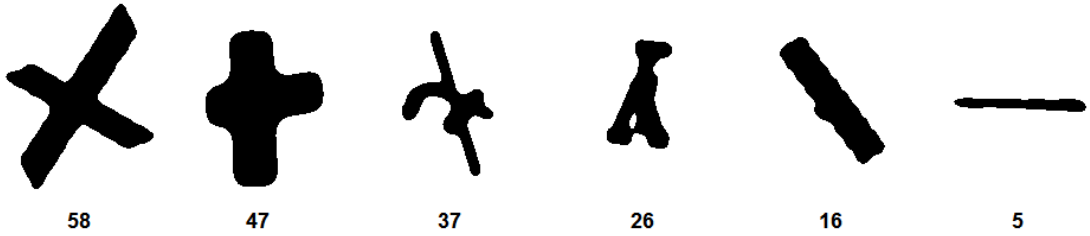
'rect2_phi': Orientierung des kleinsten umschließenden Rechtecks (vgl. Operator `smallest_rectangle2`)



'rect2_len1': Halbe Länge des kleinsten umschließenden Rechtecks (vgl. Operator `smallest_rectangle2`)



'rect2_len2': Halbe Breite des kleinsten umschließenden Rechtecks (vgl. Operator `smallest_rectangle2`)



'moments_m11': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m20': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m02': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_ia': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_ib': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m11_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_m20_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_m02_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_phi1': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_rel_invar)

'moments_phi2': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_rel_invar)

'moments_m21': Geometrische Regionenmomente (vgl. Operator moments_region_3rd)

'moments_m12': Geometrische Regionenmomente (vgl. Operator moments_region_3rd)

'moments_m03': Geometrische Regionenmomente (vgl. Operator moments_region_3rd)

'moments_m30': Geometrische Regionenmomente (vgl. Operator moments_region_3rd)

'moments_m21_invar': Geometrische Regionenmomente (vgl. Operator moments_region_3rd_invar)

'moments_m12_invar': Geometrische Regionenmomente (vgl. Operator moments_region_3rd_invar)

'moments_m03_invar': Geometrische Regionenmomente (vgl. Operator moments_region_3rd_invar)

'moments_m30_invar': Geometrische Regionenmomente (vgl. Operator moments_region_3rd_invar)

'moments_i1': Geometrische Regionenmomente (vgl. Operator moments_region_central)

'moments_i2': Geometrische Regionenmomente (vgl. Operator moments_region_central)

'moments_i3': Geometrische Regionenmomente (vgl. Operator moments_region_central)

'moments_i4': Geometrische Regionenmomente (vgl. Operator moments_region_central)

'moments_psi1': Geometrische Regionenmomente (vgl. Operator moments_region_central_invar)

'moments_psi2': Geometrische Regionenmomente (vgl. Operator moments_region_central_invar)

'moments_psi3': Geometrische Regionenmomente (vgl. Operator moments_region_central_invar)

'moments_psi4': Geometrische Regionenmomente (vgl. Operator moments_region_central_invar)

area_center (Regions : : : Area, Row, Column)

Ermitteln der Fläche und des Schwerpunkts von Regionen.

area_center berechnet die Fläche und den Schwerpunkt der Eingaberegionen. Die Fläche ist definiert als die Anzahl von Bildpunkten einer Region. Der Schwerpunkt berechnet sich als der Mittelwert der Zeilen- bzw. Spaltenkoordinaten aller Punkte.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihrer Fläche unterscheiden.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index der Region in der Eingabe entspricht. Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Region(en).
- ▷ **Area** (output_control) ..... integer(-array) ⇝ *integer*
  Fläche der Region.
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenindex des Schwerpunktes.

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

main()
{
  Tuple   area, row, column;

  HImage   img ("monkey");
  HWindow  w;

  img.Display (w);
  w.Click ();

  HRegionArray   reg = (img >= 164).Connection ();

  reg.Display (w);
  w.Click ();

  area = reg.AreaCenter (&row, &column);

  for (int i = 0; i < reg.Num (); i++)
  {
    cout << "Row    [" << i << "]" << "= " << row[i].D ();
    cout << "\t\tColumn [" << i << "]" << "= " << column[i].D () << endl;
  }

  cout << "Total number of regions: " << reg.Num () << endl;
  return(0);
}
```

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

`area_center` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Siehe auch

`select_shape`

Modul

Foundation

area_holes (Regions : : : Area)

Berechnen der Fläche der Löcher von Regionen.

Der Operator `area_holes` berechnet die Fläche der Löcher in den Eingaberegionen. Die Fläche ist definiert als die Anzahl der Bildpunkte der Löcher. Wenn eine Region mehr als ein Loch hat, wird die Summe der Flächen aller Löcher dieser Region zurückgegeben. Die Art der Nachbarschaft wird mit `set_system ('neighborhood',<4/8>)` festgelegt (Defaulteinstellung ist 8-ter Nachbarschaft).

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in der Fläche ihrer Löcher unterscheiden.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index der Region in der Eingabe entspricht. Bei einer leeren Region hat die Fläche den Wert 0.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Zu untersuchende Region(en).

▷ **Area** (output_control) ..... integer(-array) ⇝ *integer*
Fläche(n) der Löcher der Region(en).

Beispiel

```
read_image(Image,'modules/modules_01')
threshold(Image,Region,50,250)
area_holes(Region,Area)
```

Ergebnis

Der Operator `area_holes` liefert normalerweise den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Siehe auch

`area_center`, `fill_up`

Modul

Foundation

circularity (Regions : : : Circularity)

Ermitteln des Formfaktors für die Kreisförmigkeit einer Region.

`circularity` berechnet die Ähnlichkeit der Eingaberegion mit einem Kreis.

Berechnung: Sei F die Fläche der Region und max der maximale Abstand vom Schwerpunkt zu allen Konturpunkten, dann ist der Formfaktor C definiert als:

$$C' = \frac{F}{(max^2 * \pi)}$$

$$C = \min(1.0, C')$$

Der Formfaktor C ist bei einem Kreis gleich 1. Wenn die Region langgestreckt ist oder Hohlflächen hat, ist C kleiner als 1. `circularity` spricht besonders auf große Ausbuchtungen, Hohlflächen und nicht zusammenhängende Regionen an. Der Wert von C wird auf 1 begrenzt, weil die Pixel-Fläche einer Region nur eine Approximation der Fläche eines echten Kreises sein kann. Dieser Approximationsfehler ist umso größer, je kleiner die Region ist.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihrer Kreisförmigkeit unterscheiden.

Bei einer leeren Region liefert `circularity` (soweit kein anderes Verhalten festgelegt wurde (siehe `set_system`)) den Wert 0. Wird mehr als eine Region übergeben, dann werden die Zahlenwerte des Formfaktors in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Region(en).
- ▷ **`Circularity`** (output_control) ..... real(-array) ⇝ *real*
  Kreisförmigkeit der Eingaberegion(en).
  Zusicherung: 0 <= Circularity && Circularity <= 1.0

Beispiel

```
* Comparison between shape factors of rectangle, circle and ellipse:
gen_rectangle1(R1,10,10,20,20)
gen_rectangle2(R2,100,100,0.0,100,20)
gen_ellipse(E100,100,100,0.0,100,20)
gen_circle(C,100,100,20)
circularity(R1,M_R1)
circularity(R2,M_R2)
circularity(E100,M_E)
circularity(C,M_C)
fwrite_string(FileId,['quadrate:  ',M_R1])
fnew_line(FileId)
fwrite_string(FileId,['rectangle: ',M_R2])
fnew_line(FileId)
fwrite_string(FileId,['ellipse:   ',M_E])
fnew_line(FileId)
fwrite_string(FileId,['circle:    ',M_C])
fnew_line(FileId)
```

Ergebnis

`circularity` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

roundness, compactness, convexity, eccentricity

Siehe auch

area_center, select_shape

Modul

Foundation

compactness (Regions : : : Compactness)

Ermitteln des Formfaktors für die Kompaktheit einer Region.

compactness berechnet die Kompaktheit der Eingaberegionen.

Berechnung: Sei L die Länge der Kontur (siehe contlength) und F die Fläche der Region, dann ist der Formfaktor C definiert als:

$$C' = \frac{L^2}{4F\pi}$$

$$C = \max(1, C')$$

Der Formfaktor C ist bei einem Kreis gleich 1. Ist die Region langgestreckt oder hat sie Hohlflächen, dann ist C größer als 1. compactness spricht auf den Verlauf der Kontur (Rauigkeit) und auf Hohlflächen an. Der Wert von C wird auf größer als 1 beschränkt, weil die Pixel-Fläche einer Region nur eine Approximation der Fläche eines echten Kreises sein kann. Dieser Approximationsfehler ist umso größer, je kleiner die Region ist.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihrer Kompaktheit unterscheiden.

Bei einer leeren Region liefert compactness den Wert 0, soweit kein anderes Verhalten festgelegt wurde (siehe set_system). Wird mehr als eine Region übergeben, dann werden die Zahlenwerte des Formfaktors in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Zu untersuchende Region(en).

▷ **Compactness** (output_control) ..... real(-array) ⇝ *real*
Kompaktheit der Eingaberegion(en).
Zusicherung: Compactness >= 1.0 || Compactness == 0

Ergebnis

compactness liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

convexity, eccentricity

Siehe auch

contlength, area_center, select_shape

Modul

Foundation

connect_and_holes (Regions : : : NumConnected, NumHoles)

Ermitteln der Anzahl der Zusammenhangskomponenten und Hohlflächen.

connect_and_holes berechnet die Anzahl der Zusammenhangskomponenten und die Anzahl der Hohlflächen einer jeden Region aus Regions.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in der Anzahl ihrer Zusammenhangskomponenten und der Anzahl ihrer Hohlflächen unterscheiden.

Wird mehr als eine Region übergeben, dann werden die Zahlenwerte der Ausgabesteuerparameter NumConnected und NumHoles jeweils in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Region(en).
- ▷ **NumConnected** (output_control) ..... integer(-array) ⇝ *integer*
  Anzahl der Zusammenhangskomponenten einer Region.
- ▷ **NumHoles** (output_control) ..... integer(-array) ⇝ *integer*
  Anzahl der Hohlflächen einer Region.

Ergebnis

connect_and_holes liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

euler_number

Siehe auch

connection, fill_up, fill_up_shape, union1

Modul

Foundation

contlength (Regions : : : ContLength)

Ermitteln der Konturlänge einer Region.

contlength berechnet für jede Region aus Regions die Gesamtlänge der Kontur (Summe über alle Zusammenhangskomponenten der Region). Dabei wird der Abstand von zwei benachbarten Konturpunkten parallel zu den Koordinatenachsen mit 1, der Abstand in der Diagonalen mit $\sqrt{2}$ bewertet.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in der Konturlänge unterscheiden.

Wird mehr als eine Region übergeben, dann werden die Zahlenwerte der Konturlänge in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position einer Region im Eingabetupel entspricht. Bei einer leeren Region liefert contlength den Wert 0.

Achtung

Die Kontur von Hohlflächen wird nicht mitgerechnet.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Region(en).
- ▷ **ContLength** (output_control) ..... real(-array) ⇝ *real*
  Konturlänge der Eingaberegion(en).
  Zusicherung: ContLength >= 0

Beispiel

```
#include "HalconCpp.h"
#include <iostream>
using namespace HalconCpp;

int main (int argc, char *argv[])
{
  HWindow         w(10,10,512,512);
  HRegion         reg;
  int             NumOfElements = 3;

  std::cout << "Draw " << NumOfElements << " regions " << std::endl;
  GenEmptyObj(&reg);
  for (int i=0; i < NumOfElements; i++)
  {
    reg = reg.ConcatObj(w.DrawRegion());
  }

  HTuple circ = reg.Circularity ();
  HTuple cont = reg.Contlength ();

  for (int i = 0; i < NumOfElements; i++)
  {
    std::cout << "region " << i+1 << ": \tcircularity = " << circ[i].D() <<
                 "\tcontour length =" << cont[i].D() << std::endl;
  }
  std::cout << "Click into the graphics window to end." << std::endl;
  w.Click ();
  return(0);
}
```

Ergebnis

contlength liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Nachfolger

`get_region_contour`

Alternativen

`compactness`

Siehe auch

`area_center`, `get_region_contour`

Modul

Foundation

`convexity` (Regions : : : Convexity)

Ermitteln des Formfaktors für die Konvexität einer Region.

`convexity` berechnet die Konvexität jeder Eingaberegion aus `Regions`.

Berechnung: Sei F_c die Fläche der konvexen Hülle und F_o die Originalfläche der Region, dann ist der Formfaktor C definiert als:

$$C = \frac{F_o}{F_c}$$

Der Formfaktor C ist gleich 1, wenn die Region konvex ist (z.B. Rechteck, Kreis etc.). Sind Einbuchtungen oder Hohlflächen vorhanden, dann ist C kleiner als 1.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihrer Konvexität unterscheiden.

Bei einer leeren Region liefert `convexity` (soweit kein anderes Verhalten festgelegt wurde (siehe `set_system`)) den Wert 0. Wird mehr als eine Region übergeben, dann werden die Zahlenwerte des Formfaktors in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Parameter

▷ **`Regions`** (input_object) ..... region(-array) ⇝ *object*
Zu untersuchende Region(en).

▷ **`Convexity`** (output_control) ..... real(-array) ⇝ *real*
Konvexität der Eingaberegion(en).
Zusicherung: Convexity <= 1

Ergebnis

`convexity` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Siehe auch

select_shape, area_center, shape_trans

Modul

Foundation

diameter_region (Regions : : : Row1, Column1, Row2, Column2, Diameter)

Ermitteln des maximalen Abstands zweier Randpunkte einer Region.

diameter_region berechnet den maximalen Abstand zweier Randpunkte einer Region. Es werden die Koordinaten der beiden Extrempunkte und der Abstand zwischen diesen Punkten zurückgegeben.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in dem maximalen Abstands zweier Randpunkte unterscheiden.

Achtung

Bei leerer Region kann das Ergebnis von Row1, Column1, Row2 und Column2 (alle sind 0) zu Konfusionen führen.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **Row1** (output_control) ..... line.begin.y(-array) ⇝ *integer*
  Zeilenindex des ersten Extrempunktes.
- ▷ **Column1** (output_control) ..... line.begin.x(-array) ⇝ *integer*
  Spaltenindex des ersten Extrempunktes.
- ▷ **Row2** (output_control) ..... line.end.y(-array) ⇝ *integer*
  Zeilenindex des zweiten Extrempunktes.
- ▷ **Column2** (output_control) ..... line.end.x(-array) ⇝ *integer*
  Spaltenindex des zweiten Extrempunktes.
- ▷ **Diameter** (output_control) ..... number(-array) ⇝ *real*
  Abstand der beiden Extrempunkte.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

diameter_region liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, runlength_features

Nachfolger

disp_line

Alternativen

smallest_rectangle2

Modul

Foundation

eccentricity (Regions : : : Anisometry, Bulkiness, StructureFactor)

Ermitteln der aus den Ellipsenparametern abgeleiteten Formmerkmale.

eccentricity berechnet für die Regionen in Regions die drei Formmerkmale Anisometry, Bulkiness und StructureFactor:

$$\begin{aligned} \texttt{Anisometry} &= \frac{Ra}{Rb} \\ \texttt{Bulkiness} &= \frac{\pi \cdot Ra \cdot Rb}{F} \\ \texttt{StructureFactor} &= \texttt{Anisometry} \cdot \texttt{Bulkiness} - 1 \end{aligned}$$

wobei Ra und Rb die Ellipsenradien bezeichnen (siehe elliptic_axis) und F für die Fläche der Region steht (siehe area_center).

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in Anisometry, Bulkiness und StructureFactor unterscheiden.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Achtung

Es ist zu beachten, dass, wie bei allen regionenmomentenbasierten Operatoren, die Pixel als mathematische Punkte ohne flächenhafte Ausdehnung betrachtet werden, die durch die Mitte des jeweiligen Pixels gegeben sind (siehe die Dokumentation von elliptic_axis). Dies kann dazu führen, dass nichtleere Regionen einen Wert Rb = *0* besitzen. In diesem Fall werden die Ausgabemerkmale, die eine Division durch Rb erfordern, auf *0* gesetzt. Insbesondere haben also Regionen, die aus einem Punkt bestehen oder deren Punkte exakt auf einer Geraden liegen (also z.B. ein Pixel hohe horizontale Regionen oder ein Pixel breite vertikale Regionen), eine Anisometrie von *0*.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Region(en).
- ▷ **Anisometry** (output_control) ..... real(-array) ⇝ *real*
  Formmerkmal (bei einem Kreis = 1.0).
  Zusicherung: Anisometry >= 1.0
- ▷ **Bulkiness** (output_control) ..... real(-array) ⇝ *real*
  Berechnetes Formmerkmal.
- ▷ **StructureFactor** (output_control) ..... real(-array) ⇝ *real*
  Berechnetes Formmerkmal.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

eccentricity liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Siehe auch

elliptic_axis, moments_region_2nd, select_shape, area_center

Modul

Foundation

elliptic_axis (Regions : : : Ra, Rb, Phi)

Ermitteln der Parameter der äquivalenten Ellipse.

Der Operator elliptic_axis berechnet die Radien Ra und Rb sowie die Orientierung Phi der äquivalenten Ellipse mit der gleichen Orientierung und dem gleichen Seitenverhältnis wie die Eingaberegion Regions. Mehrere Eingaberegionen können in Form eines Tupels übergeben werden. Ra repräsentiert die Länge des Hauptradius der Ellipse während der Radius Rb den Nebenradius der Ellipse repräsentiert. Die Orientierung der Hauptachse bezüglich der x-Achse wird im Winkel Phi zurückgegeben und ist im Bogenmaß angegeben. Die Hauptachse der Ellipse entspricht der Hauptachse des Trägheitsmoments der Eingaberegion.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in Phi, Ra und Rb unterscheiden.

Berechnung:

Seien die Momente M_{20}, M_{02} und M_{11} normiert auf die Fläche gegeben (siehe moments_region_2nd), dann berechnen sich die Radien Ra und Rb zu:

$$\mathtt{Ra} = \frac{\sqrt{8(M_{20} + M_{02} + \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}}{2}$$

$$\mathtt{Rb} = \frac{\sqrt{8(M_{20} + M_{02} - \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}}{2}$$

Die Orientierung Phi ist definiert durch:

$$\mathtt{Phi} = -0.5atan2(2M_{11}, M_{02} - M_{20})$$

Wird mehr als eine Region übergeben, werden die Ergebnisse in Form von Tupeln zurückgeliefert, wobei der Index eines Tupelelements dem Index der zugehörigen Eingaberegion entspricht.

Wird eine leere Region übergeben, haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system('no_object_result',<Result>)).

Achtung

Es ist zu beachten, dass, wie bei allen regionenmomentenbasierten Operatoren, die Pixel als mathematische Punkte ohne flächenhafte Ausdehnung betrachtet werden, die durch die Mitte des jeweiligen Pixels gegeben sind. Daraus folgt, dass Ra und Rb auch den Wert *0* annehmen können. Für eine leere Region und eine Region, die aus genau einem Punkt besteht, wird folglich Ra = Rb = *0* zurückgeliefert. Weiterhin wird für Regionen, deren Punkte genau auf einer Geraden liegen (also z.B. ein Pixel hohe horizontale Regionen oder ein Pixel breite vertikale Regionen) Rb = *0* zurückgeliefert.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegion(en).
- ▷ **Ra** (output_control) ..... real(-array) ⇝ *real*
  Hauptradius (normiert auf die Fläche).
  Zusicherung: Ra >= 0.0
- ▷ **Rb** (output_control) ..... real(-array) ⇝ *real*
  Nebenradius (normiert auf die Fläche).
  Zusicherung: Rb >= 0.0 && Rb <= Ra

▷ **Phi** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Winkel zwischen Hauptradius und x-Achse im Bogenmaß.
Zusicherung: - pi / 2 < Phi && Phi <= pi / 2

Beispiel

```
read_image(Image,'fabrik')
regiongrowing(Image,Seg,5,5,6,100)
elliptic_axis(Seg,Ra,Rb,Phi)
area_center(Seg,_,Row,Column)
gen_ellipse(Ellipses,Row,Column,Phi,Ra,Rb)
dev_set_draw('margin')
dev_display(Ellipses)
```

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

`elliptic_axis` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Nachfolger

`gen_ellipse`

Alternativen

`smallest_rectangle2`, `orientation_region`

Siehe auch

`moments_region_2nd`, `select_shape`, `set_shape`

Literatur

R. Haralick, L. Shapiro „Computer and Robot Vision“ Addison-Wesley, 1992, pp. 73-75

Modul

Foundation

euler_number (Regions : : : EulerNumber)

Berechnen der Eulerzahl.

Der Operator `euler_number` berechnet die Eulerzahl, d.h. die Differenz von der Anzahl der Zusammenhangskomponenten und der Anzahl der Hohlflächen.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihrer Eulerzahl unterscheiden.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Zu untersuchende Region(en).

▷ **EulerNumber** (output_control) ..... integer(-array) ⇝ *integer*
Berechnete Eulerzahl.

Ergebnis

`euler_number` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`connect_and_holes`

Modul

Foundation

find_neighbors (Regions1, Regions2 : : MaxDistance : RegionIndex1, RegionIndex2)

Suchen von direkten Nachbarn.

`find_neighbors` bestimmt benachbarte Regionen. `Regions1` und `Regions2` enthalten dabei die zu testenden Regionen. `Regions1` kann dabei auf drei verschiedene Arten besetzt werden:

- `Regions1` ist leer:
  In diesem Fall werden alle Regionen in `Regions2` permutativ auf Nachbarschaft getestet.
- `Regions1` besteht aus einer Region:
  Die Regionen von `Regions1` werden mit allen Regionen in `Regions2` verglichen.
- `Regions1` besteht aus gleich vielen Regionen wie `Regions2`:
  Hier werden jeweils die Regionen an n-ter Position in `Regions1` und `Regions2` auf die Nachbarschaftsbeziehung untersucht.

`find_neighbors` verwendet die Chessboard-Norm bei der Berechnung des Abstandes. Der maximale Abstand wird mit `MaxDistance` angegeben. Zueinander benachbarte Regionen stehen an n-ter Position in `RegionIndex1` und `RegionIndex2`, d.h. Region mit Index `RegionIndex1`[n] aus `Regions1` ist benachbart zur Region mit Index `RegionIndex2`[n] aus `Regions2`.

Achtung

Verdeckte Regionen werden nicht gefunden!

Parameter

▷ **Regions1** (input_object) ..... region(-array) ⇝ *object*
Ausgangsregionen.

▷ **Regions2** (input_object) ..... region(-array) ⇝ *object*
Vergleichsregionen.

▷ **MaxDistance** (input_control) ..... integer ⇝ *integer*
Maximaler Abstand der Regionen.
Defaultwert: 1
Wertevorschläge: MaxDistance ∈ {1, 2, 3, 4, 5, 6, 7, 8, 10, 15, 20, 50}
Typischer Wertebereich: 1 ≤ MaxDistance ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **RegionIndex1** (output_control) ..... integer-array ⇝ *integer*
Indizes der gefundenen Regionen aus Regions1.

▷ **RegionIndex2** (output_control) ..... integer-array ⇝ *integer*
Indizes der gefundenen Regionen aus Regions2.

Ergebnis

find_neighbors liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, regiongrowing, connection

Siehe auch

spatial_relation, select_region_spatial, expand_region, distance_transform, interjacent, boundary

Modul

Foundation

get_region_index (Regions : : Row, Column : Index)

Ermitteln des Index aller Regionen, die einen übergebenen Punkt enthalten.

get_region_index liefert den Index aller Regionen in Regions (Wertebereich 1 bis n), die den Testpunkt (Row,Column) enthalten, d.h.:

$$|\texttt{Regions}[n] \cap \{(\texttt{Row}, \texttt{Column})\}| \neq \emptyset$$

Die zurückgegebenen Indizes können z.B. in select_obj verwendet werden, um die Regionen, die den Testpunkt enthalten, zu selektieren.

Achtung

Es kann vorkommen, dass mehr als eine Region den Punkt enthält, falls sich die Regionen überlappen. In diesem Fall werden all diese Regionen ausgegeben. Falls keine Region den angegebenen Punkt enthält, wird das leere Tupel (=keine Region) ausgegeben.

Parameter

▷ **Regions** (input_object) ..... region-array ⇝ *object*
Zu untersuchende Regionen.

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Testpunktes.
Defaultwert: 100
(lin)

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Testpunktes.
Defaultwert: 100
(lin)

▷ **Index** (output_control) ..... integer(-array) ⇝ *integer*
Index der Regionen die den Testpunkt enthalten.

Komplexität

Sei F die Fläche einer Region und N die Anzahl der Regionen, dann beträgt die Laufzeitkomplexität im Mittel $O(ln(sqrt(F)) * N)$.

Ergebnis

`get_region_index` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`select_region_point`

Siehe auch

`get_mbutton`, `get_mposition`, `test_region_point`

Modul

Foundation

get_region_thickness (Region : : : Thickness, Histogramm)

Zugriff auf die Dicke einer Region entlang der Hauptachse.

`get_region_thickness` berechnet die Dicke der Regionen entlang der Hauptachse (siehe `elliptic_axis`) für jeden Punkt der Strecke. Die Dicke an einer Stelle der Hauptachse ist dabei als der Abstand der am weitesten auseinanderliegenden Schnittpunkte der Kontur mit dem Lot auf die Hauptachse im betreffenden Punkt definiert. Zusätzlich liefert `get_region_thickness` noch das `Histogramm` der Dicken der Region. Die Länge des Histogramms entspricht dabei der größten aufgetretenen Dicke in der betrachteten Region.

Achtung

Es darf nur eine Region übergeben werden. Besitzt die Region mehrere Zusammenhangskomponenten, wird nur die erste ausgewertet. Alle anderen werden ignoriert

Parameter

▷ **Region** (input_object) ..... region ⇝ *object*
Zu analysierende Region.

▷ **Thickness** (output_control) ..... integer-array ⇝ *integer*
Dicke der Region entlang ihrer Hauptachse.

▷ **Histogramm** (output_control) ..... integer-array ⇝ *integer*
Histogramm der Dicke der Region entlang ihrer Hauptachse.

Ergebnis

`get_region_thickness` liefert den Wert TRUE, falls genau eine Region übergeben wird. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

sobel_amp, threshold, connection, select_shape, select_obj

Siehe auch

copy_obj, elliptic_axis

Modul

Foundation

hamming_distance (Regions1, Regions2 : : : Distance, Similarity)

Ermitteln des Hamming-Abstands zweier Regionen.

hamming_distance liefert den Hamming-Abstand zweier Regionen, d.h. die Zahl der Punkte der Regionen, die sich unterscheiden (Distance), also die Zahl der Punkte, die in der einen Region enthalten sind, in der anderen jedoch nicht:

$$\texttt{Distance} = |\texttt{Regions1} \cap \overline{\texttt{Regions2}}| + |\texttt{Regions2} \cap \overline{\texttt{Regions1}}|$$

Der Parameter Similarity beschreibt die Ähnlichkeit der beiden Regionen, basierend auf dem Hamming-Abstand Distance:

$$\texttt{Similarity} = 1 - \frac{\texttt{Distance}}{|\texttt{Regions1}| + |\texttt{Regions2}|}$$

Sind beide Regionen leer, dann wird Similarity auf 0 gesetzt. Es werden immer die Regionen mit dem gleichen Index aus den beiden Eingabeparametern miteinander verglichen.

Achtung

In beiden Eingabeparametern muss die gleiche Anzahl von Regionen übergeben werden.

Parameter

- ▷ **Regions1** (input_object) . . . . . region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **Regions2** (input_object) . . . . . region(-array) ⇝ *object*
  Vergleichsregionen.
- ▷ **Distance** (output_control) . . . . . integer(-array) ⇝ *integer*
  Hamming-Abstand zweier Regionen.
  Zusicherung: Distance >= 0
- ▷ **Similarity** (output_control) . . . . . real(-array) ⇝ *real*
  Ähnlichkeit zweier Regionen.
  Zusicherung: 0 <= Similarity && Similarity <= 1

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

hamming_distance liefert den Wert TRUE, falls die Anzahl der Objekte in beiden Parametern gleich ist und nicht 0 beträgt. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

intersection, complement, area_center

Siehe auch

hamming_change_region

Modul

Foundation

hamming_distance_norm (Regions1, Regions2 : : Norm : Distance, Similarity)

Ermitteln des Hamming-Abstands zweier Regionen nach Normierung.

hamming_distance_norm liefert den Hamming-Abstand zweier Regionen, d.h. die Zahl der Punkte der Regionen, die sich unterscheiden (Distance). Vor der Berechnung des Abstandes wird eine Normierung der Region in Regions1 auf die Region in Regions2 durchgeführt. Das Ergebnis ist die Zahl der Punkte, die in der einen Region enthalten sind, in der anderen jedoch nicht:

$$\texttt{Distance} = |Norm(\texttt{Regions1}) \cap \overline{\texttt{Regions2}}| + |\texttt{Regions2} \cap \overline{Norm(\texttt{Regions1})}|$$

Der Parameter Similarity beschreibt die Ähnlichkeit der beiden Regionen, basierend auf dem Hamming-Abstand Distance:

$$\texttt{Similarity} = 1 - \frac{\texttt{Distance}}{|Norm(\texttt{Regions1})| + |\texttt{Regions2}|}$$

Folgende Arten der Normierung stehen zur Verfügung:

'center': Die Region wird so verschoben, dass beide den gleichen Schwerpunkt haben.

Sind beide Regionen leer, dann wird Similarity auf 0 gesetzt. Es werden immer die Regionen mit dem gleichen Index aus den beiden Eingabeparametern miteinander verglichen.

Achtung

In beiden Eingabeparametern muss die gleiche Anzahl von Regionen übergeben werden.

Parameter

- ▷ **Regions1** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **Regions2** (input_object) ..... region(-array) ⇝ *object*
  Vergleichsregionen
- ▷ **Norm** (input_control) ..... string(-array) ⇝ *string*
  Art der Normierung.
  Defaultwert: 'center'
  Werteliste: Norm ∈ {'center'}
- ▷ **Distance** (output_control) ..... integer(-array) ⇝ *integer*
  Hamming-Abstand zweier Regionen.
  Zusicherung: Distance >= 0
- ▷ **Similarity** (output_control) ..... real(-array) ⇝ *real*
  Ähnlichkeit zweier Regionen.
  Zusicherung: 0 <= Similarity && Similarity <= 1

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

hamming_distance_norm liefert den Wert TRUE, falls die Anzahl der Objekte in beiden Parametern gleich ist und nicht 0 beträgt. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`intersection`, `complement`, `area_center`

Siehe auch

`hamming_change_region`

Modul

Foundation

height_width_ratio (Regions : : : Height, Width, Ratio)

Ermitteln der Breite, Höhe und des Seitenverhältnisses des umschließenden Rechtecks parallel zu den Koordinatenachsen.

`height_width_ratio` berechnet das umschließende Rechteck aller Eingaberegionen (parallel zu den Koordinatenachsen). Das umschließende Rechteck wird durch die Koordinaten der Eckpunkte (`Row1`,`Column1`,`Row2`,`Column2`) beschrieben (siehe `smallest_rectangle1`). Basierend auf diesen Werten werden die Breite, Höhe und das Seitenverhältnis wie folgt berechnet:

$$\begin{aligned} \texttt{Width} &= \texttt{Column2} - \texttt{Column1} + 1 \\ \texttt{Height} &= \texttt{Row2} - \texttt{Row1} + 1 \\ \texttt{Ratio} &= \texttt{Height}/\texttt{Width} \end{aligned}$$

Wird mehr als eine Region in `Regions` übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Regionen in `Regions` abgespeichert. Bei leeren Regionen haben alle Parameter den Wert 0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Zu untersuchende Regionen.

▷ **Height** (output_control) ..... extent.y(-array) ⇝ *integer*
Höhe des umschließenden Rechtecks der Region.

▷ **Width** (output_control) ..... extent.x(-array) ⇝ *integer*
Breite des umschließenden Rechtecks der Region.

▷ **Ratio** (output_control) ..... real(-array) ⇝ *real*
Seitenverhältnis des umschließenden Rechtecks der Region.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

`height_width_ratio` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Alternativen

`smallest_rectangle1`, `smallest_rectangle2`, `region_features`

Siehe auch

`select_shape`, `smallest_circle`, `elliptic_axis`, `area_center`

Modul

Foundation

inner_circle (Regions : : : Row, Column, Radius)

Ermitteln des größten Inkreises einer Region.

`inner_circle` bestimmt den größten Inkreis einer Region. Das ist die größte diskrete Kreisregion, die vollständig in die Eingangsregion passt. Für diesen Kreis werden der Mittelpunkt (`Row`, `Column`) und der Radius (`Radius`) bestimmt. Ist die Position des Kreises nicht eindeutig definiert, wird die "erste mögliche"Position (das ist soweit links oben, wie möglich) zurückgeliefert.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich im Durchmesser ihrer Inkreise unterscheiden.

Die Ausgabe von `inner_circle` ist so gewählt, dass sie als Eingabe für die HALCON-Operatoren `disp_circle`, `gen_circle` und `gen_ellipse_contour_xld` verwendet werden kann.

Werden mehrere Regionen in `Regions` übergeben, werden die Ausgabeparameter als Tupel zurückgegeben. Ist die Eingaberegion leer, haben alle Parameter den Wert *0.0*, soweit mit `set_system` kein anderes Verhalten eingestellt wurde.

Achtung

Wenn mehrere Inkreise bei einer Region vorhanden sind, wird nur die Lösung, die am weitesten links oben ist, zurückgegeben.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Zu untersuchende Regionen.

▷ **Row** (output_control) ..... circle.center.y(-array) ⇝ *real*
Zeilenindex des Mittelpunktes.

▷ **Column** (output_control) ..... circle.center.x(-array) ⇝ *real*
Spaltenindex des Mittelpunktes.

▷ **Radius** (output_control) ..... circle.radius(-array) ⇝ *real*
Radius des Inkreises.
Zusicherung: Radius >= 0

Beispiel

```
read_image(Image,'fabrik')
regiongrowing(Image,Seg,5,5,6,100)
select_shape(Seg,H,'area','and',100,2000)
inner_circle(H,Row,Column,Radius)
gen_circle(Circles,Row,Column,Radius)
dev_set_draw('margin')
dev_display(Circles)
```

Komplexität

Sei F die Fläche der Region und R der Radius des Inkreises, dann beträgt die Laufzeitkomplexität $O(sqrt(F)*R)$.

Ergebnis

`inner_circle` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Nachfolger

`gen_circle`, `disp_circle`

Alternativen

`erosion_circle`, `inner_rectangle1`

Siehe auch

`set_shape`, `select_shape`, `smallest_circle`

Modul

Foundation

inner_rectangle1 (Regions : : : Row1, Column1, Row2, Column2)

Ermitteln des größten inneren Rechtecks einer Region.

`inner_rectangle1` bestimmt das größte achsenparallele Rechteck, das in eine Region passt. Das Rechteck wird durch die Koordinaten der Eckpunkte (`Row1`, `Column1`, `Row2`, `Column2`) beschrieben.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in der Breite und Höhe ihres größten inneren Rechtecks unterscheiden.

Wird mehr als eine Region in `Regions` übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index der Region in der Eingabe entspricht. Bei leerer Region haben alle Parameter den Wert 0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
Zu untersuchende Region.

▷ **Row1** (output_control) . . . . . rectangle.origin.y(-array) ⇝ *integer*
Zeilenkoordinate der linken oberen Ecke.

▷ **Column1** (output_control) . . . . . rectangle.origin.x(-array) ⇝ *integer*
Spaltenkoordinate der linken oberen Ecke.

▷ **Row2** (output_control) . . . . . rectangle.corner.y(-array) ⇝ *integer*
Zeilenkoordinate der rechten unteren Ecke.

▷ **Column2** (output_control) . . . . . rectangle.corner.x(-array) ⇝ *integer*
Spaltenkoordinate der rechten unteren Ecke.

Ergebnis

inner_rectangle1 liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Nachfolger

disp_rectangle1, gen_rectangle1

Alternativen

inner_circle

Siehe auch

smallest_rectangle1, select_shape

Modul

Foundation

moments_region_2nd (Regions : : : M11, M20, M02, Ia, Ib)

Ermitteln der geometrischen Momente von Regionen.

moments_region_2nd berechnet die geometrischen Momente M11, M20 und M02 der Eingaberegionen in Regions. Weiterhin werden die Hauptachsen Ia und Ib berechnet.

Die Kovarianzmatrix ist definiert durch:

$$\begin{bmatrix} M20 & M11 \\ M11 & M02 \end{bmatrix}$$

Das zeilenabhängige Moment 2. Ordnung wird in M20 und das spaltenabhängige Moment 2. Ordnung wird in M02 zurückgegeben. Das Moment M11 repräsentiert die Kovarianz zwischen den Zeilen- und Spaltenkoordinaten der Punkte der Region.

Berechnung: Es seien r_0 und c_0 die Koordinaten des Schwerpunktes einer Region R. Dann sind die Momente M_{ij} definiert durch:

$$M_{ij} = \sum_{(r,c) \in R} (r_0 - r)^i (c_0 - c)^j$$

wobei r und c alle Punkte der Region R durchlaufen.
Zusätzlich sind die Längen der 1. und 2. Hauptachse definiert durch:

$$Ia = h + \sqrt{h^2 - M20 * M02 + M11^2}$$

$$Ib = h - \sqrt{h^2 - M20 * M02 + M11^2}$$

wobei

$$h = \frac{M20 + M02}{2}$$

Die Gleichung zur Berechnung der 1. und 2. Hauptachse kann von der Definition der Momente abgeleitet werden durch Diagonalisieren der Kovarianzmatrix und Umformen der daraus resultierenden Formel.

Wird mehr als eine Region übergeben, werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region wird für alle Parameter der Wert 0.0 zurückgegeben, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegionen.
- ▷ **M11** (output_control) ..... real(-array) ⇝ *real*
  Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen.
- ▷ **M20** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung (zeilenabhängig).
- ▷ **M02** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung (spaltenabhängig).
- ▷ **Ia** (output_control) ..... real(-array) ⇝ *real*
  Länge der 1. Hauptachse der Eingaberegion.
- ▷ **Ib** (output_control) ..... real(-array) ⇝ *real*
  Länge der 2. Hauptachse der Eingaberegion.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

moments_region_2nd liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

moments_region_2nd_invar

Siehe auch

elliptic_axis

Literatur

R. Haralick, L. Shapiro „Computer and Robot Vision“ Addison-Wesley, 1992, pp. 73-75

Modul

Foundation

moments_region_2nd_invar (Regions : : : M11, M20, M02)

Ermitteln der geometrischen Momente von Regionen.

moments_region_2nd_invar berechnet die normierten Momente (M20, M02) und das Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen (M11).

Berechnung: Es seien Z_0 und S_0 die Koordinaten des Schwerpunktes einer Region R mit Fläche F. Dann sind die Momente M_{ij} definiert durch:

$$M_{ij} = \frac{1}{F^2} \sum_{(Z,S) \in R} (Z_0 - Z)^i (S_0 - S)^j$$

,
wobei Z und S alle Punkte der Region R durchlaufen.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **`M11`** (output_control) ..... real(-array) ⇝ *real*
  Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen.
- ▷ **`M20`** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung (zeilenabhängig).
- ▷ **`M02`** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung (spaltenabhängig).

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

`moments_region_2nd_invar` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`moments_region_2nd`

Siehe auch

`elliptic_axis`

Modul

Foundation

`moments_region_2nd_rel_invar` (Regions : : : PHI1, PHI2)

Ermitteln der geometrischen Momente von Regionen.

`moments_region_2nd_rel_invar` berechnet die relativen normierten Momente (`PHI1`, `PHI2`).

Berechnung: Die Momente PHI_1 und PHI_2 sind definiert durch:

$$PHI_1 = M_{20} + M_{02}$$

$$PHI_2 = (M_{20} - M_{02})^2 + 4M_{11}^2$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **PHI1** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **PHI2** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.

Ergebnis

moments_region_2nd_rel_invar liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

moments_region_2nd

Siehe auch

elliptic_axis

Modul

Foundation

moments_region_3rd (Regions : : : M21, M12, M03, M30)

Ermitteln der geometrischen Momente von Regionen.

moments_region_3rd berechnet die unveränderliche zentrale Momente (M21, M12, M03, M30) mit Ordnung $(p+q)$.

Berechnung: Es seien x und y die Koordinaten des Schwerpunktes einer Region R mit Fläche Z. Dann sind die Momente M_{pq} definiert durch:

$$M_{pq} = \sum_{i=1} MZ(x_i, y_i)(x_i - x)^p (y_i - y)^q$$

,
wobei $x = \frac{m_{10}}{m_{00}}$ and $y = \frac{m_{01}}{m_{00}}$.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **M21** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (zeilenabhängig).
- ▷ **M12** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (spaltenabhängig).
- ▷ **M03** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (spaltenabhängig).
- ▷ **M30** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (zeilenabhängig).

Komplexität

Sei Z die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(Z))$.

Ergebnis

`moments_region_3rd` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`moments_region_2nd`

Siehe auch

`elliptic_axis`

Modul

Foundation

moments_region_3rd_invar (Regions : : : M21, M12, M03, M30)

Ermitteln der geometrischen Momente von Regionen.

`moments_region_3rd_invar` berechnet die invariante Momente (`M21`, `M12`, `M03`, `M30`).

Berechnung: Die invarianten Momente M_{ij} sind definiert durch:

$$M_{pq} = \frac{\mu_{pq}}{\mu^3}$$

,
wobei $p + q >= 2$ and $\mu = \mu_{00} = m_{00}$
.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **M21** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (zeilenabhängig).
- ▷ **M12** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (spaltenabhängig).
- ▷ **M03** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (spaltenabhängig).
- ▷ **M30** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (zeilenabhängig).

Komplexität

Sei Z die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(Z))$.

Ergebnis

`moments_region_3rd_invar` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`moments_region_2nd`

Siehe auch

`elliptic_axis`

Modul

Foundation

moments_region_central (Regions : : : I1, I2, I3, I4)

Ermitteln der geometrischen Momente von Regionen.

`moments_region_central` berechnet die zentrale Momente (`I1`, `I2`, `I3`, `I4`).

Berechnung: Die invarianten Momente I_i sind definiert durch:

$$
\begin{aligned}
I_1 &= \mu_{20}\mu_{02} - \mu_{11}^2 \\
I_2 &= (\mu_{30}\mu_{03} - \mu_{21}\mu_{12})^2 - 4(\mu_{30}\mu_{12} - \mu_{21}^2)(\mu_{21}\mu_{03} - \mu_{12}^2) \\
I_3 &= \mu_{20}(\mu_{21}\mu_{03} - \mu_{12}^2) - \mu_{11}(\mu_{30}\mu_{03} - \mu_{21}\mu_{12}) + \mu_{02}(\mu_{30}\mu_{12} - \mu_{21}^2) \\
I_4 &= \mu_{30}^2\mu_{02}^3 - 6\mu_{30}\mu_{21}\mu_{11}\mu_{02}^2 + 6\mu_{30}\mu_{12}\mu_{02}(2\mu_{11}^2 - \mu_{20}\mu_{02}) \\
&\quad +\mu_{30}\mu_{03}(6\mu_{20}\mu_{11}\mu_{02} - 8\mu_{11}^3) + 9\mu_{21}^2\mu_{20}\mu_{02}^2 - 18\mu_{21}\mu_{12}\mu_{20}\mu_{11}\mu_{02} \\
&\quad +6\mu_{21}\mu_{03}\mu_{20}(2\mu_{11}^2 - \mu_{20}\mu_{02}) + 9\mu_{12}^2\mu_{20}^2\mu_{02} - 6\mu_{12}\mu_{03}\mu_{11}\mu_{20}^2 + \mu_{03}^2\mu_{20}^3
\end{aligned}
$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **I1** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **I2** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **I3** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **I4** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung.

Komplexität

Sei Z die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(Z))$.

Ergebnis

moments_region_central liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

moments_region_2nd

Siehe auch

elliptic_axis

Modul

Foundation

moments_region_central_invar (Regions : : : PSI1, PSI2, PSI3, PSI4)

Ermitteln der geometrischen Momente von Regionen.

moments_region_central_invar berechnet die Momente (PSI1, PSI2, PSI3, PSI4), die bei Bewegung und lineare Transformationen unverändert bleiben.

Berechnung: Die Momente ψ_i sind definiert durch:

$$\begin{aligned}\psi_1 &= \frac{I_1}{\mu^4}\\ \psi_2 &= \frac{I_2}{\mu^1 0}\\ \psi_3 &= \frac{I_3}{\mu^7}\\ \psi_4 &= \frac{I_4}{\mu^1 1}\end{aligned}$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Zu untersuchende Regionen.

▷ **PSI1** (output_control) ..... real(-array) ⇝ *real*
Moment 2. Ordnung.

▷ **PSI2** (output_control) ..... real(-array) ⇝ *real*
Moment 2. Ordnung.

▷ **PSI3** (output_control) ..... real(-array) ⇝ *real*
Moment 2. Ordnung.

▷ **PSI4** (output_control) ..... real(-array) ⇝ *real*
Moment 2. Ordnung.

Komplexität

Sei Z die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(Z))$.

Ergebnis

moments_region_central_invar liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

moments_region_2nd

Siehe auch

elliptic_axis

Modul

Foundation

orientation_region (Regions : : : Phi)

Ermitteln Orientierung einer Region.

orientation_region berechnet die Orientierung (Phi) der Region. Das Verfahren basiert auf dem Winkel der mit elliptic_axis bestimmt wird. Es wird jedoch zusätzlich der Punkt auf dem Rand mit maximalem Abstand zum Schwerpunkt der Region bestimmt. Ist dessen Spaltenkoordinate im gedrehten System kleiner als die Spaltenkoordinate des Schwerpunkts so wird π zu dem Winkel hinzugezählt.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihrer Orientierung unterscheiden.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system('no_object_result',<Result>)).

Parameter

▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
Zu untersuchende Region(en).

▷ **Phi** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Orientierung der Region (Bogenmaß).
Zusicherung: - pi <= Phi && Phi < pi

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

`orientation_region` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Nachfolger

`disp_arrow`

Alternativen

`elliptic_axis`, `smallest_rectangle2`

Siehe auch

`moments_region_2nd`, `line_orientation`

Modul

Foundation

rectangularity (Regions : : : Rectangularity)

Ermitteln des Formfaktors für die Rechteckigkeit einer Region.

`rectangularity` berechnet die Rechteckigkeit der Eingaberegionen.

Zur Bestimmung der Rechteckigkeit wird zunächst ein Rechteck ermittelt, das die gleichen ersten und zweiten Momente besitzt wie die Eingaberegion. Die Berechnung des Rechteckigkeitsmaßes basiert schließlich auf der Fläche der Differenz zwischen dem ermittelten Rechteck und der Eingaberegion normiert auf die Fläche des Rechtecks.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihrer Rechteckigkeit unterscheiden.

Für Rechtecke liefert `rectangularity` den Wert 1 zurück. Je weniger die Eingaberegion einem Rechteck gleicht, desto geringer ist `Rectangularity`.

Bei einer leeren Region liefert `rectangularity` (soweit kein anderes Verhalten festgelegt wurde (siehe `set_system`)) den Wert 0. Wird mehr als eine Region übergeben, dann werden die Zahlenwerte für die Rechteckigkeit in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Achtung

Bei Eingaberegionen, deren Orientierung über Momente zweiter Ordnung nicht definiert ist (wie es z.B. beim Quadrat der Fall ist), wird für `Rectangularity` je nach Orientierung der Eingaberegion ein bis zu 10% zu niedriger Wert zurückgeliefert.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Zu untersuchende Region(en).

▷ **Rectangularity** (output_control) ..... real(-array) ⇝ *real*
Rechteckigkeit der Eingaberegion(en).
Zusicherung: 0 <= Rectangularity && Rectangularity <= 1.0

Ergebnis

`rectangularity` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`circularity`, `compactness`, `convexity`, `eccentricity`

Siehe auch

`contlength`, `area_center`, `select_shape`

Literatur

P. L. Rosin: „Measuring rectangularity“; Machine Vision and Applications; vol. 11; pp. 191-196; Springer-Verlag, 1999.

Modul

Foundation

region_features (Regions : : Features : Value)

Berechnen von verschiedenen Formmerkmalen für Regionen.

`region_features` berechnet für jede Eingaberegion aus `Regions` die angegebenen Merkmale (`Features`).

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie Illustrationen zu den unterschiedlichen Features.

Mögliche Werte für `Features`:

'area': Fläche des Objektes

'row': Zeilenindex der Schwerpunkts

'column': Spaltenindex der Schwerpunkts

'width': Breite der Region (parallel zu den Koordinatenachsen)

'height': Höhe der Region (parallel zu den Koordinatenachsen)

'ratio': Verhältnis der Höhe zur Breite der Region (parallel zu den Koordinatenachsen)

'row1': Zeilenindex der linken oberen Ecke

'column1': Spaltenindex der linken oberen Ecke

'row2': Zeilenindex der rechten unteren Ecke

'column2': Spaltenindex der rechten unteren Ecke

'circularity': Kreisförmigkeit (vgl. `circularity`)

'compactness': Kompaktheit (vgl. compactness)

'contlength': Gesamtlänge der Kontur (vgl. contlength)

'convexity': Konvexität (vgl. convexity)

'rectangularity': Rechteckigkeit (vgl. rectangularity)

'ra': Hauptradius der äquivalenten Ellipse (vgl. elliptic_axis)

'rb': Nebenradius der äquivalenten Ellipse (vgl. elliptic_axis)

'phi': Orientierung der äquivalenten Ellipse (vgl. elliptic_axis)

'anisometry' Anisometrie (vgl. eccentricity)

'bulkiness' Bulkiness (vgl. Operator eccentricity)

'struct_factor' Struktur Faktor (vgl. Operator eccentricity)

'outer_radius': Radius des kleinsten umschließenden Kreises (vgl. smallest_circle)

'inner_radius': Radius des größten Inkreises (vgl. inner_circle)

'inner_width': Breite des größten enthaltenen achsenparallelen Rechtecks (vgl. inner_rectangle1)

'inner_height': Höhe des größten enthaltenen achsenparallelen Rechtecks (vgl. inner_rectangle1)

'dist_mean': Mittlerer Abstand zwischen dem Rand der Region und ihrem Schwerpunkt (vgl. Operator roundness)

'dist_deviation': Standardabweichung des Abstands vom Regionenrand zum Schwerpunkt (vgl. Operator roundness)

'roundness': Rundheit (vgl. Operator roundness)

'num_sides': Anzahl Polygonseiten (vgl. Operator roundness)

'connect_num': Anzahl der Zusammenhangskomponenten (vgl. Operator connect_and_holes)

'holes_num': Anzahl der Löcher (vgl. Operator connect_and_holes)

'area_holes:' Fläche der Löcher des Objektes (vgl. Operator area_holes)

'max_diameter': Maximale Ausdehnung der Region (vgl. Operator diameter_region)

'orientation': Orientierung der Region (vgl. Operator orientation_region)

'euler_number': Eulerzahl (vgl. Operator euler_number)

'rect2_phi': Orientierung des kleinsten umschließenden Rechtecks (vgl. Operator smallest_rectangle2)

'rect2_len1': Halbe Länge des kleinsten umschließenden Rechtecks (vgl. Operator smallest_rectangle2)

'rect2_len2': Halbe Breite des kleinsten umschließenden Rechtecks (vgl. Operator smallest_rectangle2)

'moments_m11': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m20': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m02': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_ia': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_ib': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m11_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_m20_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_m02_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_phi1': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_rel_invar)

'moments_phi2': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_rel_invar)

'moments_m21': Geometrische Regionenmomente (vgl. Operator moments_region_3rd)

'moments_m12': Geometrische Regionenmomente (vgl. Operator moments_region_3rd)

'moments_m03': Geometrische Regionenmomente (vgl. Operator moments_region_3rd)

'moments_m30': Geometrische Regionenmomente (vgl. Operator moments_region_3rd)

'moments_m21_invar': Geometrische Regionenmomente (vgl. Operator moments_region_3rd_invar)

'moments_m12_invar': Geometrische Regionenmomente (vgl. Operator moments_region_3rd_invar)

'moments_m03_invar': Geometrische Regionenmomente (vgl. Operator moments_region_3rd_invar)

'moments_m30_invar': Geometrische Regionenmomente (vgl. Operator moments_region_3rd_invar)

'moments_i1': Geometrische Regionenmomente (vgl. Operator moments_region_central)

'moments_i2': Geometrische Regionenmomente (vgl. Operator moments_region_central)

'moments_i3': Geometrische Regionenmomente (vgl. Operator moments_region_central)

'moments_i4': Geometrische Regionenmomente (vgl. Operator moments_region_central)

'moments_psi1': Geometrische Regionenmomente (vgl. Operator moments_region_central_invar)

'moments_psi2': Geometrische Regionenmomente (vgl. Operator moments_region_central_invar)

'moments_psi3': Geometrische Regionenmomente (vgl. Operator moments_region_central_invar)

'moments_psi4': Geometrische Regionenmomente (vgl. Operator moments_region_central_invar)

Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie eingegeben werden.

Parameter

▷ **Regions** (input_object) ..... region-array ⇝ *object*
Regionen, die untersucht werden sollen.

▷ **Features** (input_control) ..... string(-array) ⇝ *string*
Zu berechnende Formmerkmale.
Defaultwert: 'area'
Werteliste: Features ∈ {'area', 'row', 'column', 'width', 'height', 'ratio', 'row1', 'column1', 'row2', 'column2', 'circularity', 'compactness', 'contlength', 'convexity', 'rectangularity', 'ra', 'rb', 'phi', 'anisometry', 'bulkiness', 'struct_factor', 'outer_radius', 'inner_radius', 'inner_width', 'inner_height', 'max_diameter', 'dist_mean', 'dist_deviation', 'roundness', 'num_sides', 'orientation', 'connect_num', 'holes_num', 'area_holes', 'euler_number', 'rect2_phi', 'rect2_len1', 'rect2_len2', 'moments_m11', 'moments_m20', 'moments_m02', 'moments_ia', 'moments_ib', 'moments_m11_invar', 'moments_m20_invar', 'moments_m02_invar', 'moments_phi1', 'moments_phi2', 'moments_m21', 'moments_m12', 'moments_m03', 'moments_m30', 'moments_m21_invar', 'moments_m12_invar', 'moments_m03_invar', 'moments_m30_invar', 'moments_i1', 'moments_i2', 'moments_i3', 'moments_i4', 'moments_psi1', 'moments_psi2', 'moments_psi3', 'moments_psi4'}

▷ **Value** (output_control) ..... real(-array) ⇝ *real*
Der errechnete Wert für das Feature.

Beispiel

```
read_image(Image,'monkey')
threshold(Image,S1,160,255)
connection(S1,S2)
region_features(S2,['area','anisometry'],Value)
```

Ergebnis

region_features liefert den Wert TRUE, falls die Eingabe nicht leer ist. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection, runlength_features

Nachfolger

select_shape, select_gray, shape_trans, reduce_domain, count_obj

Siehe auch

area_center, circularity, compactness, contlength, convexity, rectangularity, elliptic_axis, eccentricity, inner_circle, smallest_circle, smallest_rectangle1, smallest_rectangle2, inner_rectangle1, roundness, connect_and_holes, area_holes,

diameter_region, orientation_region, moments_region_2nd, moments_region_2nd_invar, moments_region_2nd_rel_invar, moments_region_3rd, moments_region_3rd_invar, moments_region_central, moments_region_central_invar, select_obj, select_shape

Modul

Foundation

roundness (Regions : : : Distance, Sigma, Roundness, Sides)

Ermitteln der Formfaktoren aus der Kontur.

roundness untersucht den Abstand der Kontur vom Schwerpunkt der Fläche. Im einzelnen wird der mittlere Abstand (Distance), die Abweichung vom mittleren Abstand (Sigma) und zwei daraus abgeleitete Formmerkmale bestimmt. Roundness ist das Verhältnis von Mittelwert zu Standardabweichung und Sides gibt die Anzahl der Polygonstücke an, falls es sich um ein regelmäßiges Polygon handelt.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihrem mittleren Abstand, der Abweichung vom mittleren Abstand, der Roundness und den Sides unterscheiden.

Die Kontur zur Berechnung der Merkmale wird in Abhängigkeit von der globalen Nachbarschaft bestimmt (siehe set_system).

Berechnung:

Sei p der Flächenschwerpunkt, p_i die Punkte und F die Fläche der Kontur.

$$\texttt{Distance} = \frac{1}{F} \sum ||p - p_i||$$

$$\texttt{Sigma}^2 = \frac{1}{F} \sum (||p - p_i|| - \texttt{Distance})^2$$

$$\texttt{Roundness} = 1 - \frac{\texttt{Sigma}}{\texttt{Distance}}$$

$$\texttt{Sides} = 1.4111 \left(\frac{\texttt{Distance}}{\texttt{Sigma}} \right)^{0.4724}$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Region(en).
- ▷ **Distance** (output_control) ..... real(-array) ⇝ *real*
  Mittlerer Abstand vom Schwerpunkt.
  Zusicherung: Distance >= 0.0
- ▷ **Sigma** (output_control) ..... real(-array) ⇝ *real*
  Standardabweichung von Distance.
  Zusicherung: Sigma >= 0.0
- ▷ **Roundness** (output_control) ..... real(-array) ⇝ *real*
  Formfaktor für Rundheit.
  Zusicherung: Roundness <= 1.0
- ▷ **Sides** (output_control) ..... real(-array) ⇝ *real*
  Anzahl der Polygonseiten.
  Zusicherung: Sides >= 0

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

`roundness` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`compactness`

Siehe auch

`contlength`

Literatur

R. Haralick, L. Shapiro „Computer and Robot Vision“ Addison-Wesley, 1992, pp. 61

Modul

Foundation

`runlength_distribution ( Region : : : Foreground, Background )`

Ermitteln der Verteilung der Sehnenlängen einer Region.

`runlength_distribution` berechnet die Verteilung der Sehnenlängen einer Region von Vorder- und Hintergrund. Es wird gezählt wie häufig eine bestimmte Länge auftritt. Sehnen der Länge unendlich werden nicht gezählt. Der Hintergrund sind also alle Hohlflächen der Region. Es werden so viele Werte übergeben, wie die maximale Länge von Vorder- bzw. Hintergrund vorgibt. Die Länge der beiden Tupel ist i.A. unterschiedlich. Der erste Eintrag der Tupel ist immer 0 (keine Sehnen der Länge 0). Gibt es keine Zwischenräume wird bei `Background` das leere Tupel übergeben. Analog wird bei einer leeren Region bei `Foreground` das leere Tupel übergeben.

Parameter

▷ **`Region`** (input_object) ..... region ⇝ *object*
Zu untersuchende Region.

▷ **`Foreground`** (output_control) ..... integer-array ⇝ *integer*
Längenverteilung der Region (Vordergrund).

▷ **`Background`** (output_control) ..... integer-array ⇝ *integer*
Längenverteilung des Hintergrundes.

Komplexität

Sei n die Anzahl der Sehnen der Region, dann beträgt die Laufzeitkomplexität $O(n)$.

Ergebnis

`runlength_distribution` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Wird mehr als eine Region übergeben, wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, select_obj

Alternativen

runlength_features

Siehe auch

runlength_features

Modul

Foundation

runlength_features (Regions : : : NumRuns, KFactor, LFactor, MeanLength, Bytes)

Ermitteln von charakteristischen Werten zur Lauflängenkodierung von Regionen.

runlength_features berechnet zu jeder Eingaberegion aus Regions die Anzahl der Sehnen, die bei der Speicherung dieser Region mit Hilfe der Lauflängenkodierung benötigt werden. Weiterhin wird der sogenannte „K-Faktor“ bestimmt, der angibt, um wie viel die Anzahl der Sehnen vom Ideal des Quadrates abweichen, bei dem dieser Wert gleich 1.0 ist.

Der K-Faktor (KFactor) berechnet sich nach der Formel:

$$\texttt{KFactor} = \frac{\texttt{NumRuns}}{\sqrt{Area}}$$

,

wobei $Area$ die Fläche der Region angibt. Es ist zu beachten, dass der K-Faktor kleiner als 1.0 werden kann (bei langgestreckten horizontalen Regionen).

Der L-Faktor (LFactor) gibt die mittlere Anzahl von Sehnen für jeden in der Region vorkommenden Zeilenindex aus.

MeanLength gibt die mittlere Länge der Sehnen an. Der Parameter Bytes gibt an, wie viele Bytes zur Kodierung der Region mit Lauflängen benötigt werden.

Achtung

Alle mit runlength_features berechneten Merkmale sind nicht rotationsinvariant, da die Lauflängenkodierung richtungsabhängig ist. runlength_features dient nicht zur Berechnung von Formmerkmalen, sondern zur Kontrolle und Analyse der Leistungsfähigkeit der Lauflängenkodierung.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **NumRuns** (output_control) ..... integer(-array) ⇝ *integer*
  Anzahl der Sehnen.
  Zusicherung: 0 <= NumRuns
- ▷ **KFactor** (output_control) ..... real(-array) ⇝ *real*
  Speicherfaktor gegenüber einem Quadrat.
  Zusicherung: 0 <= KFactor
- ▷ **LFactor** (output_control) ..... real(-array) ⇝ *real*
  Mittlere Anzahl von Sehnen pro Zeile.
  Zusicherung: 0 <= LFactor
- ▷ **MeanLength** (output_control) ..... real(-array) ⇝ *real*
  Mittlere Länge der Sehnen.
  Zusicherung: 0 <= MeanLength
- ▷ **Bytes** (output_control) ..... integer(-array) ⇝ *integer*
  Anzahl an Bytes, die zur Kodierung der Region benötigt werden.
  Zusicherung: 0 <= Bytes

Komplexität

Die Laufzeitkomplexität beträgt $O(1)$.

Ergebnis

`runlength_features` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Siehe auch

`runlength_distribution`

Modul

Foundation

`select_region_point ( Regions : DestRegions : Row, Column : )`

Auswählen aller Regionen, die einen übergebenen Punkt enthalten.

`select_region_point` selektiert alle Regionen aus `Regions`, die den Testpunkt (`Row`,`Column`) enthalten, d.h.:

$$|\texttt{Regions}[n] \cap \{(\texttt{Row}, \texttt{Column})\}| = 1$$

Achtung

Es kann vorkommen, dass mehr als eine Region den Punkt enthält, falls sich die Regionen überlappen. In diesem Fall werden all diese Regionen ausgegeben. Falls keine Region den angegebenen Punkt enthält, wird das leere Tupel (=keine Region) ausgegeben.

Parameter

▷ **`Regions`** (input_object) ..... region-array ⇝ *object*
Zu untersuchende Regionen.

▷ **`DestRegions`** (output_object) ..... region-array ⇝ *object*
Alle Regionen, die den Testpunkt enthalten.

▷ **`Row`** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Testpunktes.
Defaultwert: 100

▷ **`Column`** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Testpunktes.
Defaultwert: 100

Beispiel

```
read_image(Image,'fabrik')
regiongrowing(Image,Seg,3,3,5,0)
dev_set_color('red')
dev_set_draw('margin')
Button := 1
while (Button == 1)
  get_mbutton(WindowHandle,Row,Column,Button)
  select_region_point(Seg,Single,Row,Column)
  dev_display(Image)
  dev_display(Single)
endwhile
```

Komplexität

Sei F die Fläche einer Region und N die Anzahl der Regionen, dann beträgt die Laufzeitkomplexität im Mittel $O(ln(sqrt(F)) * N)$.

Ergebnis

`select_region_point` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`test_region_point`

Siehe auch

`get_mbutton`, `get_mposition`

Modul

Foundation

select_region_spatial (Regions1, Regions2 : : Direction : RegionIndex1, RegionIndex2)

Ermitteln der Lagebeziehung von Regionen.

`select_region_spatial` wählt die Regionen aus `Regions2` aus, die der Nachbarschaftsbeziehung `Direction` genügen. Die zu untersuchenden Regionen sind in `Regions1` bzw. `Regions2` zu übergeben. `Regions1` kann dabei auf drei verschiedene Arten besetzt werden:

- `Regions1` ist leer:
  In diesem Fall werden alle Regionen in `Regions2` permutativ auf Nachbarschaft getestet.
- `Regions1` besteht aus einer Region:
  Die Regionen von `Regions1` werden mit allen Regionen in `Regions2` verglichen.
- `Regions1` besteht aus gleich vielen Regionen wie `Regions2`:
  Hier werden jeweils die Regionen an n-ter Position in `Regions1` und `Regions2` auf die Nachbarschaftsbeziehung untersucht.

Mögliche Werte für `Direction` sind:

'left': `Regions2` ist links von `Regions1`
'right': `Regions2` ist rechts von `Regions1`
'above': `Regions2` ist oberhalb von `Regions1`
'below': `Regions2` ist unterhalb von `Regions1`

`select_region_spatial` berechnet die Schwerpunkte der zu vergleichenden Regionen und entscheidet anhand des Winkels zwischen der Schwerpunktgeraden und der x-Achse, ob die Richtungsbeziehung erfüllt ist. Die Relation ist jeweils in dem Bereich von -45 Grad bis +45 Grad um die Koordinatenachsen erfüllt. Die Richtungsrelation ist also so zu verstehen, dass der Schwerpunkt der zweiten Region links (bzw. rechts, oben, unten) vom Schwerpunkt der ersten Region liegen muss. Die Indizes der Regionen, die die Richtungsbeziehung erfüllen, stehen an n-ter Position in `RegionIndex1` und `RegionIndex2`, d.h. die Region mit Index `RegionIndex2`[n] steht mit Region mit Index `RegionIndex1`[n] in der angegebenen Relation. Der Zugriff auf Regionen über den Index kann mit `copy_obj` erfolgen.

Parameter

▷ **Regions1** (input_object) ..... region(-array) ⇝ *object*
Ausgangsregionen

▷ **Regions2** (input_object) ..... region(-array) ⇝ *object*
Vergleichsregionen

▷ **Direction** (input_control) ..... string ⇝ *string*
Gewünschte Nachbarschaftsbeziehung.
Defaultwert: 'left'
Werteliste: `Direction` ∈ {'left', 'right', 'above', 'below'}

▷ **RegionIndex1** (output_control) ..... integer-array ⇝ *integer*
Indizes in den Eingabetupeln (`Regions1` bzw. `Regions2`).

▷ **RegionIndex2** (output_control) ..... integer-array ⇝ *integer*
Indizes in den Eingabetupeln (`Regions1` bzw. `Regions2`).

Ergebnis

`select_region_spatial` liefert den Wert TRUE, falls `Regions2` nicht leer ist. Das Verhalten bei leerem Parameter `Regions2` (keine Eingaberegionen vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `regiongrowing`, `connection`

Alternativen

`area_center`, `intersection`

Siehe auch

`spatial_relation`, `find_neighbors`, `copy_obj`, `obj_to_integer`

Modul

Foundation

select_shape (Regions : SelectedRegions : Features, Operation, Min, Max :)

Auswählen von Regionen mit Hilfe von Formmerkmalen.

`select_shape` wählt Regionen anhand ihrer Form aus. Für jede Eingaberegion aus `Regions` werden die angegebenen Merkmale (`Features`) berechnet. Wenn jedes (`Operation` = *'and'*) oder mindestens eines (`Operation` = *'or'*) der so berechneten Merkmale in den vorgegebenen Grenzen (`Min`,`Max`) liegt, wird die Region in die Ausgabe übernommen (dupliziert). Um diese Grenzen nach unten oder oben offen zu lassen, kann statt eines Wertes auch *'min'* beziehungsweise *'max'* für `Min` und `Max` gesetzt werden.

Bedingung: $Min_i \leq Feature_i(Object) \leq Max_i$

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie Illustrationen zu den unterschiedlichen Features.

Mögliche Werte für `Features`:

'area': Fläche des Objektes

'row': Zeilenindex der Schwerpunkts

'column': Spaltenindex der Schwerpunkts

'width': Breite der Region (parallel zu den Koordinatenachsen; siehe `height_width_ratio`)

'height': Höhe der Region (parallel zu den Koordinatenachsen; siehe height_width_ratio)

'ratio': Verhältnis der Höhe zur Breite der Region (parallel zu den Koordinatenachsen; siehe height_width_ratio)

'row1': Zeilenindex der linken oberen Ecke

'column1': Spaltenindex der linken oberen Ecke

'row2': Zeilenindex der rechten unteren Ecke

'column2': Spaltenindex der rechten unteren Ecke

'circularity': Kreisförmigkeit (vgl. circularity)

'compactness': Kompaktheit (vgl. compactness)

'contlength': Gesamtlänge der Kontur (vgl. contlength)

'convexity': Konvexität (vgl. convexity)

'rectangularity': Rechteckigkeit (vgl. rectangularity)

'ra': Hauptradius der äquivalenten Ellipse (vgl. elliptic_axis)

'rb': Nebenradius der äquivalenten Ellipse (vgl. elliptic_axis)

'phi': Orientierung der äquivalenten Ellipse (vgl. elliptic_axis)

'anisometry' Anisometrie (vgl. eccentricity)

'bulkiness' Bulkiness (vgl. Operator eccentricity)

'struct_factor' Struktur Faktor (vgl. Operator eccentricity)

'outer_radius': Radius des kleinsten umschließenden Kreises (vgl. smallest_circle)

'inner_radius': Radius des größten Inkreises (vgl. inner_circle)

'inner_width': Breite des größten enthaltenen achsenparallelen Rechtecks (vgl. inner_rectangle1)

'inner_height': Höhe des größten enthaltenen achsenparallelen Rechtecks (vgl. inner_rectangle1)

'dist_mean': Mittlerer Abstand zwischen dem Rand der Region und ihrem Schwerpunkt (vgl. Operator roundness)

'dist_deviation': Standardabweichung des Abstands vom Regionenrand zum Schwerpunkt (vgl. Operator roundness)

'roundness': Rundheit (vgl. Operator roundness)

'num_sides': Anzahl Polygonseiten (vgl. Operator roundness)

'connect_num': Anzahl der Zusammenhangskomponenten (vgl. Operator connect_and_holes)

'holes_num': Anzahl der Löcher (vgl. Operator connect_and_holes)

'area_holes:' Fläche der Löcher des Objektes (vgl. Operator area_holes)

'max_diameter': Maximale Ausdehnung der Region (vgl. Operator diameter_region)

'orientation': Orientierung der Region (vgl. Operator orientation_region)

'euler_number': Eulerzahl (vgl. Operator euler_number)

'rect2_phi': Orientierung des kleinsten umschließenden Rechtecks (vgl. Operator smallest_rectangle2)

'rect2_len1': Halbe Länge des kleinsten umschließenden Rechtecks (vgl. Operator smallest_rectangle2)

'rect2_len2': Halbe Breite des kleinsten umschließenden Rechtecks (vgl. Operator smallest_rectangle2)

'moments_m11': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m20': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m02': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_ia': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_ib': Geometrische Regionenmomente (vgl. Operator moments_region_2nd)

'moments_m11_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_m20_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_m02_invar': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_invar)

'moments_phi1': Geometrische Regionenmomente (vgl. Operator moments_region_2nd_rel_invar)

'moments_phi2': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd_rel_invar`)
'moments_m21': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd`)
'moments_m12': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd`)
'moments_m03': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd`)
'moments_m30': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd`)
'moments_m21_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd_invar`)
'moments_m12_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd_invar`)
'moments_m03_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd_invar`)
'moments_m30_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd_invar`)
'moments_i1': Geometrische Regionenmomente (vgl. Operator `moments_region_central`)
'moments_i2': Geometrische Regionenmomente (vgl. Operator `moments_region_central`)
'moments_i3': Geometrische Regionenmomente (vgl. Operator `moments_region_central`)
'moments_i4': Geometrische Regionenmomente (vgl. Operator `moments_region_central`)
'moments_psi1': Geometrische Regionenmomente (vgl. Operator `moments_region_central_invar`)
'moments_psi2': Geometrische Regionenmomente (vgl. Operator `moments_region_central_invar`)
'moments_psi3': Geometrische Regionenmomente (vgl. Operator `moments_region_central_invar`)
'moments_psi4': Geometrische Regionenmomente (vgl. Operator `moments_region_central_invar`)

Wird nur ein Merkmal (`Features`) verwendet, dann ist der Wert von `Operation` bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie eingegeben werden.

Parameter

▷ **`Regions`** (input_object) ..... region-array ⇝ *object*
Regionen, die untersucht werden sollen.

▷ **`SelectedRegions`** (output_object) ..... region-array ⇝ *object*
Regionen, die die Bedingung erfüllen.

▷ **`Features`** (input_control) ..... string(-array) ⇝ *string*
Zu testende Formmerkmale.
Defaultwert: 'area'
Werteliste: `Features` ∈ {'area', 'row', 'column', 'width', 'height', 'ratio', 'row1', 'column1', 'row2', 'column2', 'circularity', 'compactness', 'contlength', 'convexity', 'rectangularity', 'ra', 'rb', 'phi', 'anisometry', 'bulkiness', 'struct_factor', 'outer_radius', 'inner_radius', 'inner_width', 'inner_height', 'max_diameter', 'dist_mean', 'dist_deviation', 'roundness', 'num_sides', 'orientation', 'connect_num', 'holes_num', 'area_holes', 'euler_number', 'rect2_phi', 'rect2_len1', 'rect2_len2', 'moments_m11', 'moments_m20', 'moments_m02', 'moments_ia', 'moments_ib', 'moments_m11_invar', 'moments_m20_invar', 'moments_m02_invar', 'moments_phi1', 'moments_phi2', 'moments_m21', 'moments_m12', 'moments_m03', 'moments_m30', 'moments_m21_invar', 'moments_m12_invar', 'moments_m03_invar', 'moments_m30_invar', 'moments_i1', 'moments_i2', 'moments_i3', 'moments_i4', 'moments_psi1', 'moments_psi2', 'moments_psi3', 'moments_psi4'}

▷ **`Operation`** (input_control) ..... string ⇝ *string*
Verknüpfungsart der einzelnen Merkmale.
Defaultwert: 'and'
Werteliste: `Operation` ∈ {'and', 'or'}

▷ **`Min`** (input_control) ..... real(-array) ⇝ *real* / integer / string
Untere Grenzen der Merkmale oder 'min'.
Defaultwert: 150.0
Typischer Wertebereich: 0.0 ≤ `Min` ≤ 99999.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 1.0

▷ **`Max`** (input_control) ..... real(-array) ⇝ *real* / integer / string
Obere Grenzen der Merkmale oder 'max'.
Defaultwert: 99999.0
Typischer Wertebereich: 0.0 ≤ `Max` ≤ 99999.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 1.0
Restriktion: Max >= Min

Beispiel

```
* Where are the eyes of the Mandrill?
read_image(Image,'monkey')
threshold(Image,Region,128,255)
connection(Region,ConnectedRegions)
select_shape(ConnectedRegions,Eyes,['area','max_diameter'],\
             'and',[500,30.0],[1000,50.0])
dev_display(Eyes)
```

Ergebnis

`select_shape` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Nachfolger

`select_gray`, `shape_trans`, `reduce_domain`, `count_obj`

Alternativen

`select_shape_std`

Siehe auch

`area_center`, `circularity`, `compactness`, `contlength`, `convexity`, `rectangularity`, `elliptic_axis`, `eccentricity`, `inner_circle`, `smallest_circle`, `smallest_rectangle1`, `smallest_rectangle2`, `inner_rectangle1`, `roundness`, `connect_and_holes`, `area_holes`, `diameter_region`, `orientation_region`, `moments_region_2nd`, `moments_region_2nd_invar`, `moments_region_2nd_rel_invar`, `moments_region_3rd`, `moments_region_3rd_invar`, `moments_region_central`, `moments_region_central_invar`, `select_obj`

Modul

Foundation

`select_shape_proto` `( Regions, Pattern : SelectedRegions : Feature, Min, Max : )`

Auswählen von Regionen, die in einer bestimmten Beziehung zueinander stehen.

`select_shape_proto` setzt zwei Regionen zueinander in Beziehung. Es wird jeweils die i-te Region aus `Regions` mit der Vereinigung der Regionen aus `Pattern` verglichen. Die Grenzen (`Min` und `Max`) werden je nach Merkmal absolut oder in Prozent (0..100) angegeben. Mögliche Werte für `Feature` sind:

'distance_dilate' Der minimale Abstand in der Maximumsnorm vom Rand von `Pattern` zum Rand jeder Region aus `Regions` wird bestimmt (vgl. `distance_rr_min_dil`).

'distance_contour' Der minimale euklidsche Abstand vom Rand von `Pattern` zum Rand jeder Region aus `Regions` wird bestimmt. (vgl. `distance_rr_min`).

'distance_center' Der euklidsche Abstand vom Schwerpunkt von `Pattern` zum Schwerpunkt jeder Region aus `Regions` wird bestimmt.

'covers' Es wird geprüft wie gut die Region `Pattern` in die Regionen aus `Regions` passt. Wenn es keine Verschiebung gibt, sodass `Pattern` eine Teilmenge von `Regions` ist, so ist die Überdeckung 0. Wenn `Pattern` nach einer entsprechenden Verschiebung mit der Region übereinstimmt, ist die Überdeckung 100. Ansonsten wird die Fläche des Opening von `Regions` mit `Pattern` mit der Fläche von `Regions` ins Verhältnis gesetzt (in Prozent).

'fits' Es wird getestet, ob `Pattern` so verschoben werden kann, dass es in `Regions` passt. Wenn dies möglich ist, wird die zugehörige Region aus `Regions` kopiert. Die Parameter `Min` und `Max` haben hier keine Bedeutung.

'overlaps_abs' Die Fläche der Schnittmenge von `Pattern` und jeder Region in `Regions` wird berechnet.

'overlaps_rel' Die Fläche der Schnittmenge von `Pattern` und jeder Region in `Regions` wird berechnet. Die relative Überlappung ergibt sich aus dem Verhältnis der Fläche der Schnittmenge und der Fläche der jeweiligen Region aus `Regions` (in Prozent).

Parameter

▷ **`Regions`** (input_object) . . . . . region(-array) ⇝ *object*
Zu untersuchende Regionen.

▷ **`Pattern`** (input_object) . . . . . region(-array) ⇝ *object*
Region, die mit `Regions` verglichen wird.

▷ **`SelectedRegions`** (output_object) . . . . . region(-array) ⇝ *object*
Regionen, die die Bedingung erfüllen.

▷ **`Feature`** (input_control) . . . . . string(-array) ⇝ *string*
Zu testende Formmerkmale.
Defaultwert: 'covers'
Werteliste: `Feature` ∈ {'distance_center', 'distance_dilate', 'distance_contour', 'covers', 'fits', 'overlaps_abs', 'overlaps_rel'}

▷ **`Min`** (input_control) . . . . . number ⇝ *real* / integer
Untere Grenze des Merkmals.
Defaultwert: 50.0
Wertevorschläge: `Min` ∈ {0.0, 1.0, 5.0, 10.0, 20.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 99.0, 100.0, 200.0, 400.0}
Typischer Wertebereich: 0.0 ≤ Min
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 5.0

▷ **`Max`** (input_control) . . . . . number ⇝ *real* / integer
Obere Grenze des Merkmals.
Defaultwert: 100.0
Wertevorschläge: `Max` ∈ {0.0, 10.0, 20.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 99.0, 100.0, 200.0, 300.0, 400.0}
Typischer Wertebereich: 0.0 ≤ Max
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 5.0

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  if (argc < 2)
  {
    cout << "Usage: " << argv[0] << " <radius of circle>" << endl;
    exit (1);
```

```
  }

  double   rad = atof (argv[1]);
  HImage   img ("monkey");
  HWindow  w;

  img.Display (w);

  HRegion       circ = HRegion::GenCircle (100, 100, rad);
  HRegionArray  reg  = img.Regiongrowing (3, 3, 5, 0);
  HRegionArray  seg  = reg.SelectShapeProto (circ, "fits", 0, 0);

  w.SetColor  ("red");
  seg.Display (w);
  w.Click ();
  return(0);
}
```

Ergebnis

`select_shape_proto` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`connection`, `draw_region`, `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`

Nachfolger

`select_gray`, `shape_trans`, `reduce_domain`, `count_obj`

Alternativen

`select_shape`

Siehe auch

`opening`, `erosion1`, `distance_rr_min_dil`, `distance_rr_min`

Modul

Foundation

select_shape_std (Regions : SelectedRegions : Shape, Percent :)

Auswählen von Regionen einer vorgegebenen Form.

`select_shape_std` vergleicht die Form der übergebenen Regionen mit den vorgegebenen Formen. Wenn die Region eine ähnliche Gestalt hat, wird sie in die Ausgabe übernommen. Mögliche Werte für `Shape` sind:

'max_area' Es wird die größte Region ausgewählt.

'rectangle1' Es wird mit `smallest_rectangle1` das umschließende Rechteck parallel zu den Koordinatenachsen bestimmt. Wenn die Flächendifferenz in Prozent größer als `Percent` ist, wird die Region übernommen.

'rectangle2' Es wird mit `smallest_rectangle2` das kleinste umschließende Rechteck mit beliebiger Orientierung bestimmt. Wenn die Flächendifferenz in Prozent größer als `Percent` ist, wird die Region übernommen. Als robustere Alternative kann der Operator `select_shape` mit *'rectangularity'* für `Feature` verwendet werden.

Parameter

▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
Eingaberegionen, die selektiert werden soll.

▷ **SelectedRegions** (output_object) . . . . . region(-array) ⇝ *object*
Regionen mit gewünschter Form.

▷ **Shape** (input_control) . . . . . string ⇝ *string*
Zu testende Formmerkmale.
Defaultwert: 'max_area'
Werteliste: `Shape` ∈ {'max_area', 'rectangle1', 'rectangle2'}

▷ **Percent** (input_control) . . . . . real ⇝ *real*
Ähnlichkeitsmaß.
Defaultwert: 70.0
Wertevorschläge: `Percent` ∈ {10.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 100.0}
Typischer Wertebereich: 0.0 ≤ `Percent` ≤ 100.0 (lin)
Minimale Schrittweite: 0.1
Empfohlene Schrittweite: 10.0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `smallest_rectangle1`, `smallest_rectangle2`

Alternativen

`intersection`, `complement`, `area_center`, `select_shape`

Siehe auch

`smallest_rectangle1`, `smallest_rectangle2`, `rectangularity`

Modul

Foundation

smallest_circle (Regions : : : Row, Column, Radius)

Ermitteln des kleinsten umschließenden Kreises einer Region.

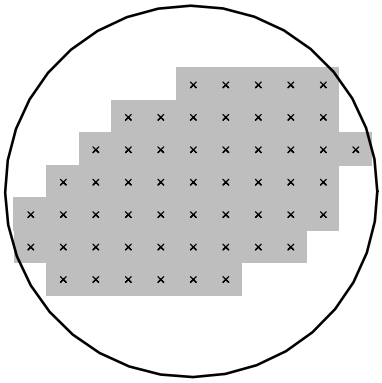
`smallest_circle` bestimmt den kleinsten umschließenden Kreis einer Region, also den Kreis mit dem kleinsten Flächeninhalt unter allen Kreisen, die die Region enthalten. Für diesen Kreis werden der Mittelpunkt (`Row`,`Column`) und der Radius (`Radius`) berechnet. Der Operator findet Anwendung, wenn z.B. die Lage und Größe von kreisförmigen Objekten (z.B. Münzen) bestimmt werden soll, die aber aufgrund schlechter Segmentierung im Inneren nicht homogen sind oder unterbrochene Ränder besitzen. Die Ausgabe des Operators ist so gewählt, dass sie als Eingabe für die Operatoren `disp_circle` und `gen_circle` verwendet werden kann.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in ihren minimalen und maximalen Radien unterscheiden.

Werden mehrere Regionen in `Regions` übergeben, so werden entsprechende Tupel als Ausgabeparameter zurückgegeben. Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Achtung

Intern erfolgt die Berechnung basierend auf den Mittelpunktskoordinaten der Regionenpixel. Um zu berücksichtigen, dass Pixel im Regionenfall eigentlich flächenhaft zu interpretieren sind, wird der berechnete Radius zum Schluss noch um 0.5 vergrößert. Dies gibt in den meisten Fällen ein gutes Ergebnis, im ungünstigsten Fall (Pixeldiagonale) ist diese Vergrößerung allerdings nicht ausreichend. Wollte man sicherstellen, dass der Umriss der Eingaberegion komplett innerhalb des Kreises liegt, müsste man $1/\sqrt{2}$ anstatt 0.5 addieren. Folglich müsste der in `Radius` zurückgegebene Wert um $1/\sqrt{2} - 0.5$ korrigiert werden. Dies wäre allerdings auch nur eine obere Abschätzung, d.h. der Kreis mit dem korrigierten Radius wäre in den meisten Fällen etwas zu groß.



Kleinster umschließender Kreis einer Region. Es ist zu beachten, dass die Berechnung auf den Pixelmittelpunkten basiert und zu dem daraus berechneten Radius 0.5 addiert wird.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **Row** (output_control) ..... circle.center.y(-array) ⇝ *real*
  Zeilenindex des Mittelpunktes.
- ▷ **Column** (output_control) ..... circle.center.x(-array) ⇝ *real*
  Spaltenindex des Mittelpunktes.
- ▷ **Radius** (output_control) ..... circle.radius(-array) ⇝ *real*
  Radius des umschließenden Kreises.
  Zusicherung: Radius >= 0

Beispiel

```
read_image(Image,'fabrik')
regiongrowing(Image,Regions,5,5,6,100)
select_shape(Regions,SelectedRegions,'area','and',100,2000)
smallest_circle(SelectedRegions,Row,Column,Radius)
gen_circle(Circles,Row,Column,Radius)
dev_display (Circles)
```

Komplexität

Sei F die Fläche der Region dann beträgt die Laufzeitkomplexität $O(sqrt(F))$.

Ergebnis

`smallest_circle` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Nachfolger

`gen_circle`, `disp_circle`

Alternativen

elliptic_axis, smallest_rectangle1, smallest_rectangle2

Siehe auch

set_shape, select_shape, inner_circle

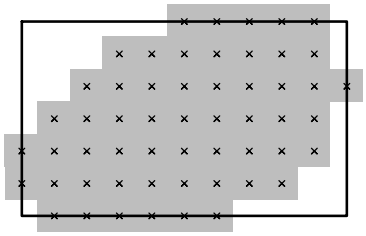
Modul

Foundation

smallest_rectangle1 (Regions : : : Row1, Column1, Row2, Column2)

Ermitteln des umschließenden Rechtecks parallel zu den Koordinatenachsen.

smallest_rectangle1 berechnet das umschließende Rechteck aller Eingaberegionen (parallel zu den Koordinatenachsen). Das umschließende Rechteck wird durch die Koordinaten der Eckpunkte (Row1,Column1,Row2,Column2) beschrieben. Die Berechnung des Rechtecks basiert auf den Mittelpunktskoordinaten der Regionenpixel.



Kleinstes umschließendes achsenparalleles Rechteck einer Region. Es ist zu beachten, dass die Berechnung auf den Pixelmittelpunkten basiert.

Wird mehr als eine Region in Regions übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes im Tupel dem Index einer Region in der Eingabe entspricht. Bei leerer Region haben alle Parameter den Wert 0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Achtung

Bei leerer Region kann das Ergebnis von Row1,Column1, Row2 und Column2 (alle sind 0) zu Konfusionen führen.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **Row1** (output_control) ..... rectangle.origin.y(-array) ⇝ *integer*
  Zeilenindex des linken oberen Eckpunkts.
- ▷ **Column1** (output_control) ..... rectangle.origin.x(-array) ⇝ *integer*
  Spaltenindex des linken oberen Eckpunkts.
- ▷ **Row2** (output_control) ..... rectangle.corner.y(-array) ⇝ *integer*
  Zeilenindex des rechten unteren Eckpunkts.
- ▷ **Column2** (output_control) ..... rectangle.corner.x(-array) ⇝ *integer*
  Spaltenindex des rechten unteren Eckpunkts.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

smallest_rectangle1 liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Nachfolger

`disp_rectangle1`, `gen_rectangle1`

Alternativen

`height_width_ratio`, `smallest_rectangle2`, `area_center`

Siehe auch

`select_shape`

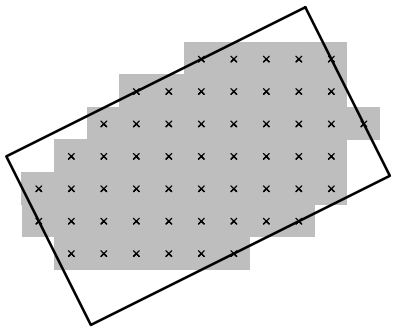
Modul

Foundation

smallest_rectangle2 (Regions : : : Row, Column, Phi, Length1, Length2)

Ermitteln des kleinsten umschließenden Rechtecks mit beliebiger Orientierung.

`smallest_rectangle2` bestimmt das kleinste umschließende Rechteck einer Region, also das Rechteck mit dem kleinsten Flächeninhalt unter allen Rechtecken, die die Region enthalten. Für dieses Rechteck werden der Mittelpunkt, der Neigungswinkel und die beiden Halbmesser berechnet. Die Berechnung des Rechtecks basiert auf den Mittelpunktskoordinaten der Regionenpixel.



Kleinstes umschließendes Rechteck einer Region. Es ist zu beachten, dass die Berechnung auf den Pixelmittelpunkten basiert.

Der Operator findet Anwendung, wenn z.B. die Lage einer Szenerie von mehreren Regionen (z.B. gedruckter Text auf einem rechteckigen Papier bzw. mit rechteckigem Druckbild (Blocksatz)) gefunden werden soll. Die Parameter von `smallest_rectangle2` sind so gewählt, dass sie direkt als Eingabe für die Operatoren `disp_rectangle2` und `gen_rectangle2` verwendet werden können.

In der Dokumentation zu diesem Kapitel (Regionen / Merkmale) finden Sie ein Bild mit Regionen, die sich in der Länge und Phi ihres kleinsten umschließenden Rechtecks unterscheiden.

Wird mehr als eine Region in `Regions` übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes im Tupel dem Index einer Region in der Eingabe entspricht. Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

▷ **Regions** (input_object) . . . . . region(-array) ⇝ *object*
Zu untersuchende Regionen.

▷ **Row** (output_control) . . . . . rectangle2.center.y(-array) ⇝ *real*
Zeilenindex des Mittelpunktes.

▷ **Column** (output_control) . . . . . rectangle2.center.x(-array) ⇝ *real*
Spaltenindex des Mittelpunktes.

▷ **Phi** (output_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real*
Orientierung des umschließenden Rechtecks (Bogenmaß)
Zusicherung: - pi / 2 < Phi && Phi <= pi / 2

▷ **Length1** (output_control) . . . . . rectangle2.hwidth(-array) ⇝ *real*
Erster Halbmesser (halbe Länge) des umschließenden Rechtecks.
Zusicherung: Length1 >= 0.0

▷ **Length2** (output_control) . . . . . rectangle2.hheight(-array) ⇝ *real*
Zweiter Halbmesser (halbe Breite) des umschließenden Rechtecks.
Zusicherung: Length2 >= 0.0 && Length2 <= Length1

Beispiel

```
read_image(Image,'fabrik')
regiongrowing(Image,Regions,5,5,6,100)
smallest_rectangle2(Regions,Row,Column,Phi,Length1,Length2)
gen_rectangle2(Rectangle,Row,Column,Phi,Length1,Length2)
dev_set_draw ('margin')
dev_display(Rectangle)
```

Komplexität

Sei F die Fläche der Region und N die Anzahl der Stützpunkte der konvexen Hülle, dann beträgt die Laufzeitkomplexität $O(sqrt(F) + N^2)$.

Ergebnis

`smallest_rectangle2` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Nachfolger

`disp_rectangle2`, `gen_rectangle2`

Alternativen

`elliptic_axis`, `smallest_rectangle1`

Siehe auch

`smallest_circle`, `set_shape`

Modul

Foundation

spatial_relation (Regions1, Regions2 : : Percent : RegionIndex1, RegionIndex2, Relation1, Relation2)

Ermitteln der Lagebeziehung von Regionen bzgl. der Koordinatenachsen.

spatial_relation wählt Regionen aus, die um Percent Prozent „links“, „rechts“, „oberhalb“ oder „unterhalb“ von anderen Regionen liegen. Regions1 und Regions2 enthalten die zu vergleichenden Regionen. Regions1 kann dabei auf drei verschiedene Arten besetzt werden:

- Regions1 enthält ein leeres Objekttupel, d.h., count_obj liefert *0* zurück:
  In diesem Fall werden alle Regionen in Regions2 permutativ auf Nachbarschaft getestet.
- Regions1 besteht aus einer Region:
  Die Regionen von Regions1 werden mit allen Regionen in Regions2 verglichen.
- Regions1 besteht aus gleich vielen Regionen wie Regions2:
  Hier werden jeweils die Regionen an n-ter Position in Regions1 und Regions2 auf die Nachbarschaftsbeziehung untersucht.

Der Prozentsatz Percent wird so interpretiert, dass die Fläche der zweiten Region um mindestens Percent Prozent echt links/rechts bzw. oberhalb/unterhalb der Regionengrenzen der ersten Region liegen muss. Die Indizes der Regionen, die mindestens eine dieser Bedingungen erfüllen, stehen dann an n-ter Position in den Ausgabeparametern RegionIndex1 und RegionIndex2. Zusätzlich enthalten die Ausgabeparameter Relation1 und Relation2 an n-ter Position die Art der Relation, in der das Regionenpaar (RegionIndex1[n], RegionIndex2[n]) steht, d.h. Region mit Index RegionIndex2[n] steht mit Region mit Index RegionIndex1[n] in der Relation1[n] und Relation2[n].

Mögliche Werte für Relation1 und Relation2 sind:

- Relation1: *'left'*, *'right'* oder *''*
- Relation2: *'above'*, *'below'* oder *''*

In RegionIndex1 und RegionIndex2 werden die Indizes der Regionen in den Tupeln der Eingaberegionen (Regions1 bzw. Regions2) als Bildidentifikatoren eingetragen. Der Zugriff auf ausgewählte Regionen über den Index kann mit copy_obj erfolgen.

Parameter

▷ **Regions1** (input_object) . . . . . region(-array) ⇝ *object*
Ausgangsregionen.

▷ **Regions2** (input_object) . . . . . region(-array) ⇝ *object*
Vergleichsregionen.

▷ **Percent** (input_control) . . . . . integer ⇝ *integer*
Prozentzahl der Fläche der Vergleichsregion, die links/rechts bzw. oberhalb/unterhalb der Regionengrenzen der Ausgangsregion liegen muss.
Defaultwert: 50
Wertevorschläge: Percent ∈ {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Typischer Wertebereich: 0 ≤ Percent ≤ 100 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: 0 <= Percent && Percent <= 100

▷ **RegionIndex1** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der Regionen im Tupel der Eingaberegionen, die die Lagerelation erfüllen.

▷ **RegionIndex2** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der Regionen im Tupel der Eingaberegionen, die die Lagerelation erfüllen.

▷ **Relation1** (output_control) . . . . . string-array ⇝ *string*
Horizontale Lagerelation, in der RegionIndex2[n] mit RegionIndex1[n] steht.

▷ **Relation2** (output_control) . . . . . string-array ⇝ *string*
Vertikale Lagerelation, in der RegionIndex2[n] mit RegionIndex1[n] steht.

Ergebnis

spatial_relation liefert den Wert TRUE, falls Regions2 nicht leer und Percent korrekt besetzt ist. Das Verhalten bei leerem Parameter Regions2 (keine Eingaberegionen vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, regiongrowing, connection

Alternativen

area_center, intersection

Siehe auch

select_region_spatial, find_neighbors, copy_obj, obj_to_integer

Modul

Foundation

22.5 Tests

test_equal_region (Regions1, Regions2 : : : IsEqual)

Test, ob die Regionen von zwei Objekten identisch sind.

test_equal_region vergleicht die Regionen der beiden Eingabeparameter. Verglichen wird das n-te Element in Regions1 mit dem n-ten Objekt in Regions2 (für alle n). Falls alle Regionen gleich sind und die Anzahl der Regionen identisch ist, erhält IsEqual den Wert TRUE, ansonsten FALSE.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

- ▷ **Regions1** (input_object) ..... region(-array) ⇝ *object*
  Testregionen.
- ▷ **Regions2** (input_object) ..... region(-array) ⇝ *object*
  Vergleichsregionen.
  Parameteranzahl: Regions1 == Regions2
- ▷ **IsEqual** (output_control) ..... integer ⇝ *integer*
  Boolescher Ergebniswert.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität bei Ergebnis TRUE $O(1)$ oder $O(sqrt(F))$, bei Ergebnis FALSE $O(sqrt(F))$.

Ergebnis

test_equal_region liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Unterscheidet sich die Anzahl der Objekte, wird eine Fehlerbehandlung durchgeführt. Ansonsten liefert test_equal_region den Wert TRUE

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

intersection, complement, area_center

Siehe auch

test_equal_obj

Modul

Foundation

test_region_point (Regions : : Row, Column : IsInside)

Testen, ob eine Region einen gegebenen Punkt enthält.

Ist ein Array von Regionen und ein einzelner Testpunkt gegeben, so prüft test_region_point, ob mindestens eine Eingaberegion aus Regions den Testpunkt (Row,Column) enthält. Ist dies der Fall, so wird IsInside auf TRUE gesetzt, andernfalls auf FALSE. Falls andererseits eine einzelne Region und mehrere Testpunkte gegeben sind, prüft test_region_point ob mindestens einer der Testpunkte (aus Row,Column) in der Region Regions enthalten ist. Ist dies der Fall, so wird IsInside auf TRUE gesetzt, andernfalls auf FALSE. Falls (Row,Column) reellwertig sind, wird intern auf ganze Pixel gerundet.

Achtung

Der Testpunkt ist nicht in einer leeren Region enthalten (kein Punkt der Region stimmt mit dem Punkt überein). Sind alle Regionen leer, wird IsInside auf FALSE gesetzt.

Bei einem leeren Eingabeobjekt wird im Fall set_system('no_object_result','true') ein leeres Tupel in IsInside zurückgegeben.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Zu untersuchende Region.
- ▷ **Row** (input_control) ..... point.y(-array) ⇝ *integer* / real
  Zeilenindex des Testpunktes.
  Defaultwert: 100
- ▷ **Column** (input_control) ..... point.x(-array) ⇝ *integer* / real
  Spaltenindex des Testpunktes.
  Parameteranzahl: Row == Column
  Defaultwert: 100
- ▷ **IsInside** (output_control) ..... integer ⇝ *integer*
  Boolscher Ergebniswert.

Komplexität

Sei F die Fläche einer Region, N die Anzahl der Regionen und M die Anzahl der Testpunkte. Bei mehreren Regionen und einem Testpunkt beträgt die mittlere Laufzeitkomplexität $O(ln(sqrt(F)) * N)$. Bei einer Region und mehreren Testpunkten beträgt die mittlere Laufzeitkomplexität $O(ln(sqrt(F)) * M)$.

Ergebnis

test_region_point liefert den Wert TRUE, falls eine Region den Testpunkt enthält. Ist dies nicht der Fall, gibt test_region_point FALSE aus. Das Verhalten bei leerem Eingabeobjekt lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, regiongrowing, connection

Alternativen

union1, intersection, area_center

Siehe auch

select_region_point

Modul

Foundation

test_subset_region (Region1, Region2 : : : IsSubset)

Test, ob eine Region in einer anderen enthalten ist.

test_subset_region testet, ob Region1 eine Untermenge von Region2 ist und gibt das Ergebnis in IsSubset zurück. Falls mehrere Regionen übergeben werden, muss die Anzahl der Regionen in Region1 und Region2 übereinstimmen. In diesem Fall wird in IsSubset ein Tupel zurückgegeben, das genauso viele Elemente enthält, wie Region1 und Region2.

Die in HALCON verfügbaren ikonischen Objekte werden bei der Einleitung zu Kapitel Objekt vorgestellt.

Parameter

- ▷ **Region1** (input_object) . . . . . region(-array) ⇝ *object*
  Testregion.
- ▷ **Region2** (input_object) . . . . . region(-array) ⇝ *object*
  Vergleichsregion.
  Parameteranzahl: Region1 == Region2
- ▷ **IsSubset** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ist Region1 in Region2 enthalten?

Ergebnis

test_subset_region liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Unterscheidet sich die Anzahl der Objekte, wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Alternativen

difference, area_center

Siehe auch

test_equal_region, compare_obj

Modul

Foundation

22.6 Transformationen

background_seg (Foreground : BackgroundRegions : :)

Ermitteln zusammenhängender Regionen des Hintergrundes.

background_seg liefert zusammenhängende Hintergrundregionen. Die Prozedur wird typischerweise nach der Kantensuche (inkl. Verdünnung) angewandt, um die von den Kanten begrenzten Bild-Segmente zu ermitteln. Die Berechnung der Zusammenhangskomponenten erfolgt mit der 4-er Nachbarschaft.

Parameter

- ▷ **Foreground** (input_object) . . . . . region(-array) ⇝ *object*
  Regionenkanten.
- ▷ **BackgroundRegions** (output_object) . . . . . region-array ⇝ *object*
  Die zusammenhängenden Hintergrundkomponenten in Foreground.

Beispiel

```
* Simulation of background_seg:
background_seg(Foreground,BackgroundRegions)
  complement(Foreground,Background)
  get_system('neighborhood',Save)
  set_system('neighborhood',4)
```

```
  connection(Background,BackgroundRegions)
  set_system('neighborhood',Save)

* Segmentation with edge filter:
read_image(Image,'fabrik')
sobel_dir(Image,Sobel,Dir,'sum_sqrt',3)
threshold(Sobel,Edges,20,255)
skeleton(Edges,Margins)
background_seg(Margins,Regions)
```

Komplexität

Sei F die Fläche des Hintergrundes, H die Bildhöhe und N die Anzahl der Ergebnisregionen, dann ist die Laufzeitkomplexität: $O(H + sqrt(F) * sqrt(N))$.

Ergebnis

`background_seg` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `connection`, `regiongrowing`, `pouring`, `class_ndim_norm`

Nachfolger

`select_shape`

Alternativen

`complement`, `connection`

Siehe auch

`threshold`, `hysteresis_threshold`, `skeleton`, `expand_region`, `set_system`, `sobel_amp`, `edges_image`, `roberts`, `bandpass_image`

Modul

Foundation

clip_region (Region : RegionClipped : Row1, Column1, Row2, Column2 :)

Reduzieren der Region auf ein Rechteck.

`clip_region` bildet den Durchschnitt aller Regionen in der Eingabe mit dem Rechteck das durch die vier Steuerparameter festgelegt wird. Dabei ist `clip_region` effizienter als `intersection` mit einem Rechteck das als Region (`gen_rectangle1`) dargestellt ist.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Zu transformierende Regionen.

▷ **RegionClipped** (output_object) ..... region(-array) ⇝ *object*
Beschnittene Regionen.

▷ **Row1** (input_control) ..... rectangle.origin.y ⇝ *integer*
Zeilen-Koordinate linkes oberes Eck des Rechtecks.
Defaultwert: 0
Wertevorschläge: `Row1` ∈ {0, 128, 200, 256}
(lin)

▷ **Column1** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spalten-Koordinate linkes oberes Eck.
Defaultwert: 0
Wertevorschläge: Column1 ∈ {0, 128, 200, 256}
(lin)

▷ **Row2** (input_control) . . . . . rectangle.corner.y ⇝ *integer*
Zeilen-Koordinate rechtes unteres Eck des Rechtecks.
Defaultwert: 256
Wertevorschläge: Row2 ∈ {128, 200, 256, 512}
Typischer Wertebereich: 0 ≤ Row2 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Column2** (input_control) . . . . . rectangle.corner.x ⇝ *integer*
Spalten-Koordinate rechtes unteres Eck.
Defaultwert: 256
Wertevorschläge: Column2 ∈ {128, 200, 256, 512}
Typischer Wertebereich: 0 ≤ Column2 ≤ 511 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

Ergebnis

clip_region liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection, regiongrowing, pouring

Nachfolger

select_shape, disp_region

Alternativen

intersection, gen_rectangle1, clip_region_rel

Modul

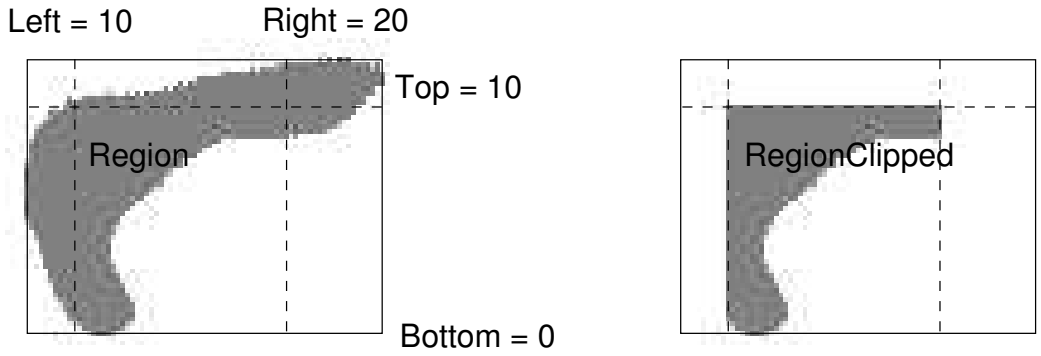
Foundation

clip_region_rel (Region : RegionClipped : Top, Bottom, Left, Right :)

Beschneiden der Region relativ zu ihrem kleinsten umschließenden Rechteck.

clip_region_rel verkleinert die Region dadurch, dass Bereiche weggeschnitten werden, die nahe am kleinsten umschließenden Rechteck der Region liegen. Konkret bedeutet das, dass der Durchschnitt der Region mit einem Rechteck gebildet wird, welches kleiner als oder gleich groß wie das kleinste umschließende Rechteck ist. Die resultierende verkleinerte Region wird in RegionClipped zurückgegeben.

Das Rechteck, mit dem die Region geschnitten wird, wird dadurch bestimmt, dass das kleinste umschließende achsenparallele Rechteck oben, unten, links und rechts jeweils um die Werte Top, Bottom, Left und Right verkleinert wird.



Region mit kleinstem umschließenden Rechteck (1) und geschnittene Region (2).

Diese vier Werte müssen größer oder gleich Null sein. Werden alle vier Werte auf Null gesetzt, so ist `RegionClipped` identisch mit der Eingaberegion.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Zu transformierende Regionen.
- ▷ **`RegionClipped`** (output_object) ..... region(-array) ⇝ *object*
  Beschnittene Regionen.
- ▷ **`Top`** (input_control) ..... integer ⇝ *integer*
  Anzahl Zeilen die „oben“ abgeschnitten werden.
  Defaultwert: 1
  Wertevorschläge: `Top` ∈ {0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50}
  Typischer Wertebereich: 0 ≤ Top (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **`Bottom`** (input_control) ..... integer ⇝ *integer*
  Anzahl Zeilen die „unten“ abgeschnitten werden.
  Defaultwert: 1
  Wertevorschläge: `Bottom` ∈ {0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50}
  Typischer Wertebereich: 0 ≤ Bottom (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **`Left`** (input_control) ..... integer ⇝ *integer*
  Anzahl Spalten die „links“ abgeschnitten werden.
  Defaultwert: 1
  Wertevorschläge: `Left` ∈ {0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50}
  Typischer Wertebereich: 0 ≤ Left (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **`Right`** (input_control) ..... integer ⇝ *integer*
  Anzahl Zeilen die „rechts“ abgeschnitten werden.
  Defaultwert: 1
  Wertevorschläge: `Right` ∈ {0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50}
  Typischer Wertebereich: 0 ≤ Right (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1

Ergebnis

`clip_region_rel` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection, regiongrowing, pouring

Nachfolger

select_shape, disp_region

Alternativen

smallest_rectangle1, intersection, gen_rectangle1, clip_region

Modul

Foundation

```
closest_point_transform ( Region : Distances,
    ClosestPoints : Metric, Foreground, ClosestPointMode, Width,
    Height : )
```

Berechnen der Nächste-Punkte-Transformation einer Region.

closest_point_transform berechnet für jedes Pixel der Eingaberegion Region (bzw. deren Komplement) den Abstand zum nächstgelegenen Pixel außerhalb der Region (d.h. zu dem Pixel auf dem äußeren Rand der Region) und gibt diese Information in Distances zurück. Zusätzlich zum Abstand wird noch das zugehörige nächstgelegenen Pixel in ClosestPoints zurückgeliefert.

Die Auswahl der Pixel, für die der Abstand berechnet werden soll, kann mit dem Parameter Foreground gesteuert werden. Falls Foreground = *'true'*, werden die Abstände nur für die Pixel innerhalb der Region berechnet. Falls Foreground = *'false'*, werden die Abstände der Pixel außerhalb der Regionen zu deren Rand berechnet. Die Abstände und nächstgelegenen Pixel werden für alle Punkte der Ausgabebilder Distances und ClosestPoints berechnet. Die Größe der Ausgabebilder wird durch Width und Height bestimmt. Die Eingaberegion wird auf die Größe der Ausgabebilder beschnitten. Falls es wichtig ist, dass die Abstände innerhalb der gesamten Region berechnet werden, sollte die Region so verschoben werden (siehe move_region), dass sie nur positive Koordinaten besitzt und die Breite und Höhe des Ausgabebildes sollten groß genug gewählt werden, dass die Region vollständig darin enthalten ist. Die entsprechende Information kann mit smallest_rectangle1 bestimmt werden.

Welche Metrik zur Berechnung des Abstandes verwendet wird, wird mit Hilfe des Parameters Metric festgelegt. Falls Metric = *'city-block'* wird der Abstand aus dem kürzesten Pfad eines Punktes zum Rand der Region berechnet, wobei nur horizontale und vertikale „Bewegungen“ erlaubt sind. Diese werden mit einem Gewicht von 1 bewertet. Falls Metric = *'chessboard'* wird der Abstand aus dem kürzesten Pfad eines Punktes zum Rand der Region berechnet, wobei horizontale, vertikale und diagonale „Bewegungen“ erlaubt sind. Diese werden mit einem Gewicht von 1 bewertet. Falls Metric = *'octagonal'* wird eine Kombination der beiden Verfahren verwendet, so dass diagonale Pfade ein etwas größeres Gewicht erhalten. Falls Metric = *'chamfer-3-4'* werden horizontale und vertikale Bewegungen mit einem Gewicht von 3 und diagonale Bewegungen mit einem Gewicht von 4 bewertet. Zur Normierung wird das resultierende Distanzbild durch 3 geteilt. Da diese Normierung Zeit kostet und man normalerweise nur an den relativen Abständen mehrerer Punkte interessiert ist, kann die Normierung mit Metric = *'chamfer-3-4-unnormalized'* unterdrückt werden. Schließlich wird für Metric = *'euclidean'* eine annähernd euklidische Distanz berechnet.

Der Parameter ClosestPointMode legt fest, wie die nächstgelegenen Punkte abgelegt werden. Für ClosestPointMode = *'absolute'* werden in ClosestPoints absolute Koordinaten abgespeichert. Für ClosestPointMode = *'relative'* wird in ClosestPoints der Versatz zur Koordinate des jeweiligen Pixels abgespeichert.

Achtung

Es ist zu beachten, dass im Normalfall die nächstgelegenen Punkte nicht eindeutig sind, d.h. dass für den in Distances im jeweiligen Pixel des Ausgabebildes zurückgelieferten Abstand typischerweise mehrere Punkte auf dem äußeren Rand der Region existieren, die diesen Abstand zu dem Pixel besitzen. So haben zum Beispiel alle Punkten auf dem Skelett der Region in der verwendeten Metrik zu mindestens zwei verschiedenen Punkten auf dem äußeren Rand der Region denselben Abstand. closest_point_transform liefert in diesem Fall einen

dieser Punkte zurück, der durch die Implementierung des Algorithmus bestimmt ist. Insbesondere sollten *keine* Invarianzen bezüglich Rotation oder Spiegelung der Region erwartet werden.

Weiterhin ist zu beachten, dass für `Foreground` = *'true'* auch Punktkoordinaten zurückgeliefert werden, die außerhalb des Bildes liegen, falls die Eingaberegion `Region` den Rand des durch `Width` und `Height` bestimmten Bildes berührt, denn der äußere Rand der Region liegt in diesem Fall ein Pixel außerhalb des Bildes. Falls die zurückgelieferten Koordinaten zum direkten Zugriff auf ein Bild verwendet werden sollen, ist also auf geeignete Randbehandlung zu achten.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Region, zu deren Rand die Distanz berechnet werden soll.
- ▷ **`Distances`** (output_object) ..... image ⇝ *object :* int4
  Rückgabebild, das die Distanz-Information enthält.
- ▷ **`ClosestPoints`** (output_object) ..... image ⇝ *object :* vector_field
  Rückgabebild, das die Koordinaten der nächstgelegenen Punkte enthält.
- ▷ **`Metric`** (input_control) ..... string ⇝ *string*
  Metrik, die bei der Nächste-Punkte-Transformation verwendet werden soll.
  Defaultwert: 'city-block'
  Werteliste: `Metric` ∈ {'city-block', 'chessboard', 'octagonal', 'chamfer-3-4', 'chamfer-3-4-unnormalized', 'euclidean'}
- ▷ **`Foreground`** (input_control) ..... string ⇝ *string*
  Berechnung der Distanzen innerhalb (*'true'*) oder außerhalb (*'false'*) der Eingaberegion.
  Defaultwert: 'true'
  Werteliste: `Foreground` ∈ {'true', 'false'}
- ▷ **`ClosestPointMode`** (input_control) ..... string ⇝ *string*
  Modus, in dem die Koordinaten der nächstgelegenen Punkte zurückgeliefert werden.
  Defaultwert: 'absolute'
  Werteliste: `ClosestPointMode` ∈ {'absolute', 'relative'}
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Breite der Ausgabebilder.
  Defaultwert: 640
  Wertevorschläge: `Width` ∈ {160, 192, 320, 384, 640, 768}
  Typischer Wertebereich: 1 ≤ Width
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Ausgabebilder.
  Defaultwert: 480
  Wertevorschläge: `Height` ∈ {120, 144, 240, 288, 480, 576}
  Typischer Wertebereich: 1 ≤ Height

Komplexität

Die Laufzeit-Komplexität ist $O(\texttt{Width} * \texttt{Height})$.

Ergebnis

`closest_point_transform` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `dyn_threshold`, `regiongrowing`

Nachfolger

`threshold`, `vector_field_to_real`

Alternativen

`distance_transform`

Siehe auch

`skeleton`

Literatur

Y. Ge, C.R. Maurer, Jr., J.M. Fitzpatrick: „Surface-based 3-D image registration using the Iterative Closest Point algorithm with a closest point transform“; in: „Medical Imaging 1996: Image Processing“, M.H. Loew, K.M. Hanson, Editors, Proc. SPIE 2710, pages 358–367, 1996.

P. Soille: „Morphological Image Analysis, Principles and Applications“; Springer Verlag Berlin Heidelberg New York, 1999.

G. Borgefors: „Distance Transformations in Arbitrary Dimensions“; Computer Vision, Graphics, and Image Processing, Vol. 27, pages 321–345, 1984.

P.E. Danielsson: „Euclidean Distance Mapping“; Computer Graphics and Image Processing, Vol. 14, pages 227–248, 1980.

Modul

Foundation

connection (Region : ConnectedRegions : :)

Berechnen von zusammenhängenden Regionen.

`connection` bestimmt die Zusammenhangskomponenten der Regionen in `Region`, zerlegt also die Regionen in zusammenhängende Bereiche. Die Nachbarschaft wird mit `set_system('neighborhood',<4/8>)` eingestellt. Voreingestellt ist die 8-ter Nachbarschaft, die für Vordergrund sinnvoll eingesetzt werden kann. Die maximale Anzahl von Zusammenhangskomponenten, die `connection` zurückliefert, kann mittels `set_system('max_connection',<Num>)` festgelegt werden. Die Voreinstellung von *0* bewirkt, dass alle Zusammenhangskomponenten zurückliefert werden. Die Umkehrung von `connection` für eine zusammenhängende Region ist `union1`.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Zu zerlegende Region.
- ▷ **ConnectedRegions** (output_object) ..... region-array ⇝ *object*
  Zusammenhängende Regionen.

Beispiel

```
read_image(Image,'clip')
dev_set_colored(12)
threshold(Image,Dark,0,150)
count_obj(Dark,NumThresholded)
dev_display (Dark)
connection(Dark,ConnectedRegions)
count_obj(ConnectedRegions,NumConnected)
dev_display (ConnectedRegions)
```

Komplexität

Sei F die Fläche der Eingaberegion und N die Anzahl der daraus erzeugten Zusammenhangskomponenten, dann ist die Laufzeitkomplexität: $O(sqrt(F) * sqrt(N))$.

Ergebnis

`connection` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

auto_threshold, threshold, dyn_threshold, erosion1

Nachfolger

select_shape, select_gray, shape_trans, set_colored, dilation1, count_obj, reduce_domain, add_channels

Alternativen

background_seg

Siehe auch

set_system, union1

Modul

Foundation

distance_transform (Region : DistanceImage : Metric, Foreground, Width, Height :)

Berechnen der Distanztransformation einer Region.

distance_transform berechnet für jeden Punkt der Eingaberegion Region (bzw. deren Komplement) den Abstand zum Rand der Region. Die Auswahl der Pixel, für die der Abstand berechnet werden soll, kann mit dem Parameter Foreground gesteuert werden. Falls Foreground = *'true'*, werden die Abstände nur für die Pixel innerhalb der Region berechnet. Falls Foreground = *'false'*, werden die Abstände der Pixel außerhalb der Regionen zu deren Rand berechnet. Die Abstände werden für alle Punkte des Ausgabebildes DistanceImage berechnet. Die Größe des Ausgabebildes wird durch Width und Height bestimmt. Die Eingaberegion wird auf die Größe des Ausgabebildes beschnitten. Falls es wichtig ist, dass die Abstände innerhalb der gesamten Region berechnet werden, sollte die Region so verschoben werden (siehe move_region), dass sie nur positive Koordinaten besitzt und die Breite und Höhe des Ausgabebildes sollten groß genug gewählt werden, dass die Region vollständig darin enthalten ist. Die entsprechende Information kann mit smallest_rectangle1 bestimmt werden.

Welche Metrik zur Berechnung des Abstandes verwendet wird, wird mit Hilfe des Parameters Metric festgelegt. Falls Metric = *'city-block'* wird der Abstand aus dem kürzesten Pfad eines Punktes zum Rand der Region berechnet, wobei nur horizontale und vertikale „Bewegungen“ erlaubt sind. Diese werden mit einem Gewicht von 1 bewertet. Falls Metric = *'chessboard'* wird der Abstand aus dem kürzesten Pfad eines Punktes zum Rand der Region berechnet, wobei horizontale, vertikale und diagonale „Bewegungen“ erlaubt sind. Diese werden mit einem Gewicht von 1 bewertet. Falls Metric = *'octagonal'* wird eine Kombination der beiden Verfahren verwendet, so dass diagonale Pfade ein etwas größeres Gewicht erhalten. Falls Metric = *'chamfer-3-4'* werden horizontale und vertikale Bewegungen mit einem Gewicht von 3 und diagonale Bewegungen mit einem Gewicht von 4 bewertet. Zur Normierung wird das resultierende Distanzbild durch 3 geteilt. Da diese Normierung Zeit kostet und man normalerweise nur an den relativen Abständen mehrerer Punkte interessiert ist, kann die Normierung mit Metric = *'chamfer-3-4-unnormalized'* unterdrückt werden. Schließlich wird für Metric = *'euclidean'* eine annähernd euklidische Distanz berechnet.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Region, zu deren Rand die Distanz berechnet werden soll.
- ▷ **DistanceImage** (output_object) ..... image ⇝ *object :* int4
  Rückgabebild, das die Distanz-Information enthält.
- ▷ **Metric** (input_control) ..... string ⇝ *string*
  Metrik, die bei der Distanztransformation verwendet werden soll.
  Defaultwert: 'city-block'
  Werteliste: Metric ∈ {'city-block', 'chessboard', 'octagonal', 'chamfer-3-4', 'chamfer-3-4-unnormalized', 'euclidean'}
- ▷ **Foreground** (input_control) ..... string ⇝ *string*
  Berechnung der Distanzen innerhalb (*'true'*) oder ausserhalb (*'false'*) der Eingaberegion.
  Defaultwert: 'true'
  Werteliste: Foreground ∈ {'true', 'false'}

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Ausgabebildes.
Defaultwert: 640
Wertevorschläge: `Width` ∈ {160, 192, 320, 384, 640, 768}
Typischer Wertebereich: 1 ≤ Width

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Ausgabebildes.
Defaultwert: 480
Wertevorschläge: `Height` ∈ {120, 144, 240, 288, 480, 576}
Typischer Wertebereich: 1 ≤ Height

Beispiel

```
* Step towards extracting the medial axis of a shape:
gen_rectangle1 (Rectangle1, 0, 0, 200, 400)
gen_rectangle1 (Rectangle2, 200, 0, 400, 200)
union2 (Rectangle1, Rectangle2, Shape)
distance_transform (Shape, DistanceImage, 'chessboard', 'true', 640, 480)
```

Komplexität

Die Laufzeit-Komplexität ist $O($`Width` $*$ `Height`$)$.

Ergebnis

`distance_transform` liefert TRUE zurück, wenn alle Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `dyn_threshold`, `regiongrowing`

Nachfolger

`threshold`

Alternativen

`closest_point_transform`

Siehe auch

`skeleton`

Literatur

P. Soille: „Morphological Image Analysis, Principles and Applications“; Springer Verlag Berlin Heidelberg New York, 1999.

G. Borgefors: „Distance Transformations in Arbitrary Dimensions“; Computer Vision, Graphics, and Image Processing, Vol. 27, pages 321–345, 1984.

P.E. Danielsson: „Euclidean Distance Mapping“; Computer Graphics and Image Processing, Vol. 14, pages 227–248, 1980.

Modul

Foundation

eliminate_runs (Region : RegionClipped : ElimShorter, ElimLonger :)

Eliminieren von Lauflängen vorgegebener Länge.

`eliminate_runs` Löscht alle Sehnen der Lauflängenkodierung der Eingaberegionen, die kürzen als `ElimShorter` oder länger als `ElimLonger` sind.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Zu transformierende Regionen.

▷ **RegionClipped** (output_object) ..... region(-array) ⇝ *object*
Beschnittene Regionen.

▷ **ElimShorter** (input_control) ..... integer ⇝ *integer*
Alle Sehnen die kürzer sind werden unterdrückt.
Defaultwert: 3
Wertevorschläge: `ElimShorter` ∈ {2, 3, 4, 5, 6, 8, 10, 12, 15}
Typischer Wertebereich: 1 ≤ `ElimShorter` ≤ 500 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **ElimLonger** (input_control) ..... integer ⇝ *integer*
Alle Sehnen die länger sind werden unterdrückt.
Defaultwert: 1000
Wertevorschläge: `ElimLonger` ∈ {50, 100, 200, 500, 1000, 2000}
Typischer Wertebereich: 1 ≤ `ElimLonger` ≤ 10000 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

Ergebnis

`eliminate_runs` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `connection`, `regiongrowing`, `pouring`

Nachfolger

`erosion1`, `dilation1`, `disp_region`

Alternativen

`shape_trans`

Modul

Foundation

expand_region (Regions, ForbiddenArea : RegionExpanded : Iterations, Mode :)

Auffüllen von „Lücken“ zwischen Regionen oder Trennen überlappender Regionen.

`expand_region` dient dazu, Lücken zwischen den Regionen, wie sie z.B. durch Unterdrückung zu kleiner Regionen nach einer Bildsegmentierung entstehen, zu schließen (Modus 'image') oder überlappende Eingaberegionen zu trennen (Modus 'region'). Beide Effekte beruhen auf der Expansion von Regionen. Dabei wird bei jeder Iteration der Expansion ein 1-Pixel breiter Streifen um die Region zu dieser hinzugefügt bzw. aus ihr entfernt.

Expandiert wird dabei nur in Bildbereiche, die nicht als „verbotene Bereiche“ (Parameter `ForbiddenArea`) ausgewiesen sind. Die Zahl der Iterationen wird mit dem Parameter `Iterations` festgelegt. Die Übergabe des strings *'maximal'* veranlasst `expand_region`, das Verfahren so lange zu iterieren, bis es konvergiert, also keine Änderungen mehr auftreten. Eine 0 auf dieser Parameterposition bewirkt die Ausgabe der nicht-überlappenden Teilregionen. Im Detail unterscheiden sich die beiden Modi *'image'* und *'region'* wie folgt:

'image' Die Eingaberegionen werden iterativ so lange ausgedehnt, bis sie eine andere Region oder einen Bildrand berühren. Der Bildrand ist in diesem Fall definiert als das Rechteck, das sich von (0,0) bis (row_max,col_max) erstreckt. Dabei entspricht (row_max,col_max) der rechten unteren Ecke des kleinsten umschließenden Rechtecks aller Eingaberegionen (d.h. aller Regionen, die in `Regions` und in `ForbiddenArea` übergeben werden). Da `expand_region` alle Regionen simultan bearbeitet, werden die „Lücken" zwischen den Regionen „gerecht" auf diese verteilt. Überlappende Regionen werden getrennt, indem ihre gemeinsamen Teilregionen (wiederum „gerecht") auf sie verteilt werden.

'region' Es wird keine Expansion der Eingaberegionen durchgeführt, sondern nur überlappende Regionen getrennt, indem die gemeinsamen Teilregionen „gerecht" auf die Regionen aufgeteilt werden. Da nach dem dafür benötigten „Negativwachstum" der Schnitt mit den Originalregionen gebildet wird, kann es in diesem Modus zu „Lücken" in den Ausgaberegionen kommen, die Segmentierung der Bildebene ist also nicht total. Diesem Problem kann mit einem zweiten Aufruf von `expand_region` mit dem Komplement der ursprünglichen Eingaberegionen als „forbidden area" begegnet werden.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *object*
  Regionen, zwischen denen Lücken geschlossen oder die getrennt werden sollen.
- ▷ **`ForbiddenArea`** (input_object) ..... region ⇝ *object*
  In diesen Bereich darf nicht expandiert werden.
- ▷ **`RegionExpanded`** (output_object) ..... region(-array) ⇝ *object*
  Expandierte oder getrennte Bildregionen.
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer* / string
  Zahl der Iterationen.
  Defaultwert: 'maximal'
  Wertevorschläge: `Iterations` ∈ {'maximal', 0, 1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100, 200}
  Typischer Wertebereich: 0 ≤ `Iterations` ≤ 1000 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 1
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Gewünschter Modus.
  Defaultwert: 'image'
  Werteliste: `Mode` ∈ {'image', 'region'}

Beispiel

```
read_image(Image,'clip')
threshold(Image,Dark,0,150)
connection(Dark,ConnectedRegions)
gen_circle (Circle, 400, 400, 200.5)
expand_region (ConnectedRegions, Circle, RegionExpanded, 'maximal', 'image')
dev_display (RegionExpanded)
```

Ergebnis

`expand_region` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)`, und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<'2 (H_MSG_TRUE)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`pouring`, `threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

dilation1

Siehe auch

expand_gray, interjacent, skeleton

Modul

Foundation

fill_up (Region : RegionFillUp : :)

Auffüllen von Hohlflächen.

fill_up füllt Hohlflächen in Regionen aus, falls solche vorhanden sind. Die Anzahl der Regionen wird dabei nicht verändert. Die Art der Nachbarschaft wird mit set_system('neighborhood',<4/8>) festgelegt (Defaulteinstellung ist 8-ter Nachbarschaft).

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Eingaberegion(en) mit evtl. vorhandenen Hohlflächen.
- ▷ **RegionFillUp** (output_object) ..... region(-array) ⇝ *object*
  Regionen, deren Hohlflächen aufgefüllt sind.

Ergebnis

fill_up liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection, regiongrowing, pouring

Nachfolger

select_shape, disp_region

Alternativen

fill_up_shape

Siehe auch

boundary

Modul

Foundation

fill_up_shape (Region : RegionFillUp : Feature, Min, Max :)

Auffüllen von Hohlflächen mit vorgegebenen Formeigenschaften.

fill_up_shape füllt lediglich diejenigen Hohlflächen in Region auf, die vorgegebenen Formeigenschaften entsprechen. Dabei gibt der Parameter Feature den Namen des Formmerkmals vor. Mit Min und Max wird das Intervall festgelegt, in dem der Merkmalswert der Hohlfläche liegen muss, damit sie ausgefüllt wird.

Parameter

▷ **Region** (input_object) . . . . . region(-array) ⇝ *object*
Eingaberegion(en).

▷ **RegionFillUp** (output_object) . . . . . region(-array) ⇝ *object*
Ausgaberegion(en) mit weniger Hohlflächen.

▷ **Feature** (input_control) . . . . . string ⇝ *string*
Formmerkmal.
Defaultwert: 'area'
Werteliste: `Feature` ∈ {'area', 'compactness', 'convexity', 'anisometry', 'phi', 'ra', 'rb', 'inner_circle', 'outer_circle'}

▷ **Min** (input_control) . . . . . number ⇝ *real* / integer
Mindestwert für Merkmal Feature.
Defaultwert: 1.0
Wertevorschläge: `Min` ∈ {0.0, 1.0, 10.0, 50.0, 100.0, 500.0, 1000.0, 10000.0}
Typischer Wertebereich: 0.0 ≤ Min

▷ **Max** (input_control) . . . . . number ⇝ *real* / integer
Maximalwert für Merkmal Feature.
Defaultwert: 100.0
Wertevorschläge: `Max` ∈ {10.0, 50.0, 100.0, 500.0, 1000.0, 10000.0, 100000.0}
Typischer Wertebereich: 0.0 ≤ Max

Beispiel

```
read_image(&Image,"monkey");
threshold(Image,&Seg,120.0,255.0);
fill_up_shape(Seg,&Filled,"area",0.0,200.0);
```

Ergebnis

`fill_up_shape` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`threshold`, `connection`, `regiongrowing`, `pouring`

Nachfolger

`select_shape`, `disp_region`

Alternativen

`fill_up`

Siehe auch

`select_shape`, `connection`, `area_center`

Modul

Foundation

junctions_skeleton (Region : EndPoints, JuncPoints : :)

Berechnen von Kreuzungs- und Endpunkten in einem Skelett.

junctions_skeleton liefert Kreuzungs- und Endpunkte in einem Skelett (siehe skeleton). Die Kreuzungspunkte in einer der Eingaberegionen aus Region werden als Region in JuncPoints, die Endpunkte entsprechend als Region in EndPoints gespeichert.

Damit junctions_skeleton sinnvolle Ergebnisse liefert sollte Region keine Linien enthalten, die breiter als ein Pixel sind. Die von skeleton erzeugten Regionen erfüllen diese Eigenschaft, während die von morph_skeleton erzeugten Regionen dies im Allgemeinen nicht tun.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Die Skelette.

▷ **EndPoints** (output_object) ..... region(-array) ⇝ *object*
Endpunkte.
Parameteranzahl: EndPoints == Region

▷ **JuncPoints** (output_object) ..... region(-array) ⇝ *object*
Kreuzungspunkte.
Parameteranzahl: JuncPoints == Region

Beispiel

```
* non-connected branches of a skeleton
skeleton(Region,Skeleton)
junctions_skeleton(Skeleton,EPoints,JPoints)
difference(Skeleton,JPoints,Rows)
set_system('neighborhood',4)
connection(Rows,Parts)
```

Komplexität

Sei F die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität: $O(F)$.

Ergebnis

junctions_skeleton liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), das bei leerer Ergebnisregion mit set_system('store_empty_region',<'2 (H_MSG_TRUE) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

skeleton

Nachfolger

area_center, connection, get_region_points, difference

Siehe auch

pruning, split_skeleton_region

Modul

Foundation

merge_regions_line_scan (CurrRegions,
PrevRegions : CurrMergedRegions, PrevMergedRegions : ImageHeight,
MergeBorder, MaxImagesRegion :)

Zusammenfügen von Regionen aus Zeilenkamerabildern.

Der Operator merge_regions_line_scan verbindet Regionen, die aus räumlich aneinandergrenzenden Bildern mit der Höhe ImageHeight segmentiert wurden und die, wenn das alte Bild verschoben wird, an den Bildrändern zusammenstoßen. Die Bilder können beispielsweise von einer Zeilenkamera nacheinander aufgenommen worden sein. Dabei wird davon ausgegangen, dass CurrRegions Regionen des aktuellen Bildes enthält und PrevRegions Regionen des vorherigen Bildes, welches die räumliche Fortsetzung des aktuellen Bildes darstellt.

Mit Hilfe des Parameters MergeBorder kann angegeben werden, ob die Oberkante des aktuellen Bildes an die Unterkante des vorherigen Bildes stößt (*'top'*) oder die Unterkante des aktuellen Bildes an die Oberkante des vorherigen Bildes (*'bottom'*).

Der Parameter MaxImagesRegion bestimmt bei der rekursiven Anwendung des Operators merge_regions_line_scan, wie viel Bilder eine aktuelle Ausgangsregion maximal zurückreichen kann. Der Regionenteil, der über diese Bilder hinausreicht, wird abgeschnitten.

Der Operator merge_regions_line_scan liefert zwei Regionenarrays zurück. PrevMergedRegions enthält alle Regionen, die ausschließlich im alten Bild liegen und nichts mit einer aktuellen Region verbunden werden konnten. In CurrMergedRegions werden dagegen alle aktuellen Regionen, gegebenenfalls vergrößert um die angrenzenden Regionen aus PrevRegions, eingetragen. Die so verbundenen Regionen erscheinen als eine einzige neue Region in CurrMergedRegions, wobei die angefügten Teile um die Bildhöhe nach oben (MergeBorder=*'top'*) oder nach unten (MergeBorder=*'bottom'*) verschoben werden und in der Ergebnisregion in CurrMergedRegions über den Bildrand hinausgehen. Dafür wird durch merge_regions_line_scan der Systemparameter *'clip_region'* (vgl. set_system) auf *'false'* gesetzt.

Parameter

- ▷ **CurrRegions** (input_object) . . . . . region(-array) ⇝ *object*
  Aktuelle Eingaberegionen.
- ▷ **PrevRegions** (input_object) . . . . . region(-array) ⇝ *object*
  Im vorhergehenden Zyklus zusammengefügte Regionen.
- ▷ **CurrMergedRegions** (output_object) . . . . . region(-array) ⇝ *object*
  Aktuelle Regionen, die ggf. mit den alten Regionen verbunden wurden.
- ▷ **PrevMergedRegions** (output_object) . . . . . region(-array) ⇝ *object*
  Alte Regionen, die nicht mit den aktuellen verbunden werden konnten.
- ▷ **ImageHeight** (input_control) . . . . . integer ⇝ *integer*
  Höhe der Ausgangsbilder.
  Defaultwert: 512
  Wertevorschläge: ImageHeight ∈ {240, 480, 512, 1024}
- ▷ **MergeBorder** (input_control) . . . . . string ⇝ *string*
  Im aktuellen Bild die Zeile, die mit dem vorhergehenden Bild zusammenstößt.
  Defaultwert: 'top'
  Werteliste: MergeBorder ∈ {'top', 'bottom'}
- ▷ **MaxImagesRegion** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl der Bilder, über die sich eine Region erstrecken darf.
  Defaultwert: 3
  Wertevorschläge: MaxImagesRegion ∈ {1, 2, 3, 4, 5}

Ergebnis

merge_regions_line_scan liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

partition_dynamic (Region : Partitioned : Distance, Percent :)

Horizontales Teilen einer Region an Stellen mit geringer vertikaler Ausdehnung.

partition_dynamic zerlegt die Eingaberegion Region horizontal in Teilregionen, die etwa die Breite Distance haben. Die Eingaberegion wird dabei an Stellen aufgeschnitten, die eine relativ kleine vertikale Ausdehnung haben.

Die Positionen, an denen die Eingaberegion aufgeschnitten wird werden auf folgende Weise bestimmt: Zuerst werden vorläufige Positionen bestimmt, die gleichabständig über die gesamte Breite der Eingaberegion verteilt sind, d.h., alle resultierenden Teilregionen wären gleich breit. Hierfür wird die Anzahl n der resultierenden Teilregionen bestimmt, indem die Breite der Eingaberegion durch Distance geteilt und das Ergebnis auf die nächste Ganzzahl gerundet wird. Der Abstand zwischen den vorläufigen Positionen ist das Ergebnis der Division der Breite der Eingaberegion durch n. Beachten Sie, dass der Abstand zwischen den vorläufigen Positionen typischerweise nicht mit Distance identisch ist. Die endgültigen Positionen, an denen die Eingaberegion aufgeschnitten wird werden nun in der Umgebung der vorläufigen Positionen so bestimmt, dass die Eingaberegion an Stellen aufgeschnitten wird, an der sie die geringste vertikale Ausdehnung innerhalb dieser Umgebung hat. Die maximale Verschiebung der endgültigen Position gegenüber der vorläufigen Position ist Distance*Percent*0.01.

Die resultierenden Teilregionen werden in Partitioned zurückgegeben. Beachten Sie, dass die Eingaberegion nur dann geteilt wird, wenn ihre Breite größer als 1.5 mal Distance ist.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Region die zerlegt werden soll.

▷ **Partitioned** (output_object) ..... region-array ⇝ *object*
Zerlegte Region.

▷ **Distance** (input_control) ..... real ⇝ *real*
Genäherte Breite der resultierenden Teilregionen.

▷ **Percent** (input_control) ..... real ⇝ *real*
Maximale prozentuale Verschiebung der Positionen an denen die Eingaberegion geteilt wird.
Defaultwert: 20
Wertevorschläge: Percent ∈ {0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 70.0, 90.0, 100.0}
Typischer Wertebereich: $0 \leq$ Percent ≤ 100

Ergebnis

partition_dynamic liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), das bei leerer Ergebnisregion mit set_system('store_empty_region',<'2 (H_MSG_TRUE) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection

Alternativen

partition_rectangle

Siehe auch

intersection, smallest_rectangle1, shape_trans, clip_region

Modul

Foundation

partition_rectangle (Region : Partitioned : Width, Height :)

Aufteilen einer Region in Rechtecke von ungefähr gleicher Größe.

partition_rectangle zerlegt die Eingaberegion in Rechtecke der Größe Width mal Height. Die Rechtecke werden in Partitioned zurückgeliefert. Die Region wird immer in etwa gleich große Rechtecke zerlegt. Falls dies mit der gewünschten Rechteckgröße nicht möglich ist, werden Width und Height so angepasst, dass die resultierenden Rechtecke in etwa gleich groß sind. Wenn die Region kleiner als die angegebene Größe ist, bleibt sie unverändert. Eine Zerlegung findet erst statt, wenn die Größe mindestens 1.5 mal so groß ist wie durch die Parameter vorgegeben.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *object*
  Region die zerlegt werden soll.
- ▷ **Partitioned** (output_object) ..... region(-array) ⇝ *object*
  Zerlegte Region.
- ▷ **Width** (input_control) ..... extent.x ⇝ *real*
  Breite der einzelnen Rechtecke.
- ▷ **Height** (input_control) ..... extent.y ⇝ *real*
  Höhe der einzelnen Rechtecke.

Ergebnis

partition_rectangle liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), das bei leerer Ergebnisregion mit set_system('store_empty_region',<'2 (H_MSG_TRUE) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection

Alternativen

partition_dynamic

Siehe auch

intersection, smallest_rectangle1, shape_trans, clip_region

Modul

Foundation

rank_region (Region : RegionCount : Width, Height, Number :)

Berechnen des Rangoperators für Regionen.

rank_region berechnet den binären Rangoperator. Es wird eine Filtermaske (Rechteck: Height x Width) verwendet. Dabei wird für jede Position die Anzahl der Punkte aus Region gezählt, die in der Rechteckmaske liegen. Ist die Anzahl größer oder gleich Number, dann wird der Schwerpunkt des Rechtecks in der Ergebnisregion eingetragen. Wählt man

$$\text{Number} = \frac{\text{Height} * \text{Width}}{2},$$

so erhält man den Median Operator.

Achtung

Als Werte für `Height` und `Width` sind nur ungerade Zahlen > 3 zugelassen. Werden andere Werte eingegeben, so werden diese automatisch (ohne Fehlermeldung) entsprechend modifiziert (nächstgrößere ungerade Zahl).

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *object*
  Zu transformierende Region(en).
- ▷ **`RegionCount`** (output_object) ..... region(-array) ⇝ *object*
  Ergebnisregion(en)
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert: 15
  Wertevorschläge: `Width` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21}
  Typischer Wertebereich: $3 \leq$ `Width` ≤ 511 (lin)
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
  Restriktion: Width >= 3 && odd(Width)
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert: 15
  Wertevorschläge: `Height` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21}
  Typischer Wertebereich: $3 \leq$ `Height` ≤ 511 (lin)
  Minimale Schrittweite: 2
  Empfohlene Schrittweite: 2
  Restriktion: Height >= 3 && odd(Height)
- ▷ **`Number`** (input_control) ..... integer ⇝ *integer*
  Zahl der Punkte innerhalb der Filtermaske >= Number: Punkt wird in Ausgaberegion aufgenommen.
  Defaultwert: 70
  Wertevorschläge: `Number` ∈ {5, 10, 20, 40, 60, 80, 90, 120, 150, 200}
  Typischer Wertebereich: $1 \leq$ `Number` ≤ 1000 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
  Restriktion: Number > 0

Beispiel

```
read_image(Image,'monkey')
mean_image(Image,Mean,5,5)
dyn_threshold(Image,Mean,Points,25,'light')
rank_region(Points,Textur,15,15,30)
gen_circle(Mask,10,10,3)
opening(Textur,Mask,Seg)
```

Komplexität

Sei F die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität: $O(F * 8)$.

Ergebnis

`rank_region` liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, connection, regiongrowing, pouring, class_ndim_norm

Nachfolger

select_shape, disp_region

Alternativen

closing_rectangle1, expand_region

Siehe auch

rank_image, mean_image

Modul

Foundation

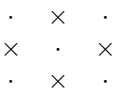
remove_noise_region (InputRegion : OutputRegion : Type :)

Beseitigen von Rauschen bei Regionen.

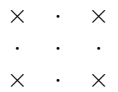
remove_noise_region dient zum Beseitigen von Rauschen bei einer Region.

Abhängig von Type wird folgendes strukturierendes Element verwendet:

'n_4' Es wird ein strukturierendes Element verwendet, das aus den vier 4-er Nachbarn eines Punktes besteht:



'n_8' Es wird ein strukturierendes Element verwendet, das aus den vier 8-er Nachbarn eines Punktes besteht:



'n_48' Es wird ein strukturierendes Element verwendet, das aus den vier 4-er Nachbarn und den vier 8-er Nachbarn eines Punktes besteht:



Mit diesem strukturierenden Element wird eine Dilatation und anschließend der Durchschnitt mit den Eingabedaten durchgeführt. Hierdurch werden alle Punkte, die keinen entsprechenden Nachbarn haben, gelöscht.

Parameter

- ▷ **InputRegion** (input_object) ..... region(-array) ⇝ *object*
  Regionen die modifiziert werden.
- ▷ **OutputRegion** (output_object) ..... region(-array) ⇝ *object*
  Regionen mit weniger Rauschen.
- ▷ **Type** (input_control) ..... string ⇝ *string*
  Modus der Rauschunterdrückung.
  Defaultwert: 'n_4'
  Werteliste: Type ∈ {'n_4', 'n_8', 'n_48'}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität $O(\sqrt{F} * 4)$.

Ergebnis

remove_noise_region liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

connection, regiongrowing, pouring, class_ndim_norm

Nachfolger

select_shape

Siehe auch

dilation1, intersection, gen_region_points

Modul

Foundation

shape_trans (Region : RegionTrans : Type :)

Transformieren der Form von Regionen.

shape_trans dient zur Transformation der Form von Regionen in Abhängigkeit von Type:

'convex' Konvexe Hülle.

'ellipse' Ellipse mit den gleichen Momenten und Fläche wie die Eingaberegion

'outer_circle' Kleinster umschließender Kreis.

'inner_circle' Größter Inkreis.

'rectangle1' Kleinstes umschließendes Rechteck parallel zu den Koordinatenachsen.

'rectangle2' Kleinstes umschließendes Rechteck.

'inner_rectangle1' Größtes enthaltenes achsenparalleles Rechteck.

'inner_center' Liefert als Ausgaberegion den Punkt auf dem Skelett der Eingaberegion, der am nächsten zum Schwerpunkt der Eingaberegion liegt.

Achtung

Für den Fall Type = *'outer_circle'* ist zu beachten, dass die zurückgegebene Kreisregion die ursprüngliche Region in manchen Fällen nicht komplett beinhaltet. Dies liegt daran, dass intern die Operatoren smallest_circle und gen_circle aufgerufen werden. Wie in der Dokumentation von smallest_circle beschrieben, kann der berechnete Radius bis zu $1/\sqrt{2}-0.5$ Pixel zu klein sein. Zusätzlich kommt hinzu, dass der mit gen_circle generierte Kreis bis zu 0.5 Pixel in beide Richtungen also um maximal $1/\sqrt{2}$ Pixel verschoben wird. Insgesamt kann die ursprüngliche Region also um bis zu 1 Pixel über die zurückgegebene Kreisregion hinausragen.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Regionen, die transformiert werden.

▷ **RegionTrans** (output_object) ..... region(-array) ⇝ *object*
Transformierte Regionen.

▷ **Type** (input_control) ..... string ⇝ *string*
Art der Transformation.
Defaultwert: 'convex'
Werteliste: Type ∈ {'convex', 'ellipse', 'outer_circle', 'inner_circle', 'rectangle1', 'rectangle2', 'inner_rectangle1', 'inner_center'}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität $O(F)$.

Ergebnis

shape_trans liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

connection, regiongrowing

Nachfolger

disp_region, regiongrowing_mean, area_center

Siehe auch

convexity, elliptic_axis, area_center, smallest_rectangle1, smallest_rectangle2, inner_rectangle1, set_shape, select_shape, inner_circle

Modul

Foundation

skeleton (Region : Skeleton : :)

Verdünnen von Regionen.

skeleton berechnet das Skelett, d.h., die Mittelachse, der Eingaberegionen. Das Skelett ist so konstruiert, dass jeder Punkt auf dem Skelett als Mittelpunkt eines Kreises mit größtmöglichem Radius angesehen werden kann, welcher noch vollständig in der Region enthalten ist.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Zu verdünnende Regionen.

▷ **Skeleton** (output_object) ..... region(-array) ⇝ *object*
Skelett-Region.
Parameteranzahl: Skeleton == Region

Komplexität

Sei F die Fläche des umschließenden Rechtecks, dann ist die Laufzeitkomplexität: $O(F)$ (pro Region).

Ergebnis

skeleton liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

sobel_amp, edges_image, bandpass_image, threshold, hysteresis_threshold

Nachfolger

junctions_skeleton, pruning

Alternativen

morph_skeleton, thinning

Siehe auch

gray_skeleton, sobel_amp, edges_image, roberts, bandpass_image, threshold

Literatur

Eckardt, U. „Verdünnung mit Perfekten Punkten“, Proceedings 10. DAGM-Symposium, IFB 180, Zurich, 1988

Modul

Foundation

sort_region (Regions : SortedRegions : SortMode, Order, RowOrCol :)

Sortieren von Regionen aufgrund ihrer relativen Lage.

sort_region ordnet die Regionen bezüglich ihrer relativen Lage an. Alle Sortierungen mit Ausnahme von *'character'* verwenden einen Punkt der Region. Diese Punkte werden bei RowOrCol = *'row'* zuerst bzgl. der Zeile dann bzgl. der Spalte sortiert; bei *'column'* wird zuerst der Spaltenwert verwendet. Der Parameter SortMode kann mit folgenden Werten belegt werden:

'character' Die Regionen werden wie Buchstaben aufgefasst, die entweder horizontal oder vertikal gelesen werden. Dabei werden sie in der durch RowOrCol angegebenen Leserichtung angeordnet, wobei folgende Spezifikationen gelten:

- *'row'*: Als Zeilen aufgefasst ist die Leserichtung zunächst von links nach rechts und anschließend von oben nach unten.
- *'column'*: Als Spalten aufgefasst ist die Leserichtung zunächst von oben nach unten und anschließend von links nach rechts.

Für SortMode *'character'* kann ein zweiter numerischer Wert zusätzlich zu *'character'* an SortMode übergeben werden. Dieser Wert spezifiziert den maximalen Prozentsatz an Überlappung. Diese Überlappung ist von dem Parameter RowOrCol abhängig und wird folgendermaßen bestimmt:

- RowOrCol = *'row'*: Überlappung der Zeilenkoordinaten zweier Regionen
- RowOrCol = *'column'*: Überlappung der Spaltenkoordinaten zweier Regionen,

wobei der Prozentsatz sich auf die kleinere der sich überlappenden Regionen (bzgl. dieser Achse) bezieht. Regionen ohne Koordinatenüberlappung oder mit einer Koordinatenüberlappung kleiner als dem eingestellten Prozentsatz werden als in getrennten Zeilen bzw. Spalten betrachtet und entsprechend sortiert. Ist die Überlappung größer, werden sie als in der gleichen Zeile oder Spalte betrachtet und deswegen innerhalb der Zeile oder Spalte sortiert. Wenn man diesen zusätzlichen Parameter zum Beispiel auf 0 setzt, werden alle sich bezüglich der Zeilen- bzw. Spaltenkoordinaten überlappende Regionen als in der gleichen Zeile bzw. Spalte gesehen und sortiert. Der Defaultwert dieses Parameters beträgt 15. Das heißt, dass benachbarte Zeilen oder Spalten sich maximal um 15% überlappen dürfen um als getrennt zu gelten. Es gilt zu beachten, für Order = *'false'* werden die Buchstaben umgekehrt zur oben beschriebenen Leserichtung sortiert.

'first_point' Der Punkt mit dem kleinsten Spaltenwert in der ersten Zeile der Region.

'last_point' Der Punkt mit dem größten Spaltenwert in der letzten Zeile der Region.

'upper_left' Linkes oberes Eck des umschließenden Rechtecks.

'upper_right' Rechtes oberes Eck des umschließenden Rechtecks.

'lower_left' Linkes unteres Eck des umschließenden Rechtecks.

'lower_right' Rechtes unteres Eck des umschließenden Rechtecks.

Der Parameter Order legt fest, ob auf- oder absteigend sortiert wird. Bei *'true'* wird aufsteigend bei *'false'* absteigend sortiert.

Parameter

▷ **Regions** (input_object) . . . . . region-array ⇝ *object*
Anzuordnende Regionen.

▷ **SortedRegions** (output_object) . . . . . region-array ⇝ *object*
Angeordnete Regionen.

▷ **SortMode** (input_control) . . . . . tuple(-array) ⇝ *string* / integer / real
Art der Sortierung.
Defaultwert: 'first_point'
Werteliste: SortMode ∈ {'character', 'first_point', 'last_point', 'upper_left', 'lower_left', 'upper_right', 'lower_right'}

▷ **Order** (input_control) . . . . . string ⇝ *string*
Aufsteigend oder absteigend sortieren.
Defaultwert: 'true'
Werteliste: Order ∈ {'true', 'false'}

▷ **RowOrCol** (input_control) . . . . . string ⇝ *string*
Anordnung erst nach Zeile oder Spalte.
Defaultwert: 'row'
Werteliste: RowOrCol ∈ {'row', 'column'}

Ergebnis

Sind die Parameter korrekt, dann liefert sort_region den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

do_ocr_multi_class_mlp, do_ocr_single_class_mlp

Modul

Foundation

split_skeleton_lines (SkeletonRegion : : MaxDistance : BeginRow, BeginCol, EndRow, EndCol)

Auftrennen von Linien (codiert durch 1 Pixel breite unverzweigte Linien).

split_skeleton_lines trennt Eingabelinien (codiert durch 1 Pixel breite unverzweigte Regionen) gemäß ihrer Krümmung in Teillinien auf. Eine Linie wird dabei aufgetrennt, wenn der maximale Abstand eines Linienpunktes von der Verbindungsgeraden zwischen Anfangs- und Endpunkt der Linie größer als MaxDistance ist (Split & Merge Algorithmus). Zurückgeliefert wird dann die Geradenapproximation (Anfangs- und Endpunkte) an die Eingabelinie.

Achtung

Die Eingaberegionen müssen unverzweigte Linien codieren, also einzelne Äste des Skeletts.

Parameter

▷ **SkeletonRegion** (input_object) . . . . . region-array ⇝ *object*
Eingabelinien (codiert als 1 Pixel breite unverzweigte Regionen).

▷ **MaxDistance** (input_control) . . . . . integer ⇝ *integer*
Krümmungsmaß (max. Abstand der Linienpunkte von der Verbindungsgeraden zwischen Anfangs- und Endpunkt).
Defaultwert: 3
Wertevorschläge: MaxDistance ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Typischer Wertebereich: 1 ≤ MaxDistance ≤ 500 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **BeginRow** (output_control) . . . . . line.begin.y-array ⇝ *integer*
Zeilenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **BeginCol** (output_control) . . . . . line.begin.x-array ⇝ *integer*
Spaltenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **EndRow** (output_control) . . . . . line.end.y-array ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Ausgabelinien.

▷ **EndCol** (output_control) . . . . . line.end.x-array ⇝ *integer*
Spaltenkoordinaten der Endpunkte der Ausgabelinien.

Beispiel

```
read_image(Image,'fabrik')
edges_image (Image, ImaAmp, ImaDir, 'lanser2', 0.5, 'nms', 8, 16)
threshold (ImaAmp, RawEdges, 8, 255)
```

```
skeleton (RawEdges, Skeleton)
junctions_skeleton (Skeleton, EndPoints, JuncPoints)
difference (Skeleton, JuncPoints, SkelWithoutJunc)
connection (SkelWithoutJunc, SingleBranches)
select_shape (SingleBranches, SelectedBranches, 'area', 'and', 16, 99999)
split_skeleton_lines (SelectedBranches, 3, BeginRow, BeginCol, EndRow, \
                      EndCol)
```

Ergebnis

split_skeleton_lines liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), und das bei leerer Ergebnisregion mit set_system('store_empty_region',<'2 (H_MSG_TRUE) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

connection, select_shape, skeleton, junctions_skeleton, difference

Nachfolger

select_lines, partition_lines, disp_line

Siehe auch

split_skeleton_region, detect_edge_segments

Modul

Foundation

split_skeleton_region (SkeletonRegion : RegionLines : MaxDistance :)

Auftrennen von Linien (codiert durch 1 Pixel breite unverzweigte Regionen).

split_skeleton_region trennt Eingabelinien (codiert durch 1 Pixel breite unverzweigte Regionen) gemäß ihrer Krümmung in Teillinien auf. Eine Linie wird dabei aufgetrennt, wenn der maximale Abstand eines Linienpunktes von der Verbindungsgeraden zwischen Anfangs- und Endpunkt der Linie größer als MaxDistance ist (Split & Merge Algorithmus). Zurückgeliefert wird jedoch keine Geradenapproximation an die Eingabelinien, sondern die Originallinien aufgeteilt auf mehrere Ausgaberegionen.

Achtung

Die Eingaberegionen müssen unverzweigte Linien codieren, also einzelne Äste des Skeletts.

Parameter

▷ **SkeletonRegion** (input_object) ..... region(-array) ⇝ *object*
Eingabelinien (codiert als 1 Pixel breite unverzweigte Regionen).

▷ **RegionLines** (output_object) ..... region-array ⇝ *object*
Aufgespaltene Ausgabelinien.

▷ **MaxDistance** (input_control) ..... integer ⇝ *integer*
Krümmungsmaß (max. Abstand der Linienpunkte von der Verbindungsgeraden zwischen Anfangs- und Endpunkt).
Defaultwert: 3
Wertevorschläge: MaxDistance ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Typischer Wertebereich: 1 ≤ MaxDistance ≤ 500 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Beispiel

```
read_image(Image,'fabrik')
edges_image (Image, ImaAmp, ImaDir, 'lanser2', 0.5, 'nms', 8, 16)
threshold (ImaAmp, RawEdges, 8, 255)
skeleton (RawEdges, Skeleton)
junctions_skeleton (Skeleton, EndPoints, JuncPoints)
difference (Skeleton, JuncPoints, SkelWithoutJunc)
connection (SkelWithoutJunc, SingleBranches)
select_shape (SingleBranches, SelectedBranches, 'area', 'and', 16, 99999)
split_skeleton_region (SelectedBranches, Lines, 3)
```

Ergebnis

`split_skeleton_region` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)`, und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<'2 (H_MSG_TRUE)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`connection`, `select_shape`, `skeleton`, `junctions_skeleton`, `difference`

Nachfolger

`count_obj`, `select_shape`, `select_obj`, `area_center`, `elliptic_axis`, `smallest_rectangle2`, `get_region_polygon`, `get_region_contour`

Siehe auch

`split_skeleton_lines`, `get_region_polygon`, `gen_polygons_xld`

Modul

Foundation

22.7 Zugriff

`get_region_contour` (Region : : : Rows, Columns)

Zugriff auf die Kontur eines Objektes.

`get_region_contour` gibt die Kontur einer Region aus. Eine Kontur ist eine Folge von Zeilen- (`Rows`) und Spaltenkoordinaten (`Columns`), die die Umrandung der Region beschreiben. Die Kontur liegt auf der Region. Sie beginnt bei der kleinsten Zeilennummer. In dieser Zeile bei dem Punkt mit dem größten Spaltenindex. Die Umlaufrichtung ist der Uhrzeigersinn. Der erste Punkt der Kontur ist gleich dem letzten. Hohlflächen der Region werden ignoriert. `get_region_contour` liefert die Koordinaten in Form von Tupeln ab. Eine leere Region wird als leeres Tupel übergeben.

Achtung

Hohlflächen der Region werden ignoriert. Es darf nur eine Region übergeben werden und die muss genau eine Zusammenhangskomponente haben.

Parameter

▷ **Region** (input_object) ..... region ⇝ *object*
Auszugebende Region.

▷ **Rows** (output_control) ..... contour.y-array ⇝ *integer*
Zeilennummern der Konturpunkte.

▷ **Columns** (output_control) ..... contour.x-array ⇝ *integer*
Spaltennummern der Konturpunkte.
Parameteranzahl: Columns == Rows

Ergebnis

`get_region_contour` liefert normalerweise den Wert TRUE. Wird mehr als eine Zusammenhangskomponente übergeben, wird eine Fehlerbehandlung ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`sobel_amp`, `threshold`, `skeleton`, `edges_image`, `gen_rectangle1`, `gen_circle`

Siehe auch

`copy_obj`, `get_region_polygon`

Modul

Foundation

get_region_convex (Region : : : Rows, Columns)

Zugriff auf die konvexe Hülle als Kontur.

`get_region_convex` gibt die konvexe Hülle einer Region als Polygon aus. Das Polygon ist die minimale Folge von Zeilen- (`Rows`) und Spaltenkoordinaten (`Columns`), die die Hülle der Region beschreiben. Die Polygonpunkte liegen auf der Region. Das Polygon beginnt bei der kleinsten Zeilennummer; in dieser Zeile bei dem Punkt mit dem größten Spaltenindex. Die Umlaufrichtung ist der Uhrzeigersinn. Der erste Punkt des Polygons ist gleich dem letzten. `get_region_convex` liefert die Koordinaten in Form von Tupeln ab. Eine leere Region wird als leeres Tupel übergeben.

Parameter

▷ **Region** (input_object) ..... region ⇝ *object*
Auszugebende Region.

▷ **Rows** (output_control) ..... contour.y-array ⇝ *integer*
Zeilennummern der Konturpunkte.

▷ **Columns** (output_control) ..... contour.x-array ⇝ *integer*
Spaltennummern der Konturpunkte.
Parameteranzahl: Columns == Rows

Ergebnis

`get_region_convex` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, skeleton, dyn_threshold

Nachfolger

disp_polygon

Alternativen

shape_trans

Siehe auch

select_obj, get_region_contour

Modul

Foundation

get_region_points (Region : : : Rows, Columns)

Zugriff auf die Punkte einer Region.

get_region_points gibt die Regionendaten in Form von Koordinatenlisten aus. Die Koordinaten sind in folgender Ordnung sortiert:

$$(r_1, c_1) \leq (r_2, c_2) := (r_1 < r_2) \quad \vee \quad (r_1 = r_2) \wedge (c1 \leq c2)$$

get_region_points liefert die Koordinaten in Form von Tupeln ab. Eine leere Region wird als leeres Tupel übergeben.

Achtung

Es darf nur eine Region übergeben werden.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *object*
  Auf diese Region wird zugegriffen.
- ▷ **Rows** (output_control) ..... coordinates.y-array ⇝ *integer*
  Zeilennummern der Punkte in der Region
- ▷ **Columns** (output_control) ..... coordinates.x-array ⇝ *integer*
  Spaltennummern der Punkte in der Region.
  Parameteranzahl: Columns == Rows

Ergebnis

get_region_points liefert normalerweise den Wert TRUE. Wird mehr als eine Region übergeben, wird ein Exception ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

sobel_amp, threshold, connection

Alternativen

get_region_runs

Siehe auch

copy_obj, gen_region_points

Modul

Foundation

get_region_polygon (Region : : Tolerance : Rows, Columns)

Polygonapproximation einer Region.

get_region_polygon berechnet ein Polygon, das den Rand einer Region annähern soll. Ein Polygon ist eine Folge von Zeilen- (Rows) und Spaltenkoordinaten (Columns), die die Kontur der Region beschreibt. Von dem Polygon werden nur die Stützpunkte ausgegeben. Der Parameter Tolerance gibt an, wie groß der maximale Abstand zwischen dem Polygon und dem Regionenrand sein darf. Hohlflächen der Region werden ignoriert. get_region_polygon liefert die Koordinaten in Form von Tupeln ab.

Achtung

Hohlflächen der Region werden ignoriert. Es darf nur eine Region übergeben werden und die muss genau eine Zusammenhangskomponente haben.

Parameter

▷ **Region** (input_object) ..... region ⇝ *object*
Zu approximierende Region.

▷ **Tolerance** (input_control) ..... number ⇝ *real* / integer
Maximaler Abstand zwischen dem Polygon und dem Regionenrand.
Defaultwert: 5.0
Wertevorschläge: Tolerance ∈ {0.0, 2.0, 5.0, 10.0}
Typischer Wertebereich: 0.0 ≤ Tolerance (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 1.0

▷ **Rows** (output_control) ..... polygon.y-array ⇝ *integer*
Zeilennummern der Stützpunkte der Kontur.

▷ **Columns** (output_control) ..... polygon.x-array ⇝ *integer*
Spaltennummern der Stützpunkte der Kontur.
Parameteranzahl: Columns == Rows

Ergebnis

get_region_polygon liefert normalerweise den Wert TRUE. Wird mehr als eine Zusammenhangskomponente übergeben, wird eine Fehlerbehandlung ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

sobel_amp, threshold, skeleton, edges_image

Siehe auch

copy_obj, gen_region_polygon, disp_polygon, get_region_contour, set_line_approx

Modul

Foundation

get_region_runs (Region : : : Row, ColumnBegin, ColumnEnd)

Zugriff auf die Lauflängenkodierung einer Region.

get_region_runs gibt die Regionendaten in Form von Sehnentupeln aus. Die Sehnendarstellung entsteht, indem man eine Region zeilenweise mit aufsteigender Zeilenzahl (= von „oben“ nach „unten“) untersucht. Jede Zeile wird von links nach rechts durchlaufen (aufsteigende Spaltenzahl). Dabei werden alle Anfangs- und Endpunkte von Regionenabschnitten (= Sehnen) gespeichert. Eine Region lässt sich somit durch eine Folge von Sehnen beschreiben, wobei eine Sehne durch Zeilennummer, Anfangs- und Endpunkte (Spaltennummer) definiert

ist. `get_region_runs` liefert die drei Komponenten der Sehnen in Form von Tupeln. Bei einer leeren Region werden drei leere Tupel zurückgegeben.

Achtung

Es darf nur eine Region übergeben werden.

Parameter

- ▷ **`Region`** (input_object) ..... region ⇝ *object*
  Auszugebende Region.
- ▷ **`Row`** (output_control) ..... chord.y-array ⇝ *integer*
  Zeilennummern der Sehnen.
- ▷ **`ColumnBegin`** (output_control) ..... chord.x1-array ⇝ *integer*
  Spaltennummern der Anfangspunkte der Sehnen.
  Parameteranzahl: ColumnBegin == Row
- ▷ **`ColumnEnd`** (output_control) ..... chord.x2-array ⇝ *integer*
  Spaltennummern der Endpunkte der Sehnen.
  Parameteranzahl: ColumnEnd == Row

Ergebnis

`get_region_runs` liefert normalerweise den Wert TRUE. Wird mehr als eine Region übergeben, wird eine Fehlerbehandlung ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`threshold`, `connection`

Alternativen

`get_region_points`

Siehe auch

`copy_obj`, `gen_region_runs`

Modul

Foundation

Kapitel 23

Segmentation

23.1 Flächenwachstum

```
expand_gray ( Regions,  Image,
    ForbiddenArea : RegionExpand : Iterations,  Mode,  Threshold : )
```

Auffüllen von „Lücken“ zwischen Regionen (abhängig vom Grau- oder Farbwert) oder Trennen überlappender Regionen.

`expand_gray` dient dazu, Lücken zwischen den Regionen der Eingabebilder, wie sie z.B. durch Unterdrückung zu kleiner Regionen nach einer Bildsegmentierung entstehen, zu schließen (Modus *'image'*) oder überlappende Regionen zu trennen (Modus *'region'*). Beide Effekte beruhen auf der Expansion von Regionen. Dabei wird bei jeder Iteration der Expansion die Bildpunkte innerhalb eines 1-Punkt breiten Streifens um die Region zu dieser hinzugefügt, deren Grau- bzw. Farbwerte sich von den Werten der benachbarten Randpunkte der Region um höchstens `Threshold` (je Kanal) unterscheiden. Bei Bildern vom Typ *'cyclic'* (z.B. Richtungsbilder) werden auch Bildpunkte mit einer Grauwertdifferenz von mindestens $255 - $ `Threshold` zur Region hinzugenommen.

Expandiert wird dabei nur in Bildbereiche, die nicht als „verbotene Bereiche“ (Parameter `ForbiddenArea`) ausgewiesen sind. Die Zahl der Iterationen wird mit dem Parameter `Iterations` festgelegt. Die Übergabe des strings *'maximal'* veranlasst bei `expand_gray`, das Verfahren so lange zu iterieren, bis es konvergiert, also keine Änderungen mehr auftreten. Eine 0 auf dieser Parameterposition bewirkt die Elimination der überlappenden Bildbereiche der Eingaberegionen. Im Detail unterscheiden sich die beiden Modi *'image'* und *'region'* wie folgt:

'image' Die Eingaberegionen werden iterativ so lange ausgedehnt, bis sie eine andere Region oder einen Bildrand berühren (bzw. das Wachstum aufgrund zu hoher Grauwertdifferenzen zum Stillstand kommt). Da `expand_gray_ref` alle Regionen simultan bearbeitet, werden die „Lücken“ zwischen den Regionen „gerecht“ auf Regionen mit ähnlichem Grau- oder Farbwert verteilt. Überlappende Regionen werden getrennt, indem ihre gemeinsamen Teilregionen (wiederum „gerecht“) auf sie verteilt werden.

'region' Es wird keine Expansion der Eingaberegionen durchgeführt, sondern es werden nur überlappende Bildregionen getrennt, indem die gemeinsamen Teilregionen „gerecht“ auf die bzgl. Grau- bzw. Farbwert passende Regionen aufgeteilt werden.

Achtung

Da Regionen nur in Bereiche mit passenden Grau- bzw. Farbwerten expandiert werden, bleiben im allgemeinen Lücken zwischen den Ausgaberegionen bestehen. Die Segmentierung der Bildebene ist also nicht total.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *object*
  Regionen, zwischen denen Lücken geschlossen oder die getrennt werden sollen.
- ▷ **`Image`** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic
  Bild (evtl. mehrkanalig) für Graustufen- bzw. Farbtest.
- ▷ **`ForbiddenArea`** (input_object) ..... region ⇝ *object*
  In diesen Bereich darf nicht expandiert werden.
- ▷ **`RegionExpand`** (output_object) ..... region(-array) ⇝ *object*
  Expandierte oder getrennte Regionen.

▷ **Iterations** (input_control) . . . . . string ⇝ *string* / integer
Zahl der Iterationen.
Defaultwert: 'maximal'
Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'maximal'}
Typischer Wertebereich: 1 ≤ Iterations ≤ 500 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Gewünschter Modus.
Defaultwert: 'image'
Werteliste: Mode ∈ {'image', 'region'}

▷ **Threshold** (input_control) . . . . . integer(-array) ⇝ *integer*
Maximalabweichung zwischen Randpixeln und Kandidaten für die Expansion.
Defaultwert: 32
Wertevorschläge: Threshold ∈ {5, 10, 15, 20, 25, 30, 40, 50}
Typischer Wertebereich: 1 ≤ Threshold ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  HImage   image (argv[1]);
  HRegion  empty_region;
  HWindow  win;

  image.Display (win);

  HRegionArray seg = (image >= 100).Connection ();

  seg.Display (win);
  HRegionArray exp = seg.ExpandGray1 (image, empty_region,
                                      "maximal", "image", 32);
  win.SetDraw ("margin");
  win.SetColored (12);
  exp.Display (win);
  win.Click ();

  return (0);
}
```

Ergebnis

expand_gray liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), und das bei leerer Ergebnisregion mit set_system('store_empty_region',<'2 (H_MSG_TRUE) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

connection, regiongrowing, pouring, class_ndim_norm

Nachfolger

select_shape

Siehe auch

expand_gray_ref, expand_region

Modul

Foundation

```
expand_gray_ref ( Regions, Image,
    ForbiddenArea : RegionExpand : Iterations, Mode, RefGray,
    Threshold : )
```

Auffüllen von „Lücken“ zwischen Regionen (abhängig vom Grau- oder Farbwert) oder Trennen überlappender Regionen.

expand_gray_ref dient dazu, Lücken zwischen den Eingaberegionen, wie sie z.B. durch Unterdrückung zu kleiner Regionen nach einer Bildsegmentierung entstehen, zu schließen (Modus *'image'*) oder überlappende Regionen zu trennen (Modus *'region'*). Beide Effekte beruhen auf der Expansion von Regionen. Dabei werden bei jeder Iteration der Expansion die Bildpunkte innerhalb eines 1-Punkt breiten Streifens um die Region zu dieser hinzugefügt, deren Grau- bzw. Farbwerte sich vom übergebenen Referenzgrauwert der Region um höchstens Threshold (je Kanal) unterscheiden. Bei Bildern vom Typ *'cyclic'* (z.B. Richtungsbilder) werden auch Bildpunkte mit einer Grauwertdifferenz von mindestens $255 - \texttt{Threshold}$ zur Region hinzugenommen.

Expandiert wird dabei nur in Bildbereiche, die nicht als „verbotene Bereiche“ (Parameter ForbiddenArea) ausgewiesen sind. Die Zahl der Iterationen wird mit dem Parameter Iterations festgelegt. Die Übergabe des strings *'maximal'* veranlasst bei expand_gray_ref, das Verfahren so lange zu iterieren, bis es konvergiert, also keine Änderungen mehr auftreten. Eine 0 auf dieser Parameterposition bewirkt die Ausgabe der nichtüberlappenden Teilregionen. Im Detail unterscheiden sich die beiden Modi *'image'* und *'region'* wie folgt:

'image' Die Eingaberegionen werden iterativ so lange ausgedehnt, bis sie eine andere Region oder einen Bildrand berühren (bzw. das Wachstum aufgrund zu hoher Grauwertdifferenzen zum Stillstand kommt). Da expand_gray_ref alle Regionen simultan bearbeitet, werden die „Lücken“ zwischen den Regionen „gerecht“ auf Regionen mit ähnlichem Grau- oder Farbwert verteilt. Überlappende Regionen werden getrennt, indem ihre gemeinsamen Teilregionen (wiederum „gerecht“) auf sie verteilt werden.

'region' Es wird keine Expansion der Eingaberegionen durchgeführt, sondern es werden nur überlappende Bildregionen getrennt, indem die gemeinsamen Teilregionen „gerecht“ auf die bzgl. Grau- bzw. Farbwert passende Regionen aufgeteilt werden.

Achtung

Da Regionen nur in Bereiche mit passenden Grau- bzw. Farbwerten expandiert werden, bleiben im allgemeinen Lücken zwischen den Ausgaberegionen bestehen. Die Segmentierung der Bildebene ist also nicht total.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
  Regionen, zwischen denen Lücken geschlossen oder die getrennt werden sollen.
- ▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic
  Bild (evtl. mehrkanalig) für Graustufen- bzw. Farbtest.
- ▷ **ForbiddenArea** (input_object) ..... region ⇝ *object*
  In diesen Bereich darf nicht expandiert werden.
- ▷ **RegionExpand** (output_object) ..... region(-array) ⇝ *object*
  Expandierte oder getrennte Regionen.

▷ **Iterations** (input_control) . . . . . string ⇝ *string* / integer
Zahl der Iterationen.
Defaultwert: 'maximal'
Wertevorschläge: Iterations ∈ {'maximal', 1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 70, 100, 150, 200, 300, 500}
Typischer Wertebereich: 1 ≤ Iterations ≤ 500 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Gewünschter Modus.
Defaultwert: 'image'
Werteliste: Mode ∈ {'image', 'region'}

▷ **RefGray** (input_control) . . . . . integer(-array) ⇝ *integer*
Referenzgrauwert.
Defaultwert: 128
Wertevorschläge: RefGray ∈ {1, 10, 20, 50, 100, 128, 200, 255}
Typischer Wertebereich: 1 ≤ RefGray ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10

▷ **Threshold** (input_control) . . . . . integer(-array) ⇝ *integer*
Maximalabweichung zwischen Referenzgrauwert und Kandidaten für die Expansion.
Defaultwert: 32
Wertevorschläge: Threshold ∈ {4, 10, 15, 20, 25, 30, 40}
Typischer Wertebereich: 1 ≤ Threshold ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  HImage   image (argv[1]);
  HRegion  empty_region;
  HWindow  win;

  win.SetDraw ("margin");
  win.SetColored (12);

  image.Display (win);

  HRegionArray seg = (image >= 100).Connection ();
  seg.Display (win);

  Tuple iter = "maximal";
  Tuple mode = "image";
  Tuple refg = 128;
  Tuple thrs = 32;

  HRegionArray exp = seg.ExpandGrayRef (image, empty_region,
                                        iter, mode, refg, thrs);
  exp.Display (win);
  win.Click ();
```

```
  return (0);
}
```

Ergebnis

`expand_gray_ref` liefert normalerweise den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)`, und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<'2 (H_MSG_TRUE)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`connection`, `regiongrowing`, `pouring`, `class_ndim_norm`

Nachfolger

`select_shape`

Siehe auch

`expand_gray`, `expand_region`

Modul

Foundation

regiongrowing (Image : Regions : RasterHeight, RasterWidth, Tolerance, MinSize :)

Segmentieren mit Hilfe von Flächenwachstum.

`regiongrowing` liefert Bilder mit „gleicher“ Intensität - gerastert in Rechtecken der Größe `RasterHeight` × `RasterWidth`. Zur Entscheidung, ob benachbarte Rasterfelder zum selben Bild gehören oder nicht, wird nur die Differenz der Grauwerte ihrer Zentren herangezogen. Ist die Grauwertdifferenz kleiner oder gleich `Tolerance`, werden die Rasterfelder zu einer Region verschmolzen.

Sind g_1 und g_2 zwei zu untersuchende Grauwerte, dann gehören sie zur gleichen Region, falls:

$$|g_1 - g_2| < \texttt{Tolerance}$$

Bei Pixeln vom Typ cyclic wird folgende Formel verwendet:

$$(|g_1 - g_2| < \texttt{Tolerance}) \ \wedge \ (|g_1 - g_2| \leq 127)$$
$$(256 - |g_1 - g_2| < \texttt{Tolerance}) \ \wedge \ (|g_1 - g_2| > 127)$$

Bei Schrittweiten größer als 1 ist ein typisches Vorgehen, die Eingabeobjekte vor dem Aufruf von `regiongrowing` mit einem Tiefpassfilter der Größe `RasterHeight` × `RasterWidth` zu glätten (damit die ausgewerteten Schwerpunkte der Rasterfelder auch wirklich repräsentativ für das gesamte Rasterfeld sind). Ist das Bildmaterial nicht stark verrauscht und die Schrittweite klein, dann kann die vorherige Glättung in vielen Fällen auch entfallen.

In den Ergebnisbildern werden bei Schrittweiten größer als 1 nicht nur die untersuchten Punkte gesetzt, sondern es wird um jeden Punkt ein Rechteck der Größe `RasterHeight` × `RasterWidth` gesetzt. Es werden nur Regionen, die mindestens `MinSize` Punkte umfassen, ausgegeben.

Regiongrowing ist ein sehr schnelles Verfahren und ist daher für zeitkritische Anwendungen zu empfehlen.

Achtung

`RasterWidth` und `RasterHeight` werden automatisch auf ungerade Werte konvertiert.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / int4 / real
Eingabebild.

▷ **Regions** (output_object) ..... region-array ⇝ *object*
Segmentierte Regionen.

▷ **RasterHeight** (input_control) ..... extent.y ⇝ *integer*
Vertikaler Abstand zwischen den Testpunkten (Höhe des Rasters).
Defaultwert: 3
Wertevorschläge: `RasterHeight` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21}
Typischer Wertebereich: 1 ≤ `RasterHeight` ≤ 99 (lin)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: RasterHeight >= 1 && odd(RasterHeight)

▷ **RasterWidth** (input_control) ..... extent.x ⇝ *integer*
Horizontaler Abstand zwischen den Testpunkten (Breite des Rasters).
Defaultwert: 3
Wertevorschläge: `RasterWidth` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21}
Typischer Wertebereich: 1 ≤ `RasterWidth` ≤ 99 (lin)
Minimale Schrittweite: 2
Empfohlene Schrittweite: 2
Restriktion: RasterWidth >= 1 && odd(RasterWidth)

▷ **Tolerance** (input_control) ..... number ⇝ *real* / integer
Punkte mit Grauwertdifferenz kleiner oder gleich Tolerance werden zum selben Objekt gezählt.
Defaultwert: 6.0
Wertevorschläge: `Tolerance` ∈ {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 18.0, 25.0}
Typischer Wertebereich: 1.0 ≤ `Tolerance` ≤ 127.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 1.0
Restriktion: 0 <= Tolerance && Tolerance < 127

▷ **MinSize** (input_control) ..... integer ⇝ *integer*
Mindestgröße der Ausgaberegionen.
Defaultwert: 100
Wertevorschläge: `MinSize` ∈ {1, 5, 10, 20, 50, 100, 200, 500, 1000}
Typischer Wertebereich: 1 ≤ MinSize
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5
Restriktion: MinSize >= 1

Beispiel

```
read_image(Image,'fabrik')
mean_image(Image,Mean,RasterHeight,RasterWidth)
regiongrowing(Mean,Result,RasterHeight,RasterWidth,6.0,100)
```

Komplexität

Sei N die Anzahl der gefundenen Regionen und M die Anzahl der untersuchten Punkte in einer solchen Region, dann ist die Laufzeitkomplexität $O(N * log(M) * M)$.

Ergebnis

`regiongrowing` liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.

Vorgänger

binomial_filter, mean_image, gauss_filter, smooth_image, median_image, anisotropic_diffusion

Nachfolger

select_shape, reduce_domain, select_gray

Alternativen

regiongrowing_n, regiongrowing_mean, label_to_region

Modul

Foundation

regiongrowing_mean (Image : Regions : StartRows, StartColumns, Tolerance, MinSize :)

Ausführen eines Flächenwachstums mit Mittelwertbildung.

regiongrowing_mean führt ein Flächenwachstum mit Mittelwertbildung ausgehend von vorgegebenen Startpunkten aus. StartRows und StartColumns geben die Startpunkte vor, von denen aus Regionen erzeugt werden. Zu jedem Zeitpunkt des Wachstumsprozesses ist der Mittelwert der bisherigen Region bekannt. Die Grauwerte der Randpunkte der Region werden mit diesem Wert verglichen und zu der Region hinzugefügt, falls der Grauwert nur wenig abweicht (Differenz $<$ Tolerance). Regionen die zu klein sind (Fläche $<$ MinSize) werden unterdrückt.

Werden keine Startpunkte vorgegeben (leere Tupel), dann beginnt der Expansionsprozess bei dem ersten Bildpunkt (links oben) und wird nach jeder neu entstandenen Region mit dem ersten, bisher unbearbeiteten Bildpunkt, fortgesetzt.

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4
Eingabebild.

▷ **Regions** (output_object) . . . . . region-array ⇝ *object*
Segmentierte Regionen.

▷ **StartRows** (input_control) . . . . . point.y(-array) ⇝ *integer*
Zeilenkoordinaten der Startpunkte.
Defaultwert: []
Typischer Wertebereich: 0 ≤ StartRows
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **StartColumns** (input_control) . . . . . point.x(-array) ⇝ *integer*
Spaltenkoordinaten der Startpunkte.
Defaultwert: []
Typischer Wertebereich: 0 ≤ StartColumns
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Tolerance** (input_control) . . . . . number ⇝ *real*
Maximale Abweichung vom Mittelwert.
Defaultwert: 5.0
Wertevorschläge: Tolerance ∈ {0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 15.0, 17.0, 20.0, 25.0, 30.0, 40.0}
Restriktion: Tolerance > 0.0

▷ **MinSize** (input_control) . . . . . integer ⇝ *integer*
Mindestgröße einer Region.
Defaultwert: 100
Wertevorschläge: MinSize ∈ {0, 10, 30, 50, 100, 500, 1000, 2000}
Typischer Wertebereich: 0 ≤ MinSize
Minimale Schrittweite: 1
Empfohlene Schrittweite: 100
Restriktion: MinSize >= 0

Ergebnis

`regiongrowing_mean` liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`binomial_filter`, `gauss_filter`, `sigma_image`, `anisotropic_diffusion`, `median_image`, `mean_image`

Nachfolger

`select_shape`, `reduce_domain`, `opening`, `expand_region`

Alternativen

`regiongrowing`, `regiongrowing_n`

Modul

Foundation

`regiongrowing_n` `( MultiChannelImage : Regions : Metric, MinTolerance, MaxTolerance, MinSize : )`

Ausführen eines Flächenwachstumverfahrens für mehrkanalige Bilder.

`regiongrowing_n` führt ein mehrkanaliges Flächenwachstum durch. Die n Kanäle liefern in jedem Bildpunkt einen n-dimensionalen Merkmalsvektor. Benachbarte Punkte werden zu einer Ausgaberegion zusammengefasst, wenn die Differenz ihrer Merkmalsvektoren bezüglich der vorgegebenen Metrik im Intervall [`MinTolerance`, `MaxTolerance`] liegen. Untersucht werden direkte Nachbarn in der 4-er Nachbarschaft. Dabei stehen folgende Metriken zur Verfügung:

g_A bezeichne einen Grauwert im Vektor A, dem Merkmalsvektor in einem Punkt a, g_B die entsprechenden Grauwerte im Vektor B, dem Merkmalsvektor in einem Nachbarpunkt b. $g(d)$ ist der Grauwert mit Index d. $MinT$ steht für `MinTolerance`, $MaxT$ für `MaxTolerance`.

'1-norm': Betragssumme

$$MinT \leq \frac{1}{n} \sum |g_A - g_B| \leq MaxT$$

'2-norm': Euklidischer Abstand

$$MinT \leq \sqrt{\frac{\sum (g_A - g_B)^2}{n}} \leq MaxT$$

'3-norm': p - Norm mit p = 3

$$MinT \leq \sqrt[3]{\frac{\sum (g_A - g_B)^3}{n}} \leq MaxT$$

'4-norm': p - Norm mit p = 4

$$MinT \leq \sqrt[4]{\frac{\sum (g_A - g_B)^4}{n}} \leq MaxT$$

'n-norm': Minkowsky-Distanz

$$MinT \leq \sqrt[n]{\frac{\sum (g_A - g_B)^n}{n}} \leq MaxT$$

'max-diff': Supremumsdistanz

$$MinT \leq \max \{|g_A - g_B|\} \leq MaxT$$

'min-diff': Infimumsdistanz

$$MinT \leq \min\{|g_A - g_B|\} \leq MaxT$$

'variance': Varianz der Grauwertdifferenzen

$$MinT \leq Var(g_A - g_B) \leq MaxT$$

'dot-product': Skalarprodukt

$$MinT \leq \frac{1}{n}\sqrt{\sum (g_A g_B)} \leq MaxT$$

'correlation': Korrelation

$$m_A = \frac{1}{n}\sum g_A$$

$$Var_A = \frac{1}{n}\sqrt{\sum (g_A - m_A)^2}$$

$$m_B = \frac{1}{n}\sum g_B$$

$$Var_B = \frac{1}{n}\sqrt{\sum (g_B - m_B)^2}$$

$$MinT \leq \frac{1}{n^2}\sum \frac{(g_A - m_A)(g_B - m_B)}{(Var_A Var_B)} \leq MaxT$$

'mean-diff': Differenz der arithmetischen Mittel

$$a = \frac{1}{n}\sum g_A$$

$$b = \frac{1}{n}\sum g_B$$

$$MinT \leq |a - b| \leq MaxT$$

'mean-ratio': Quotient der arithmetischen Mittel

$$a = \frac{1}{n}\sum g_A$$

$$b = \frac{1}{n}\sum g_B$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'length-diff': Differenz der Vektorlängen

$$a = \sqrt{\frac{\sum g_A^2}{n}}$$

$$b = \sqrt{\frac{\sum g_B^2}{n}}$$

$$MinT \leq |a - b| \leq MaxT$$

'length-ratio': Quotient der Vektorlängen

$$a = \sqrt{\frac{\sum g_A^2}{n}}$$

$$b = \sqrt{\frac{\sum g_B^2}{n}}$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'n-norm-ratio': Quotient der Vektorlängen bzgl. der p-Norm mit $p = n$

$$a = \sqrt[n]{\frac{\sum g_A^n}{n}}$$

$$b = \sqrt[n]{\frac{\sum g_B^n}{n}}$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'gray-max-diff': Differenz der Maxima

$$a = \max\{|g_A|\}$$

$$b = \max\{|g_B|\}$$

$$MinT \leq |a - b| \leq MaxT$$

'gray-max-ratio': Quotient der Maxima

$$a = \max\{|g_A|\}$$

$$b = \max\{|g_B|\}$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'gray-min-diff': Differenz der Minima

$$a = \min\{|g_A|\}$$

$$b = \min\{|g_B|\}$$

$$MinT \leq |a - b| \leq MaxT$$

'gray-min-ratio': Quotient der Minima

$$a = \min\{|g_A|\}$$

$$b = \min\{|g_B|\}$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'variance-diff': Differenz der Varianzen über allen Grauwerten (Kanälen)

$$MinT \leq |Var(g_A) - Var(g_B)| \leq MaxT$$

'variance-ratio': Quotient der Varianzen über allen Grauwerten (Kanälen)

$$MinT \leq \frac{Var(g_B)}{Var(g_A)} \leq MaxT$$

'mean-abs-diff': Differenz der Betragssummen über allen Grauwerten (Kanälen)

$$a = \sum_{d,k,k<d} |g_A(d) - g_A(k)|$$

$$b = \sum_{d,k,k<d} |g_B(d) - g_B(k)|$$

$$MinT \leq \frac{|a - b|}{\text{Anzahl der Summen}} \leq MaxT$$

'mean-abs-ratio': Quotient der Betragssummen über allen Grauwerten (Kanälen) (channels)

$$a = \sum_{d,k,k<d} |g_A(d) - g_A(k)|$$

$$b = \sum_{d,k,k<d} |g_B(d) - g_B(k)|$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'max-abs-diff': Differenz der maximalen Abstände der Komponenten

$$a = \max\{g_A(d), g_A(k)\}$$

$$b = \max\{g_B(d), g_B(k)\}$$

$$MinT \leq |a - b| \leq MaxT$$

'max-abs-ratio': Quotient der maximalen Abstände der Komponenten

$$a = \max\{g_A(d), g_A(k)\}$$

$$b = \max\{g_B(d), g_B(k)\}$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'min-abs-diff': Differenz der minimalen Abstände der Komponenten

$$a = \min\{g_A(d), g_A(k)\}, k < d$$

$$b = \min\{g_B(d), g_B(k)\}, k < d$$

$$MinT \leq |a - b| \leq MaxT$$

'min-abs-ratio': Quotient der minimalen Abstände der Komponenten

$$a = \min\{g_A(d), g_A(k)\}, k < d$$

$$b = \min\{g_B(d), g_B(k)\}, k < d$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'plane': Für alle d_1,$d_2 \in [1, n]$ muss gelten:

$$g_A(d_1) > g_A(d_2) \Rightarrow g_B(d_1) > g_B(d_2)$$

$$g_A(d_1) < g_A(d_2) \Rightarrow g_B(d_1) < g_B(d_2)$$

Regionen mit einer Fläche geringer als `MinSize` werden unterdrückt.

Parameter

- ▷ **`MultiChannelImage`** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Eingabebild.
- ▷ **`Regions`** (output_object) ..... region-array ⇝ *object*
  Segmentierte Regionen.
- ▷ **`Metric`** (input_control) ..... string ⇝ *string*
  Metrik für Distanz der Merkmalsvektoren.
  Defaultwert: '2-norm'
  Werteliste: `Metric` ∈ {'1-norm', '2-norm', '3-norm', '4-norm', 'n-norm', 'max-diff', 'min-diff', 'variance', 'dot-product', 'correlation', 'mean-diff', 'mean-ratio', 'length-diff', 'length-ratio', 'n-norm-ratio', 'gray-max-diff', 'gray-max-ratio', 'gray-min-diff', 'gray-min-ratio', 'variance-diff', 'variance-ratio', 'mean-abs-diff', 'mean-abs-ratio', 'max-abs-diff', 'max-abs-ratio', 'min-abs-diff', 'min-abs-ratio', 'plane'}
- ▷ **`MinTolerance`** (input_control) ..... number ⇝ *real* / integer
  Untere Schwelle für den Merkmalsabstand.
  Defaultwert: 0.0
  Wertevorschläge: `MinTolerance` ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0, 25.0, 30.0}
- ▷ **`MaxTolerance`** (input_control) ..... number ⇝ *real* / integer
  Obere Schwelle für den Merkmalsabstand.
  Defaultwert: 20.0
  Wertevorschläge: `MaxTolerance` ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0, 25.0, 30.0}

▷ **MinSize** (input_control) . . . . . integer ⇝ *integer*
Mindestgröße der Ausgaberegionen.
Defaultwert: 30
Wertevorschläge: MinSize ∈ {1, 10, 25, 50, 100, 200, 500, 1000}
Typischer Wertebereich: 1 ≤ MinSize
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5

Ergebnis

regiongrowing_n liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe set_system). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

compose2, compose3

Alternativen

class_2dim_sup, class_ndim_norm

Siehe auch

regiongrowing_mean

Modul

Foundation

23.2 Kanten

```
detect_edge_segments ( Image : : SobelSize, MinAmplitude,
    MaxDistance, MinLength : BeginRow, BeginCol, EndRow, EndCol )
```

Detektion gerader Kantenstücke.

detect_edge_segments detektiert gerade Kantenstücke in einem Graustufenbild. Die gefundenen Kantensegmente werden als Strecken Linien) mit einem Anfangs-(BeginRow,BeginCol) und einem Endpunkt (EndRow,EndCol) zurückgeliefert. Die Kantendetektion basiert auf dem Sobelfilter (Variante *'sum_abs'*, Filtergröße, gemäß SobelSize). Auf dessen Filterantwort bezieht sich auch der Parameter MinAmplitude: Nur Pixel mit Intensität größer oder gleich MinAmplitude im gefilterten Eingangsbild werden als Kandidaten für Kantenpunkte verwendet. Die Rohkanten werden verdünnt und in gerade Teilstücke zerlegt. Aus technischen Gründen gehen dabei Kantenpunkte, in denen sich mehrere Kanten treffen, verloren. Somit liefert detect_edge_segments im allgemeinen keine geschlossenen Objektkonturen (bzw. lineare Approximationen daran). Der Parameter MaxDistance steuert die Zerlegung von Kanten in gerade Teilstücke. Er gibt die maximale Abweichung eines Kantenpunktes von der approximierenden Geraden an. Aus Laufzeitgründen wird dabei nicht der euklidische Abstand, sondern die Betragssumme der Koordinatendifferenzen verwendet. MinLength gibt die gewünschte Mindestlänge der Ausgabelinien an. Kürzere Linien werden unterdrückt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *object : byte*
Eingabebild.

▷ **SobelSize** (input_control) . . . . . integer ⇝ *integer*
Filtergröße für Sobeloperator.
Defaultwert: 5
Werteliste: SobelSize ∈ {3, 5, 7, 9, 11, 13}

▷ **MinAmplitude** (input_control) ..... integer ⇝ *integer*
Mindestamplitude für Kanten.
Defaultwert: 32
Wertevorschläge: MinAmplitude ∈ {10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 80, 90, 100, 110}
Typischer Wertebereich: 1 ≤ MinAmplitude ≤ 255
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: MinAmplitude >= 0

▷ **MaxDistance** (input_control) ..... integer ⇝ *integer*
Maximale Abweichung der approximierenden Geraden von der ursprünglichen Kante.
Defaultwert: 3
Wertevorschläge: MaxDistance ∈ {2, 3, 4, 5, 6, 7, 8}
Typischer Wertebereich: 1 ≤ MaxDistance ≤ 30
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: MaxDistance >= 0

▷ **MinLength** (input_control) ..... integer ⇝ *integer*
Mindestlänge der Ausgabelinien.
Defaultwert: 10
Wertevorschläge: MinLength ∈ {3, 5, 7, 9, 11, 13, 16, 20}
Typischer Wertebereich: 1 ≤ MinLength ≤ 500
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: MinLength >= 0

▷ **BeginRow** (output_control) ..... line.begin.y-array ⇝ *integer*
Zeilenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **BeginCol** (output_control) ..... line.begin.x-array ⇝ *integer*
Spaltenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **EndRow** (output_control) ..... line.end.y-array ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Ausgabelinien.

▷ **EndCol** (output_control) ..... line.end.x-array ⇝ *integer*
Spaltenkoordinaten der Endpunkte der Ausgabelinien.

Beispiel

```
Htuple  SobelSize,MinAmplitude,MaxDistance,MinLength;
Htuple  RowBegin,ColBegin,RowEnd,ColEnd;

create_tuple(&SobelSize,1);
set_i(SobelSize,5,0);
create_tuple(&MinAmplitude,1);
set_i(MinAmplitude,32,0);
create_tuple(&MaxDistance,1);
set_i(MaxDistance,3,0);
create_tuple(&MinLength,1);
set_i(MinLength,10,0);
T_detect_edge_segments(Image,SobelSize,MinAmplitude,MaxDistance,MinLength,
                       &RowBegin,&ColBegin,&RowEnd,&ColEnd);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert detect_edge_segments den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

sigma_image, median_image

Nachfolger

partition_lines, line_position, line_orientation

Alternativen

sobel_amp, threshold, skeleton

Modul

Foundation

hysteresis_threshold (Image : RegionHysteresis : Low, High, MaxLength :)

Segmentieren von Bildern mittels Hysterese-Schwellenwertoperation.

hysteresis_threshold führt eine Schwellenwertoperation mit Hysterese (nach Canny) durch. Dabei werden alle Punkte im Eingabebild Image größer oder gleich der oberen Schwelle High sofort in die Ausgaberegion(en) übernommen („sichere“ Punkte). Umgekehrt werden alle Punkte mit Grauwerten echt kleiner als die untere Schwelle Low zurückgewiesen. „Potentielle“ Punkte mit Grauwerten zwischen den beiden Schwellen schließlich werden dann übernommen, wenn sie durch einen Pfad mit maximaler Länge MaxLength von „potentiellen“ Punkten mit einem „sicheren“ Punkt verbunden sind. Die „sicheren“ Punkte strahlen also auf ihre Umgebung aus, sie „wirken nach“ (Hysterese).

Achtung

Bei Bildern vom Typ byte, uint2 oder int4 muss die untere Schwelle Low > 0 sein.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / int4 / real
  Eingabebild.
- ▷ **RegionHysteresis** (output_object) ..... region(-array) ⇝ *object*
  segmentierte Region.
- ▷ **Low** (input_control) ..... number ⇝ *integer* / real
  Untere Schwelle für die Grauwerte.
  Defaultwert: 30
  Wertevorschläge: Low ∈ {5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
- ▷ **High** (input_control) ..... number ⇝ *integer* / real
  Obere Schwelle für die Grauwerte.
  Defaultwert: 60
  Wertevorschläge: High ∈ {5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130}
  Restriktion: High >= Low
- ▷ **MaxLength** (input_control) ..... integer ⇝ *integer*
  Maximale Länge eines Pfades „potentieller“ Punkte zu einem „sicheren“ Punkt hin.
  Defaultwert: 10
  Wertevorschläge: MaxLength ∈ {1, 2, 3, 5, 7, 10, 12, 14, 17, 20, 25, 30, 35, 40, 50}
  Typischer Wertebereich: 1 ≤ MaxLength
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 5
  Restriktion: MaxLength > 1

Ergebnis

hysteresis_threshold liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe set_system). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

dyn_threshold, threshold, class_2dim_sup, fast_threshold

Siehe auch

edges_image, sobel_dir, background_seg

Literatur

J. Canny, „Finding Edges and Lines in Images“; Report, AI-TR-720, M.I.T. Artificial Intelligence Lab., Cambridge, MA, 1983.

Modul

Foundation

nonmax_suppression_amp (ImgAmp : ImageResult : Mode :)

Unterdrücken von nicht maximalen Punkten auf einer Kante.

nonmax_suppression_amp unterdrückt Punkte aus dem Definitionsbereich von ImgAmp, deren Grauwerte keine (gerichteten) Maxima sind. Im Gegensatz zu nonmax_suppression_dir wird dabei kein explizites Richtungsbild benötigt. Es stehen zwei Modi zur Verfügung:

'hvnms' Ein Punkt gilt hier als Maximum, wenn sein Grauwert entweder horizontal oder vertikal in einem Suchbereich von $\pm$ 5 Pixeln größer oder gleich den Grauwerten seiner Nachbarpunkte ist. Nicht-Maximum Punkte werden aus der Region entfernt, die Grauwerte bleiben unverändert.

'loc_max' Ein Punkt gilt hier als Maximum, wenn sein Grauwert größer oder gleich den Grauwerten seiner acht Nachbarpunkte ist.

Parameter

▷ **ImgAmp** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
Amplitudenbild.

▷ **ImageResult** (output_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
Bild mit verdünnten Kantenregionen.

▷ **Mode** (input_control) ..... string ⇝ *string*
Horizontale/vertikale NMS oder ungerichtete NMS.
Defaultwert: 'hvnms'
Werteliste: Mode ∈ {'hvnms', 'loc_max'}

Ergebnis

nonmax_suppression_amp liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe set_system). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

sobel_amp

Nachfolger

threshold, hysteresis_threshold

Alternativen

local_max, nonmax_suppression_dir

Siehe auch

skeleton

Literatur

S.Lanser: „Detektion von Stufenkanten mittels rekursiver Filter nach Deriche“; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

J.Canny: „Finding Edges and Lines in Images“; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge, MA; 1983.

Modul

Foundation

nonmax_suppression_dir (ImgAmp, ImgDir : ImageResult : Mode :)

Unterdrücken von nicht maximalen Punkten auf einer Kante unter Verwendung eines Richtungsbildes.

nonmax_suppression_dir unterdrückt Punkte aus den Bildregionen von ImgAmp, deren Grauwerte keine (gerichteten) Maxima sind. ImgDir ist dabei ein Richtungsbild (Einheit: 2 Grad, d.h. 50 Grad sind z.B. mit Wert 25 codiert), wie es z.B. von edges_image geliefert wird. Es stehen zwei Modi zur Verfügung:

'nms' Für jeden Bildpunkt wird getestet, ob sein Grauwert senkrecht zu seiner Richtung maximal ist. Im Modus *'nms'* werden dazu nur die beiden der Senkrechten am nächsten liegenden Nachbarpunkte für die Entscheidung herangezogen. Ist der Grauwert eines dieser Nachbarn echt größer als der Grauwert des zu testenden Punktes, wird dieser unterdrückt (d.h. aus der Region entfernt, die Grauwerte bleiben unverändert).

'inms' Wie *'nms'*, die beiden benötigten Vergleichswerte werden jedoch durch Interpolation der Grauwerte von vier Nachbarpunkten gewonnen.

Parameter

- ▷ **ImgAmp** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Amplitudenbild.
- ▷ **ImgDir** (input_object) ..... singlechannelimage(-array) ⇝ *object :* direction
  Richtungsbild.
- ▷ **ImageResult** (output_object) ..... image(-array) ⇝ *object :* byte / uint2 / real
  Bild mit verdünnten Kantenregionen.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Non-Maximum Suppression, interpolierende NMS.
  Defaultwert: 'nms'
  Werteliste: Mode ∈ {'nms', 'inms'}

Ergebnis

nonmax_suppression_dir liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe set_system). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

edges_image, sobel_dir, frei_dir

Nachfolger

threshold, hysteresis_threshold

Alternativen

nonmax_suppression_amp

Siehe auch

skeleton

Literatur

S.Lanser: „Detektion von Stufenkanten mittels rekursiver Filter nach Deriche“; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

J.Canny: „Finding Edges and Lines in Images“; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge; 1983.

Modul

Foundation

23.3 Klassifikation

add_samples_image_class_gmm (Image, ClassRegions : : GMMHandle, Randomize :)

Hinzufügen von Trainingsmustern aus einem Bild zu den Trainingsdaten eines Gaussian Mixture Models.

add_samples_image_class_gmm fügt Trainingsmuster aus dem Bild Image zu dem durch GMMHandle gegebenen Gaussian Mixture Model (GMM) hinzu. add_samples_image_class_gmm wird dazu verwendet, die Trainingsmuster bereitzustellen, um einen Klassifikator zur Pixelklassifikation von mehrkanaligen Bildern mit classify_image_class_gmm zu trainieren. Die Funktionsweise von add_samples_image_class_gmm ist analog zu add_sample_class_gmm. Das Bild Image muss so viele Kanäle besitzen, wie mit NumDim in create_class_gmm festgelegt worden ist. In ClassRegions werden die Trainingsgebiete für die NumClasses Pixelklassen angegeben. ClassRegions muss also ein Tupel mit NumClasses Regionen beinhalten. Die Reihenfolge der Regionen in ClassRegions legt die Klassen der Pixel fest. Falls in Image eine zu trainierende Klasse nicht vorkommt, muss an der entsprechenden Stelle des Tupels eine leere Region übergeben werden. Durch diesen Mechanismus können mehrere Bilder verwendet werden, um mit mehrmaligem Aufruf von add_samples_image_class_gmm Trainingsmuster für alle relevanten Pixelklassen bereitzustellen. Die Regionen in ClassRegions sollten repräsentative Trainingsmuster für die jeweiligen Klassen enthalten. Sie brauchen deshalb das Eingabebild nicht zu überdecken. Die Regionen in ClassRegions sollten sich nicht überlappen, da dies dazu führen würde, dass in den Trainingsdaten die Merkmalsvektoren in den Überlappungsbereichen mehreren Klassen zugeordnet wären, was zu einer schlechteren Klassifikationsleistung führen kann. Ganzzahlige Bilddaten können ungeeignet für eine Modellierung mit GMMs sein. Randomize kann verwendet werden, um dieses Problem zu umgehen. Dies ist im Detail bei add_sample_class_gmm erklärt.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Trainingsbild.

▷ **ClassRegions** (input_object) ..... region-array ⇝ *object*
Regionen der zu trainierenden Klassen.

▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
Handle des GMM.

▷ **Randomize** (input_control) ..... real ⇝ *real*
Standardabweichung des Gaußschen Rauschens, das zu den Trainingsdaten hinzugefügt wird.
Defaultwert: 0.0
Wertevorschläge: Randomize ∈ {0.0, 1.5, 2.0}
Restriktion: Randomize >= 0.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_samples_image_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- GMMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_gmm`

Nachfolger

`train_class_gmm`, `write_samples_class_gmm`

Alternativen

`read_samples_class_gmm`

Siehe auch

`classify_image_class_gmm`, `add_sample_class_gmm`, `clear_samples_class_gmm`, `get_sample_num_class_gmm`, `get_sample_class_gmm`

Modul

Foundation

add_samples_image_class_knn (Image, ClassRegions : : KNNHandle :)

Hinzufügen von Trainingsmustern aus einem Bild zu den Trainingsdaten eines k-Nearest-Neighbor-Klassifikators.

`add_samples_image_class_knn` fügt Trainingsmuster aus dem Bild `Image` zu dem durch `KNNHandle` gegebenen k-Nearest-Neighbor-Klassifikator (k-NN) hinzu. `add_samples_image_class_knn` wird dazu verwendet, die Trainingsmuster bereitzustellen, um einen Klassifikator zur Pixelklassifikation von mehrkanaligen Bildern mit `classify_image_class_knn` zu trainieren. Die Funktionsweise von `add_samples_image_class_knn` ist analog zu `add_sample_class_knn`. Da der k-NN hier aber immer zur Klassifikation verwendet wird, muss bei `create_class_knn` die korrekte Anzahl an Kanälen angegeben worden sein. Das Bild `Image` muss so viele Kanäle besitzen, wie mit `NumDim` in `create_class_knn` festgelegt worden ist. In `ClassRegions` werden die Trainingsgebiete für die Pixelklassen angegeben. `ClassRegions` muss also ein Tupel mit mindestens 2 Regionen beinhalten. Die Reihenfolge der Regionen in `ClassRegions` legt die Klassen der Pixel fest. Falls in `Image` eine zu trainierende Klasse nicht vorkommt, muss an der entsprechenden Stelle des Tupels eine leere Region übergeben werden. Durch diesen Mechanismus können mehrere Bilder verwendet werden, um mit mehrmaligem Aufruf von `add_samples_image_class_knn` Trainingsmuster für alle relevanten Pixelklassen bereitzustellen. Die Regionen in `ClassRegions` sollten repräsentative Trainingsmuster für die jeweiligen Klassen enthalten. Sie brauchen deshalb das Eingabebild nicht zu überdecken. Die Regionen in `ClassRegions` sollten sich nicht überlappen, da dies dazu führen würde, dass in den Trainingsdaten die Merkmalsvektoren in den Überlappungsbereichen mehreren Klassen zugeordnet wären, was zu einer langsameren Konvergenz des Trainings und einer schlechteren Klassifikationsleistung führen kann.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Trainingsbild.

▷ **ClassRegions** (input_object) ..... region-array ⇝ *object*
Regionen der zu trainierenden Klassen.

▷ **KNNHandle** (input_control) ..... class_knn ⇝ *handle*
Handle des k-NN-Klassifikators.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_samples_image_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- KNNHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_knn`

Nachfolger

`train_class_knn`

Alternativen

`add_sample_class_knn`

Siehe auch

`classify_image_class_knn`, `add_sample_class_knn`, `add_samples_image_class_svm`

Modul

Foundation

add_samples_image_class_mlp (Image, ClassRegions : : MLPHandle :)

Hinzufügen von Trainingsmustern aus einem Bild zu den Trainingsdaten eines mehrschichtigen Perzeptrons.

`add_samples_image_class_mlp` fügt Trainingsmuster aus dem Bild `Image` zu dem durch `MLPHandle` gegebenen mehrschichtigen Perzeptron (MLP) hinzu. `add_samples_image_class_mlp` wird dazu verwendet, die Trainingsmuster bereitzustellen, um einen Klassifikator zur Pixelklassifikation von mehrkanaligen Bildern mit `classify_image_class_mlp` zu trainieren. Die Funktionsweise von `add_samples_image_class_mlp` ist analog zu `add_sample_class_mlp`. Da das MLP hier aber immer zur Klassifikation verwendet wird, muss bei `create_class_mlp` `OutputFunction` = *'softmax'* angegeben werden. Das Bild `Image` muss so viele Kanäle besitzen, wie mit `NumInput` in `create_class_mlp` festgelegt worden ist. In `ClassRegions` werden die Trainingsgebiete für die `NumOutput` Pixelklassen angegeben. `ClassRegions` muss also ein Tupel mit `NumOutput` Regionen beinhalten. Die Reihenfolge der Regionen in `ClassRegions` legt die Klassen der Pixel fest. Falls in `Image` eine zu trainierende Klasse nicht vorkommt, muss an der entsprechenden Stelle des Tupels eine leere Region übergeben werden. Durch diesen Mechanismus können mehrere Bilder verwendet werden, um mit mehrmaligem Aufruf von `add_samples_image_class_mlp` Trainingsmuster für alle relevanten Pixelklassen bereitzustellen. Die Regionen in `ClassRegions` sollten repräsentative Trainingsmuster für die jeweiligen Klassen enthalten. Sie brauchen deshalb das Eingabebild nicht zu überdecken. Die Regionen in `ClassRegions` sollten sich nicht überlappen, da dies dazu führen würde, dass in den Trainingsdaten die Merkmalsvektoren in den Überlappungsbereichen mehreren Klassen zugeordnet wären, was zu einer langsameren Konvergenz des Trainings und einer schlechteren Klassifikationsleistung führen kann.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Trainingsbild.

▷ **ClassRegions** (input_object) ..... region-array ⇝ *object*
Regionen der zu trainierenden Klassen.

▷ **MLPHandle** (input_control) ..... class_mlp ⇝ *handle*
Handle des MLP.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_samples_image_class_mlp` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MLPHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_mlp`

Nachfolger

`train_class_mlp`, `write_samples_class_mlp`

Alternativen

`read_samples_class_mlp`

Siehe auch

`classify_image_class_mlp`, `add_sample_class_mlp`, `clear_samples_class_mlp`, `get_sample_num_class_mlp`, `get_sample_class_mlp`, `add_samples_image_class_svm`

Modul

Foundation

add_samples_image_class_svm (Image, ClassRegions : : SVMHandle :)

Hinzufügen von Trainingsmustern aus einem Bild zu den Trainingsdaten einer Support-Vektor-Maschine.

`add_samples_image_class_svm` fügt Trainingsmuster aus dem Bild `Image` zu der durch `SVMHandle` gegebenen Support-Vektor-Maschine (SVM) hinzu. `add_samples_image_class_svm` wird dazu verwendet, die Trainingsmuster bereitzustellen, um einen Klassifikator zur Pixelklassifikation von mehrkanaligen Bildern mit `classify_image_class_svm` zu trainieren. Die Funktionsweise von `add_samples_image_class_svm` ist analog zu `add_sample_class_svm`.

Das Bild `Image` muss so viele Kanäle besitzen, wie mit `NumFeatures` in `create_class_svm` festgelegt worden ist. In `ClassRegions` werden die Trainingsgebiete für die `NumClasses` Pixelklassen angegeben. `ClassRegions` muss also ein Tupel mit `NumClasses` Regionen beinhalten. Die Reihenfolge der Regionen in `ClassRegions` legt die Klassen der Pixel fest. Falls in `Image` eine zu trainierende Klasse nicht vorkommt, muss an der entsprechenden Stelle des Tupels eine leere Region übergeben werden. Durch diesen Mechanismus können mehrere Bilder verwendet werden, um mit mehrmaligem Aufruf von `add_samples_image_class_svm` Trainingsmuster für alle relevanten Pixelklassen bereitzustellen.

Die Regionen in `ClassRegions` sollten repräsentative Trainingsmuster für die jeweiligen Klassen enthalten. Sie brauchen deshalb das Eingabebild nicht zu überdecken. Die Regionen in `ClassRegions` sollten sich nicht überlappen, da dies dazu führen würde, dass in den Trainingsdaten die Merkmalsvektoren in den Überlappungsbereichen mehreren Klassen zugeordnet wären, was zu einer langsameren Konvergenz des Trainings und einer schlechteren Klassifikationsleistung führen kann.

Eine weitere Anwendung ist die automatische Detektion von Abweichungen in Farben bzw. Texturen. Hierbei wird die SVM, die in dem Modus *'novelty-detection'* erzeugt wurde, eine korrekte Trainingsmenge übergeben (z.B. Regionen mit Hautpartien für Hautdetektion oder die in dem Fertigungsprozess als korrekt definierten Texturbeispiele). Anschließend können mit `classify_image_class_svm` abweichende Regionen detektiert werden (z.B. Nicht-Haut Regionen oder Fehler in Texturen).

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Trainingsbild.

▷ **ClassRegions** (input_object) ..... region-array ⇝ *object*
Regionen der zu trainierenden Klassen.

▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
Handle der SVM.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_samples_image_class_svm` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SVMHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_class_svm`

Nachfolger

`train_class_svm`, `write_samples_class_svm`

Alternativen

`read_samples_class_svm`

Siehe auch

`classify_image_class_svm`, `add_sample_class_svm`, `clear_samples_class_svm`, `get_sample_num_class_svm`, `get_sample_class_svm`, `add_samples_image_class_mlp`

Modul

Foundation

class_2dim_sup (ImageCol, ImageRow, FeatureSpace : RegionClass2Dim : :)

Segmentieren durch 2-dimensionale Pixelklassifikation.

`class_2dim_sup` klassifiziert Bildpunkte mit Hilfe von zweikanaligen Bildern und einem zweidimensionalen Merkmalsraum. Für jeden Bildpunkt werden zwei Grauwerte (aus jedem Bild einer) als Merkmale verwendet. Der Merkmalsraum wird durch eine Region repräsentiert. Die Klassifikation wird wie folgt durchgeführt:

Ein Bildpunkt aus der Eingaberegion wird genau dann übernommen, wenn der Punkt (g_r, g_c), der durch die Grauwerte festgelegt wird, in der Region `FeatureSpace` enthalten ist. Dabei ist g_r ein Grauwert aus dem Bild `ImageRow` und g_c ein Grauwert aus dem Bild `ImageCol`.

Sei P ein Bildpunkt mit den Koordinaten $P = (R, C)$, g_r der Grauwert an der Position (R, C) in dem Bild `ImageRow` und g_c der Grauwert an der Position (R, C) in dem Bild `ImageCol`. Dann wird er in die Ergebnisregion (`RegionClass2Dim`) übernommen, falls:

$$(g_r, g_c) \in \texttt{FeatureSpace}$$

g_r wird also als Zeilen- und g_c als Spaltenkoordinate interpretiert.

Für die Erzeugung von `FeatureSpace` siehe `histo_2dim`. Der Merkmalsraum kann vor der Anwendung von `class_2dim_sup` noch mit Regionentransformationen wie `rank_region`, `dilation1`, `shape_trans`, `elliptic_axis`, etc. bearbeitet werden.

Die Parameter `ImageCol` und `ImageRow` müssen gleich viele Bilder mit jeweils gleicher Größe enthalten. Die Bildpunkte werden aus dem Durchschnitt der beiden Definitionsbereiche entnommen (siehe `reduce_domain`).

Parameter

▷ **ImageCol** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1
Bild mit Grauwerten für ersten Kanal.

▷ **ImageRow** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1
Bild mit Grauwerten für zweiten Kanal.

▷ **FeatureSpace** (input_object) ..... region(-array) ⇝ *object*
Region, die den Merkmalsraum festlegt.

▷ **RegionClass2Dim** (output_object) ..... region(-array) ⇝ *object*
Regionen, die das Klassifikationskriterium erfüllen.

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " 'image' " << endl;
    return (-1);
  }

  HRegion   feats, cd2reg;
  HImage    image (argv[1]),
            text1, text2,
            mean1, mean2,
            histo;

  HWindow   win;
  Hlong      nc;

  if ((nc = image.CountChannels ()) != 3)
  {
    cout << argv[1] << " is not a rgb-image " << endl;
    return (-2);
  }

  image.Display (win);

  win.SetColor ("green");
  cout << "Draw the region of interrest " << endl;

  HRegion  region = win.DrawRegion ();

  text1 = image.TextureLaws ("el", 2, 5);
  mean1 = text1.MeanImage (21, 21);
  text2 = mean1.TextureLaws ("es", 2, 5);
  mean2 = text2.MeanImage (21, 21);

  histo = region.Histo2dim (mean1, mean2);
  feats = histo.Threshold (1.0, 1000000.0);

  win.SetDraw ("fill");
  win.SetColor ("red");

  feats.Display (win);

  cout << "Charakteristics area in red" << endl;

  cd2reg = mean1.Class2dimSup (mean2, feats);

  win.SetColor ("blue");
  cd2reg.Display (win);
```

```
  cout << "Result of classification in blue " << endl;
  win.Click ();
  return (0);
}
```

Komplexität

Sei A die Fläche der Eingaberegion dann ist die Laufzeitkomplexität $O(256^2 + A)$.

Ergebnis

`class_2dim_sup` liefert den Wert TRUE. Falls alle Parameter korrekt sind für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`histo_2dim`, `threshold`, `draw_region`, `dilation1`, `opening`, `shape_trans`

Nachfolger

`connection`, `select_shape`, `select_gray`

Alternativen

`class_ndim_norm`, `threshold`

Siehe auch

`histo_2dim`

Modul

Foundation

`class_2dim_unsup` `( Image1, Image2 : Classes : Threshold, NumClasses : )`

Segmentieren zweier Bilder mittels Clustering.

`class_2dim_unsup` führt eine Klassifikation mit zwei einkanaligen Bildern durch. Als erstes wird ein zweidimensionales Histogramm über die zwei gegebenen Bilder aufgestellt (`histo_2dim`). In diesem zweidimensionalen Histogramm wird das erste Maximum gesucht. Es stellt das erste Clusterzentrum dar. Das Histogramm wird mit den Bildpunkten berechnet, die in beiden Bildern definiert sind (siehe `reduce_domain`). Danach werden im Bild alle Pixel gesucht, die sich höchstens um `Threshold` von den Grauwerten der des Clusterzentrums unterscheiden (Maximalabstand vom Zentrum). Die gefundenen Pixel bilden eine Region. Anschließend werden die zu den gefundenen Pixel gehörenden Punkte im Histogramm gelöscht, damit sie bei der nächsten Klassenbildung nicht mehr berücksichtigt werden müssen. In diesem modifizierten Histogramm wird wiederum das Maximum gesucht; es bildet das neue Clusterzentrum. Die obigen genannten Schritte werden wiederholt. Insgesamt werden die Schritte `NumClasses`-mal, wiederholt; es ergeben sich also `NumClasses`-Ausgaberegionen. Es werden nur Punkte ausgegeben, die in beiden Bildern definiert sind.

Achtung

Die beiden Eingabebilder müssen die gleiche Größe haben.

Parameter

- ▷ **`Image1`** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Erstes Eingabebild.
- ▷ **`Image2`** (input_object) ..... singlechannelimage ⇝ *object :* byte
  Zweites Eingabebild.
- ▷ **`Classes`** (output_object) ..... region-array ⇝ *object*
  Ergebnis der Klassifikation.

▷ **Threshold** (input_control) ..... integer ⇝ *integer*
Schwellenwert (maximaler Abstand vom Clusterzentrum).
Defaultwert: 15
Wertevorschläge: `Threshold` ∈ {0, 2, 5, 8, 12, 17, 20, 30, 50, 70}

▷ **NumClasses** (input_control) ..... integer ⇝ *integer*
Anzahl der Klassen (Clusterzentren).
Defaultwert: 5
Wertevorschläge: `NumClasses` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50}

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " 'image' " << endl;
    return (-1);
  }

  HImage   colimg (argv[1]),
           green, blue;

  HWindow  w;
  Hlong     nc;

  if ((nc = colimg.CountChannels ()) != 3)
  {
    cout << argv[1] << " is not a rgb-image " << endl;
    return (-2);
  }

  colimg.Display (w);

  HImage        red = colimg.Decompose3 (&green, &blue);
  HRegionArray  seg = red.Class2dimUnsup (green, 15, 5);

  w.SetDraw ("margin");
  w.SetColored (12);
  seg.Display (w);
  w.Click ();

  return (0);
}
```

Ergebnis

`class_2dim_unsup` liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

decompose2, decompose3, median_image, anisotropic_diffusion, reduce_domain

Nachfolger

select_shape, select_gray, connection

Alternativen

threshold, histo_2dim, class_2dim_sup, class_ndim_norm

Modul

Foundation

class_ndim_norm (MultiChannelImage : Regions : Metric, SingleMultiple, Radius, Center :)

Ausführen einer Pixelklassifikation mit Hyperkugeln oder Hyperwürfeln.

class_ndim_norm klassifiziert die Pixel der in MultiChannelImage enthaltenen Bilder. Als Ergebnis werden in Regions für jedes Klassifizierungsobjekt eine (oder mehrere) Region(en) ausgegeben (s.u.). Die verwendete Metrik (*'euclid'* oder *'maximum'*) wird mit Metric festgelegt. Es ist darauf zu achten, dass dieser Parameter genauso wie bei learn_ndim_norm besetzt wird. Mit SingleMultiple wird festgelegt, ob als Ergebnis (in Regions) eine Region ('single') oder für jedes Cluster eine eigene Region ('multiple') erzeugt werden soll. Der Parameter Radius gibt die Clusterradien bzw. die halbe Clusterkantenlänge an. Center enthält die Koordinaten aller Clusterzentren.

Parameter

- ▷ **MultiChannelImage** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Mehrkanaliges Eingabebild.
- ▷ **Regions** (output_object) ..... region-array ⇝ *object*
  Ergebnis der Klassifikation.
- ▷ **Metric** (input_control) ..... string ⇝ *string*
  Verwendete Metrik.
  Defaultwert: 'euclid'
  Werteliste: Metric ∈ {'euclid', 'maximum'}
- ▷ **SingleMultiple** (input_control) ..... string ⇝ *string*
  Als Ergebnis eine Region oder für jeden Cluster eine eigene Region.
  Defaultwert: 'single'
  Werteliste: SingleMultiple ∈ {'single', 'multiple'}
- ▷ **Radius** (input_control) ..... number(-array) ⇝ *real* / integer
  Clusterradien bzw. halbe Clusterkantenlängen (von learn_ndim_norm geliefert).
- ▷ **Center** (input_control) ..... number(-array) ⇝ *real* / integer
  Koordinaten aller Clusterzentren (von learn_ndim_norm geliefert).

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main ()
{
  HImage   image ("meer"),
           t1, t2, t3,
           m1, m2, m3, m;
```

```
  HWindow  w;

  w.SetColor ("green");
  image.Display (w);

  cout << "Draw your region of interest " << endl;

  HRegion testreg = w.DrawRegion ();

  t1 = image.TextureLaws ("el", 2, 5);     m1 = t1.MeanImage (21, 21);
  t2 = image.TextureLaws ("es", 2, 5);     m2 = t2.MeanImage (21, 21);
  t3 = image.TextureLaws ("le", 2, 5);     m3 = t3.MeanImage (21, 21);

  m  = m1.Compose3 (m2, m3);

  Tuple Metric = "euclid";
  Tuple Radius = 20.0;
  Tuple MinNum = 5;
  Tuple NbrCha = 3;

  HRegion empty;
  Tuple cen, t;

  Radius = testreg.LearnNdimNorm (empty, m, Metric, Radius,
                                  MinNum, NbrCha, &cen, &t);
  Tuple RegMod = "multiple";

  HRegionArray reg = m.ClassNdimNorm (Metric, RegMod, Radius, cen, NbrCha);

  w.SetColored (12);
  reg.Display (w);
  cout << "Result of classification" << endl;
  return (0);
}
```

Komplexität

Sei N die Anzahl der Cluster (= Länge der Tupel Radius und Center) und A die Fläche der Eingaberegion(en), dann ist die Laufzeitkomplexität $O(N, A)$.

Ergebnis

`class_ndim_norm` liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`learn_ndim_norm`, `compose2`, `compose3`, `compose4`, `compose5`, `compose6`, `compose7`

Nachfolger

`connection`, `select_shape`, `reduce_domain`, `select_gray`

Alternativen

`class_2dim_sup`, `class_2dim_unsup`

Modul

Foundation

classify_image_class_gmm (Image : ClassRegions : GMMHandle, RejectionThreshold :)

Klassifizieren eines Bildes durch ein Gaussian Mixture Model.

classify_image_class_gmm führt auf dem mehrkanaligen Bild Image eine Pixelklassifikation mit dem Gaussian Mixture Model (GMM) GMMHandle durch. Das GMM muss vor der Verwendung von classify_image_class_gmm mit train_class_gmm trainiert werden. Image muss die bei create_class_gmm mit NumDim spezifizierte Anzahl von Kanälen besitzen. Als Ausgabe werden in ClassRegions NumClasses Regionen als Ergebnis der Pixelklassifikation zurückgegeben. Die Reihenfolge der Regionen in ClassRegions entspricht dabei der Reihenfolge der Klassen, wie sie durch die Trainingsregionen in add_samples_image_class_gmm definiert wurde. Der Parameter RejectionThreshold erlaubt die Zurückweisung von zu unsicher klassifizierten Pixeln. Er stellt einen Schwellenwert auf dem von der Klassifikation zurückgelieferten K-Sigma-Wahrscheinlichkeitsmaß dar (siehe classify_class_gmm und evaluate_class_gmm). Alle Pixel mit einer Wahrscheinlichkeit unterhalb von RejectionThreshold werden keiner Klasse zugeordnet.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Eingabebild.

▷ **ClassRegions** (output_object) ..... region-array ⇝ *object*
Segmentierte Klassen.

▷ **GMMHandle** (input_control) ..... class_gmm ⇝ *handle*
Handle des GMM.

▷ **RejectionThreshold** (input_control) ..... real ⇝ *real*
Schwellenwert für die Rückweisung der Klassifikation.
Defaultwert: 0.5
Wertevorschläge: RejectionThreshold ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: RejectionThreshold >= 0.0 && RejectionThreshold <= 1.0

Beispiel

```
read_image (Image, 'ic')
gen_rectangle1 (Board, 80, 320, 110, 350)
gen_rectangle1 (Capacitor, 359, 263, 371, 302)
gen_rectangle1 (Resistor, 200, 252, 290, 256)
gen_rectangle1 (IC, 180, 135, 216, 165)
concat_obj (Board, Capacitor, Classes)
concat_obj (Classes, Resistor, Classes)
concat_obj (Classes, IC, Classes)
create_class_gmm (3, 4, [1,30], 'full', 'none',0, 42, GMMHandle)
add_samples_image_class_gmm (Image, Classes, GMMHandle, 1.5)
get_sample_num_class_gmm (GMMHandle, NumSamples)
train_class_gmm (GMMHandle, 150, 1e-4, 'training', 1e-4, Centers, Iter)
classify_image_class_gmm (Image, ClassRegions, GMMHandle, 0.0001)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert classify_image_class_gmm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

train_class_gmm, read_class_gmm

Alternativen

classify_image_class_knn, classify_image_class_mlp, classify_image_class_svm, classify_image_class_lut, class_ndim_norm, class_2dim_sup

Siehe auch

add_samples_image_class_gmm, create_class_gmm

Modul

Foundation

classify_image_class_knn (Image : ClassRegions, DistanceImage : KNNHandle, RejectionThreshold :)

Klassifizieren eines Bildes durch einen k-NN-Klassifikator

classify_image_class_knn führt auf dem mehrkanaligen Bild Image eine Pixelklassifikation mit dem k-NN-Klassifikator KNNHandle durch. Der k-NN muss vor der Verwendung von classify_image_class_knn mit train_class_knn trainiert werden. Image muss die bei create_class_knn mit NumInput spezifizierte Anzahl von Kanälen besitzen. Als Ausgabe werden in ClassRegions NumClasses Regionen als Ergebnis der Pixelklassifikation zurückgegeben. Die Reihenfolge der Regionen in ClassRegions entspricht dabei der Reihenfolge der Klassen, wie sie durch die Trainingsregionen in add_samples_image_class_knn definiert wurde. Der Parameter RejectionThreshold erlaubt die Zurückweisung von zu unsicher klassifizierten Pixeln. Er stellt einen Schwellenwert auf dem von der Klassifikation zurückgelieferten Wahrscheinlichkeitsmaß dar (siehe classify_class_knn). Alle Pixel mit einer Wahrscheinlichkeit unterhalb von RejectionThreshold werden keiner Klasse zugeordnet. Da ein k-NN-Klassifikator bei Pixeldaten, die im Merkmalsraum außerhalb der konvexen Hülle der Trainingsdaten liegen einer der Klassen zuordnet, empfiehlt es sich in vielen Fällen, trotz der Verwendung von RejectionThreshold, eine Rückweisungsklasse explizit zu trainieren, indem mit add_samples_image_class_knn Trainingsmuster für die Rückweisungsklasse abgespeichert werden und der k-NN mit train_class_knn neu trainiert wird.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Eingabebild.

▷ **ClassRegions** (output_object) ..... region-array ⇝ *object :* real
Segmentierte Klassen.

▷ **DistanceImage** (output_object) ..... image ⇝ *object*
Bild mit Distanzen pro Pixel zum jeweiligen nächsten Nachbarn.

▷ **KNNHandle** (input_control) ..... class_knn ⇝ *handle*
Handle des k-NN-Klassifikators.

▷ **RejectionThreshold** (input_control) ..... real ⇝ *real*
Schwellenwert für die Rückweisung der Klassifikation.
Defaultwert: 0.5
Wertevorschläge: RejectionThreshold ∈ {0.0, 0.1, 0.2, 0.3, 5.0, 10.0, 255.0}
Restriktion: RejectionThreshold >= 0.0

Beispiel

```
read_image (Image, 'ic')
gen_rectangle1 (Board, 80, 320, 110, 350)
gen_rectangle1 (Capacitor, 359, 263, 371, 302)
gen_rectangle1 (Resistor, 200, 252, 290, 256)
gen_rectangle1 (IC, 180, 135, 216, 165)
concat_obj (Board, Capacitor, Classes)
concat_obj (Classes, Resistor, Classes)
concat_obj (Classes, IC, Classes)
create_class_knn (3, KNNHandle)
add_samples_image_class_knn (Image, Classes, KNNHandle)
get_sample_num_class_knn (KNNHandle, NumSamples)
```

```
train_class_knn (KNNHandle, [], [])
classify_image_class_knn (Image, ClassRegions, DistanceImage, KNNHandle, 0.5)
dev_display (ClassRegions)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `classify_image_class_knn` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`train_class_knn`, `read_class_knn`

Alternativen

`classify_image_class_svm`, `classify_image_class_mlp`, `classify_image_class_gmm`, `classify_image_class_lut`, `class_ndim_norm`, `class_2dim_sup`

Siehe auch

`add_samples_image_class_knn`, `create_class_knn`

Modul

Foundation

classify_image_class_lut (Image : ClassRegions : ClassLUTHandle :)

Klassifizieren eines Byte-Bildes mit einer Look-Up-Table.

`classify_image_class_lut` führt auf einem mehrkanaligen Byte-Bild `Image` eine Pixelklassifikation anhand einer Look-Up-Table (LUT) `ClassLUTHandle` durch. Der Operator kann `classify_image_class_gmm`, `classify_image_class_knn`, `classify_image_class_mlp` und `classify_image_class_svm` ersetzen, was die Klassifikation deutlich beschleunigt, weil die entsprechende Klasse nicht in jedem Bildpunkt extra berechnet werden muss, da alle möglichen Antworten des GMM, KNN, MLP bzw. SVM in der LUT gespeichert sind. Dieser LUT Klassifikator muss vor der Verwendung von `classify_image_class_lut` durch den entsprechend trainierten Klassifikator mit `create_class_lut_gmm`, `create_class_lut_knn`, `create_class_lut_mlp` bzw. `create_class_lut_svm` erzeugt werden. Für die Klassifizierung sind die in `create_class_gmm`, `create_class_knn`, `create_class_mlp` bzw. `create_class_svm` angegeben Parameter von Bedeutung: Das Byte-Bild `Image` muss die selbe Anzahl von Kanälen besitzen, wie in `NumInput`, `NumFeatures` bzw. `NumDim` spezifiziert wurde. Als Ausgabe werden in `ClassRegions` `NumOutput` bzw. `NumClasses` Regionen als Ergebnis der Pixelklassifikation zurückgegeben.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte
  Eingabebild.
- ▷ **ClassRegions** (output_object) ..... region-array ⇝ *object*
  Segmentierte Klassen.
- ▷ **ClassLUTHandle** (input_control) ..... class_lut ⇝ *handle*
  Handle des LUT Klassifikators.

Beispiel

```
read_image (Image, 'patras')
gen_rectangle1 (Sea, 10, 10, 120, 270)
gen_rectangle2 (Deck, [170,400], [350,375], [-0.56192,-0.75139], \
                [64,104], [26,11])
```

```
union1 (Deck, Deck)
gen_rectangle1 (Walls, 355, 623, 420, 702)
gen_rectangle2 (Chimney, 286, 623, -0.56192, 64, 33)
concat_obj (Sea, Deck, Classes)
concat_obj (Classes, Walls, Classes)
concat_obj (Classes, Chimney, Classes)
*
* create MLP classifier and train it with sample classes
create_class_mlp (3, 3, 4, 'softmax', 'principal_components', 3, \
                  42, MLPHandle)
add_samples_image_class_mlp (Image, Classes, MLPHandle)
train_class_mlp (MLPHandle, 200, 1, 0.01, Error, ErrorLog)
*
* create the LUT classifier
create_class_lut_mlp (MLPHandle, [], [], ClassLUTHandle)
*
* classify the image with the LUT
classify_image_class_lut (Image, ClassRegions, ClassLUTHandle)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `classify_image_class_lut` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`create_class_lut_gmm`, `create_class_lut_knn`, `create_class_lut_mlp`, `create_class_lut_svm`

Alternativen

`classify_image_class_gmm`, `classify_image_class_knn`, `classify_image_class_mlp`, `classify_image_class_svm`

Siehe auch

`create_class_lut_gmm`, `create_class_lut_knn`, `create_class_lut_mlp`, `create_class_lut_svm`, `create_class_lut_gmm`

Modul

Foundation

`classify_image_class_mlp` (`Image` : `ClassRegions` : `MLPHandle`, `RejectionThreshold` :)

Klassifizieren eines Bildes durch ein mehrschichtiges Perzeptron.

`classify_image_class_mlp` führt auf dem mehrkanaligen Bild `Image` eine Pixelklassifikation mit dem mehrschichtigen Perzeptron (MLP) `MLPHandle` durch. Das MLP muss vor der Verwendung von `classify_image_class_mlp` mit `train_class_mlp` trainiert werden. `Image` muss die bei `create_class_mlp` mit `NumInput` spezifizierte Anzahl von Kanälen besitzen. Als Ausgabe werden in `ClassRegions` `NumOutput` Regionen als Ergebnis der Pixelklassifikation zurückgegeben. Die Reihenfolge der Regionen in `ClassRegions` entspricht dabei der Reihenfolge der Klassen, wie sie durch die Trainingsregionen in `add_samples_image_class_mlp` definiert wurde. Der Parameter `RejectionThreshold` erlaubt die Zurückweisung von zu unsicher klassifizierten Pixeln. Er stellt einen Schwellenwert auf dem von der Klassifikation zurückgelieferten Wahrscheinlichkeitsmaß dar (siehe `classify_class_mlp` und `evaluate_class_mlp`). Alle Pixel mit einer Wahrscheinlichkeit unterhalb von `RejectionThreshold` werden keiner Klasse zugeordnet. Da ein MLP Pixeldaten, die im Merkmalsraum außerhalb der konvexen Hülle der Trainingsdaten liegen,

typischerweise mit einer relativ hohen Wahrscheinlichkeit (Konfidenz) einer der Klassen zuordnet, empfiehlt es sich in vielen Fällen trotz der Verwendung von RejectionThreshold, eine Rückweisungsklasse explizit zu trainieren, indem mit add_samples_image_class_mlp Trainingsmuster für die Rückweisungsklasse abgespeichert werden und das Netz mit train_class_mlp neu trainiert wird.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Eingabebild.

▷ **ClassRegions** (output_object) ..... region-array ⇝ *object*
Segmentierte Klassen.

▷ **MLPHandle** (input_control) ..... class_mlp ⇝ *handle*
Handle des MLP.

▷ **RejectionThreshold** (input_control) ..... real ⇝ *real*
Schwellenwert für die Rückweisung der Klassifikation.
Defaultwert: 0.5
Wertevorschläge: RejectionThreshold ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Restriktion: RejectionThreshold >= 0.0 && RejectionThreshold <= 1.0

Beispiel

```
read_image (Image, 'ic')
gen_rectangle1 (Board, 80, 320, 110, 350)
gen_rectangle1 (Capacitor, 359, 263, 371, 302)
gen_rectangle1 (Resistor, 200, 252, 290, 256)
gen_rectangle1 (IC, 180, 135, 216, 165)
concat_obj (Board, Capacitor, Classes)
concat_obj (Classes, Resistor, Classes)
concat_obj (Classes, IC, Classes)
create_class_mlp (3, 3, 4, 'softmax', 'principal_components', 3, 42, \
                  MLPHandle)
add_samples_image_class_mlp (Image, Classes, MLPHandle)
get_sample_num_class_mlp (MLPHandle, NumSamples)
train_class_mlp (MLPHandle, 200, 1, 0.01, Error, ErrorLog)
classify_image_class_mlp (Image, ClassRegions, MLPHandle, 0.5)
dev_display (ClassRegions)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert classify_image_class_mlp den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

train_class_mlp, read_class_mlp

Alternativen

classify_image_class_gmm, classify_image_class_knn, classify_image_class_svm, classify_image_class_lut, class_ndim_norm, class_2dim_sup

Siehe auch

add_samples_image_class_mlp, create_class_mlp

Modul

Foundation

classify_image_class_svm (Image : ClassRegions : SVMHandle :)

Klassifizieren eines Bildes durch eine Support-Vektor-Maschine.

classify_image_class_svm führt auf dem mehrkanaligen Bild Image eine Pixelklassifikation mit der Support-Vektor-Maschine (SVM) SVMHandle durch. Die SVM muss vor der Verwendung von classify_image_class_svm mit train_class_svm trainiert werden. Image muss die bei create_class_svm mit NumFeatures spezifizierte Anzahl von Kanälen besitzen. Als Ausgabe werden in ClassRegions NumClasses Regionen als Ergebnis der Pixelklassifikation zurückgegeben. Die Reihenfolge der Regionen in ClassRegions entspricht dabei der Reihenfolge der Klassen, wie sie durch die Trainingsregionen in add_samples_image_class_svm definiert wurde.

Damit eine SVM Pixeldaten, die im Merkmalsraum außerhalb der konvexen Hülle der Trainingsdaten liegen, nicht einfach klassifiziert, empfiehlt es sich in vielen Fällen, eine Rückweisungsklasse explizit zu trainieren, indem mit add_samples_image_class_svm Trainingsmuster für die Rückweisungsklasse abgespeichert werden und die SVM mit train_class_svm neu trainiert wird.

Eine Möglichkeit, die explizite Angabe einer Rückweisungsklasse zu vermeiden, ist die Verwendung einer SVM im Modus *'novelty-detection'* (siehe die entsprechende Beschreibung in create_class_svm und add_samples_image_class_svm).

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *object :* byte / cyclic / direction / int1 / int2 / uint2 / int4 / real
Eingabebild.

▷ **ClassRegions** (output_object) ..... region-array ⇝ *object*
Segmentierte Klassen.

▷ **SVMHandle** (input_control) ..... class_svm ⇝ *handle*
Handle der SVM.

Beispiel

```
read_image (Image, 'ic')
gen_rectangle1 (Board, 20, 270, 160, 420)
gen_rectangle1 (Capacitor, 359, 263, 371, 302)
gen_rectangle1 (Resistor, 200, 252, 290, 256)
gen_rectangle1 (IC, 180, 135, 216, 165)
concat_obj (Board, Capacitor, Classes)
concat_obj (Classes, Resistor, Classes)
concat_obj (Classes, IC, Classes)
create_class_svm (3, 'rbf', 0.01, 0.01, 4, 'one-versus-all', \
                  'normalization', 3, SVMHandle)
add_samples_image_class_svm (Image, Classes, SVMHandle)
train_class_svm (SVMHandle, 0.001, 'default')
reduce_class_svm (SVMHandle, 'bottom_up', 2, 0.01, SVMHandleReduced)
classify_image_class_svm (Image, ClassRegions, SVMHandleReduced)
dev_display (ClassRegions)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert classify_image_class_svm den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

train_class_svm, read_class_svm, reduce_class_svm

Alternativen

classify_image_class_gmm, classify_image_class_knn, classify_image_class_mlp, classify_image_class_lut, class_ndim_norm, class_2dim_sup

Siehe auch

add_samples_image_class_svm, create_class_svm

Modul

Foundation

learn_ndim_norm (Foreground, Background, Image : : Metric, Distance, MinNumberPercent : Radius, Center, Quality)

Konstruieren von Clustern für class_ndim_norm.

learn_ndim_norm erzeugt aus den in Foreground enthaltenen Regionen Klassifikationscluster, die im Operator class_ndim_norm verwendet werden. Mit Background kann eine Klasse von Pixeln angegeben werden, die bei der Klassifikation (class_ndim_norm) nicht gefunden werden sollen. Dieser Parameter darf auch leer sein (leeres Objekt).

Mit dem Parameter Distance wird der maximale Radius für die Cluster festgelegt. Er beschreibt den minimalen Abstand zweier Clusterzentren. Wird der Parameter Distance klein gewählt, so können die (kleinen) Hyperkugeln (-Würfel) den Merkmalsraum gut approximieren. Gleichzeitig steigt jedoch der Rechenaufwand beim Klassifizieren.

Das Verhältnis aus Anzahl der Pixel in einem Cluster zu der Gesamtzahl (in Prozent) muss über dem Wert von MinNumberPercent liegen, ansonsten wird dieser Cluster nicht ausgegeben. Der Parameter MinNumberPercent dient dazu, Ausreißer in der Trainingsmenge zu eliminieren. Wird er zu groß gewählt, dann werden zu viele Cluster unterdrückt.

Es können zwei verschiedene Verfahren verwendet werden: Die minimale euklidische Distanz (n-dimensionale Kugeln) und das Maximum-Verfahren (n-dimensionale Würfel) für die Beschreibung der Pixel des zu klassifizierenden Bildobjekts im n-dimensionalen Histogramm (Parameter Metric). Die euklidische Metrik liefert i.A. die besseren Ergebnisse, benötigt jedoch auch mehr Rechenzeit. Der Parameter Quality gibt die Qualität der Clusterbildung an. Dabei wird die Überschneidung der abgewiesenen Objekte mit den Klassifizierungsobjekten berechnet. Null bedeutet maximale Überschneidung, Werte größer Null geben das entsprechende Verhältnis der Überlappung an. Wird kein Zurückweisungsobjekt angegeben, so ist dieser Wert 1. Die Objekte in Background haben jedoch keinen Einfluss auf die Konstruktion der Cluster. Sie dienen nur zur Kontrolle der zu erwartenden Ergebnisse.

Aus Anwendersicht besteht der wesentliche Unterschied zwischen learn_ndim_norm und learn_ndim_box darin, dass bei letzterem die Background-Klasse den Klassifikationsprozess selbst mitsteuert. Hier wird eine Trennfläche zwischen Foreground- und Background-Klasse erzeugt, so dass keine Punkte im Merkmalsraum falsch klassifiziert werden. Bei learn_ndim_norm dagegen ist eine Überlappung von Foreground- und Background-Klasse erlaubt. Dies schlägt sich dann im Rückgabewert Quality nieder: Je größer die Überlappung, desto kleiner die Güte der Klassifikation.

Parameter

- ▷ **Foreground** (input_object) ..... region(-array) ⇝ *object*
  Vordergrundregion
- ▷ **Background** (input_object) ..... region(-array) ⇝ *object*
  Hintergrundregion (Rückweisungsklasse)
- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte
  Trainingsbild.
- ▷ **Metric** (input_control) ..... string ⇝ *string*
  Verwendete Metrik
  Defaultwert: 'euclid'
  Werteliste: Metric ∈ {'euclid', 'maximum'}

▷ **Distance** (input_control) ..... number ⇝ *real* / integer
Maximaler Radius für die Cluster.
Defaultwert: 10.0
Wertevorschläge: Distance ∈ {1.0, 2.0, 3.0, 4.0, 6.0, 8.0, 10.0, 13.0, 17.0, 24.0, 30.0, 40.0}
Typischer Wertebereich: 0.0 ≤ Distance ≤ 511.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 1.0
Restriktion: Distance > 0.0

▷ **MinNumberPercent** (input_control) ..... number ⇝ *real* / integer
Das Verhältnis aus Anzahl der Pixel in einem Cluster zur Gesamtzahl (in Prozent) muss über dem Wert von MinNumberPercent liegen (sonst wird der Cluster nicht ausgegeben).
Defaultwert: 0.01
Wertevorschläge: MinNumberPercent ∈ {0.001, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0}
Typischer Wertebereich: 0.0 ≤ MinNumberPercent ≤ 100.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.1
Restriktion: 0 <= MinNumberPercent && MinNumberPercent <= 100

▷ **Radius** (output_control) ..... real-array ⇝ *real*
Clusterradien bzw. halbe Clusterkantenlängen.

▷ **Center** (output_control) ..... real-array ⇝ *real*
Koordinaten aller Clusterzentren.

▷ **Quality** (output_control) ..... real ⇝ *real*
Überschneidung der abgewiesenen Objekte mit den Klassifizierungsobjekten (1: keine Überschneidung).
Zusicherung: 0 <= Quality && Quality <= 1

Ergebnis

learn_ndim_norm liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder sind die Flags *'no_object_result'* und *'empty_region_result'* einstellbar (siehe set_system). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

min_max_gray, sobel_amp, binomial_filter, gauss_filter, reduce_domain, diff_of_gauss

Nachfolger

class_ndim_norm, connection, dilation1, erosion1, opening, closing, rank_region, shape_trans, skeleton

Alternativen

learn_ndim_box, learn_class_box

Siehe auch

class_ndim_norm, histo_2dim

Literatur

P. Haberäcker, „Digitale Bildverarbeitung“; Hanser-Studienbücher, München, Wien, 1987

Modul

Foundation

23.4 Maximally Stable Extremal Regions

segment_image_mser (Image : MSERDark, MSERLight : Polarity, MinArea, MaxArea, Delta, GenParamName, GenParamValue :)

Segmentieren von Maximally Stable Extremal Regions (MSER).

segment_image_mser segmentiert ein Bild in Regionen mit homogenen Grauwerten. Dies erfolgt mit Hilfe der Methode der Maximally Stable Extremal Regions (MSER). Die Homogenität einer Region wird bestimmt, indem ihre lokale Umgebung betrachtet wird. Der Operator ist daher besonders für die Segmentierung von Objekten vor einem inhomogenen Hintergrund und für Anwendungen mit wechselnder Beleuchtung geeignet.

Parameter

Polarity Der Parameter Polarity bestimmt, ob MSERs extrahiert werden, die dunkler als ihre Umgebung sind, oder solche, die heller als ihre Umgebung sind.

Wert	Bedeutung
'dark'	Es werden nur MSERs extrahiert, die dunkler als ihre Umgebung sind.
'light'	Es werden nur MSERs extrahiert, die heller als ihre Umgebung sind.
'both' (Defaultwert)	Es werden beide Arten von MSERs extrahiert.

MinArea, MaxArea Die Parameter MinArea und MaxArea beschränken die Größe der zurückgegebenen MSERs.

Sehr kleine Werte von MinArea, z.B. Werte kleiner als 5, können zu deutlich höheren Laufzeiten führen. Dies gilt insbesondere dann, wenn das zu segmentierende Bild starkes Rauschen enthält.

Wird für MaxArea ein leeres Tupel übergeben (Default), werden nur MSERs zurückgegeben die echte Teilmengen der Zusammenhangskomponenten des Bilddefinitionsbereiches sind.

Delta Der Parameter Delta beeinflusst die Selektivität des Verfahrens. Größere Werte erhöhen die Selektivität und führen zu weniger MSERs. Kleinere Werte führen zu mehr MSERs.

Eine genauere Beschreibung der Parameter und ihrer Auswirkungen findet sich im Abschnitt Segmentierungsprozess.

Die folgenden generischen Parameter können die für eine Feinanpassung der MSER-Segmentierung verwendet werden. Die generischen Parameter können mittels GenParamName und GenParamValue gesetzt werden.

'max_variation': Die maximale Variation der Fläche einer Region im Bereich von ± Delta Schwellenwerten. Größere Werte führen zu mehr MSERs. Kleinere Werte führen zu weniger MSERs.

Eine Definition des Begriffs 'variation' sowie eine genauere Beschreibung des generischen Parameters *'max_variation'* und seiner Auswirkungen findet sich im Abschnitt Segmentierungsprozess.

Werte: reelle Werte größer gleich 0.0

Vorgeschlagene Werte: 0.1, *0.2*, *0.5*, *1.0*, *2.0*, *5.0*

Defaultwert: 0.2

'min_diversity': Die Verschiedenartigkeit (minimale relative Größendifferenz oder 'diversity') zweier sich überlappender MSERs. Kleinere Werte führen zu mehr MSERs. Größere Werte führen zu weniger MSERs.

Eine Definition des Begriffs Verschiedenartigkeit ('diversity') sowie eine genauere Beschreibung des generischen Parameters *'min_diversity'* und seiner Auswirkungen findet sich im Abschnitt Segmentierungsprozess.

Werte: reelle Werte größer gleich 0.0

Vorgeschlagene Werte: 0.1, *0.5*, *0.8*, *1.0*, *2.0*, *5.0*

Defaultwert: 0.8

'may_touch_border': Steuert die Behandlung von Regionen, die den Rand des Definitionsbereiches des Eingabebildes berühren.

Wird der Parameter *'may_touch_border'* auf *'false'* gesetzt, so werden alle Regionen verworfen, die den Rand des Definitionsbereiches berühren. Wird der Parameter *'may_touch_border'* auf *'true'* gesetzt, so werden solche Regionen zurück gegeben.

Werte: 'false', *'true'*

Defaultwert: 'false', wenn der maximale Definitionsbereiches eines Bildes verwendet wird; *'true'*, wenn der Definitionsbereiche reduziert wird.

'min_gray', *'max_gray'*: Die Werte *'min_gray'* und *'max_gray'* reduzieren den Eingabedefinitionsbereich dynamisch durch die Anwendung eines Schwellenwertes (threshold) auf dem Eingabebild. Alle Pixel außerhalb des angegebenen Grauwertbereiches werden bei der Segmentierung nicht beachtet. Dadurch kann die Laufzeit erheblich reduziert werden.

Bitte beachten Sie, dass falls das Eingabebild Image einen maximalen Definitionsbereich hat, der durch Eingabe von *'min_gray'* und *'max_gray'* verkleinert wird, kann die Standardeinstellung von *'may_touch_border'* dazu führen, dass mehr Ergebnisregionen zurückgegeben werden, als ohne Einschränkung des Graubereiches.

Werte: ganzzahlige Werte größer gleich 0

Defaultwert: 'min_gray': 0, *'max_gray'*: 255 für Byte-Bilder, 65535 für uint2-Bilder

'max_gray': Der maximale Grauwert der Pixel, die im Segmentierungsprozess berücksichtigt werden sollen. Wird der Parameter auf einen Wert kleiner als der größte im Bild vorkommenden Grauwert gesetzt, werden alle Pixel mit Grauwerten größer als dieser Wert bei der Extraktion der MSER ignoriert. Dadurch kann die Laufzeit erheblich reduziert werden. Bitte beachten Sie, dass das Bild mit den gesetzten Schwellenwerten in einem Vorverarbeitungsschritt segmentiert wird. Dadurch entsteht eine neue Bilddomäne mit einem neuen Rand. Deswegen kann es notwendig sein *'may_touch_border'* auf *'true'* zu setzen um die gewünschten Regionen zu erhalten.

Werte: ganzzahlige Werte größer gleich 0.

Vorgeschlagene Werte: 200, 230

Defaultwert: 255 für Bilder vom Typ byte und 65535 für Bilder vom Typ uint2.

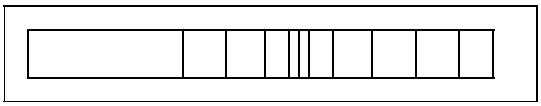
Segmentierungsprozess

In einem ersten Schritt wird das Eingabebild mit allen Schwellenwerten t, von 0 bis zum maximal vorkommenden Grauwert, segmentiert.

Zur Erläuterung wird im folgenden Beispiel ein Eingabebild mit zwölf Grauwerten (0...11) verwendet. Auf der rechten Seite der folgenden Abbildung sind die Ränder der resultierenden Schwellenwert-Regionen dargestellt.



Beispielbild mit den Grauwerten 0 (= schwarz) bis 11 (= weiß)



Ränder der Schwellenwert-Regionen für alle Schwellenwerte (t = 0...11)

Die resultierenden Schwellenwert-Regionen werden in ihre Zusammenhangskomponenten (4-er Nachbarschaft) zerlegt. Da die einzelnen Komponenten durch wachsende Schwellenwerte entstehen, nimmt ihre Fläche mit jedem neuen Schwellenwert zu. Diese Flächenveränderung der einzelnen Komponenten wird im Folgenden ausgewertet. Eine MSER ist eine Komponente, deren Fläche im Bereich von $\pm$ `Delta` Schwellenwerten nicht signifikant variiert. Um als MSER akzeptiert zu werden, muss die Variation der Fläche ein lokales Minimum sein, und sie darf nicht größer als *'max_variation'* sein. Darüber hinaus muss die Verschiedenartigkeit ('diversity') von sich überlappenden MSERs größer als *'min_diversity'* sein (siehe unten).

Die Variation der Fläche einer Komponente ist wie folgt definiert:

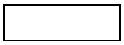
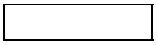
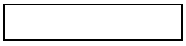
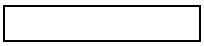
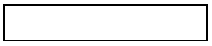
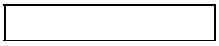
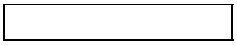
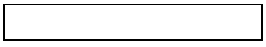
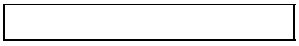
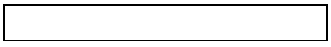
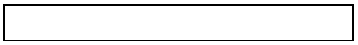
$$Variation_t = \frac{Flaeche_{t+\Delta} - Flaeche_{t-\Delta}}{Flaeche_t}$$

mit:

$$Flaeche_{t-\Delta} = Flaeche_0, \quad t - \Delta < 0$$
$$Flaeche_{t+\Delta} = Flaeche_{t_{max}}, \quad t + \Delta > t_{max}$$

Ein Verkleinern des Wertes von *'max_variation'* führt zu einer geringeren Anzahl akzeptierter Regionen.

In unserem Beispiel besteht jede Region nur aus einer einzigen Komponente. Daher werden die Bezeichnungen (Schwellenwert-)Region und Komponente im Folgenden synonym verwendet. Die folgende Tabelle zeigt die Fläche der Schwellenwert-Regionen und ihre Variation für `Delta` gleich 1 und für `Delta` gleich 2.

Schwellenwert t	Schwellenwert-Region	Fläche der Schwellenwert-Region	Variation der Fläche (für `Delta` = 1)	Variation der Fläche (für `Delta` = 2)
0		320	**0.28**	**0.53**
1		410	0.41	0.54
2		490	0.27	0.49
3		540	0.13	0.31
4		560	**0.07**	**0.25**
5		580	0.12	0.29
6		630	0.21	0.38
7		710	0.24	0.44
8		800	0.23	**0.41**
9		890	**0.18**	1.67
10		960	1.36	1.46
11		2200	**0.56**	**0.60**

Die Fläche der Schwellenwert-Regionen steigt mit wachsendem Schwellenwert t an. Für `Delta` gleich 1, treten lokale Minima der Variation für die Schwellenwerte t = 0, t = 4, t = 9 und t = 11 auf. Daher sind die entsprechenden Schwellenwert-Regionen MSER-Kandidaten. Für `Delta` gleich 2 entsprechen die MSER-Kandidaten den Schwellenwert-Regionen für t = 0, t = 4, t = 8 und t = 11.

Standardmäßig werden die Zusammenhangskomponenten des Definitionsbereichs nicht als MSERs zurückgegeben. Dieses Verhalten kann dadurch geändert werden, dass *'may_touch_border'* auf *'true'* und `MaxArea` auf einen Wert größer der Fläche des Definitionsbereichs, gesetzt wird. Auf diese Weise können auch die Zusammenhangskomponenten selbst als MSERs extrahiert werden. In unserem Beispiel führt das Standardverhalten dazu, dass die MSER-Kandidaten die den Schwellenwert-Regionen für t = 11 entsprechen, eliminiert werden.

Die folgende Abbildung zeigt die resultierenden MSER-Kandidaten für `Delta` gleich 1 und für `Delta` gleich 2 (mit *'max_variation'* gleich 1.0). Die MSERs sind dabei jeweils dem Eingabebild überlagert.

MSER-Kandidaten für Delta gleich 1	MSER-Kandidaten für Delta gleich 2
	
entspricht t = 0	entspricht t = 0
	
entspricht t = 4	entspricht t = 4
	
entspricht t = 9	entspricht t = 8

Die Segmentierungsmethode teilt das Bild nicht in disjunkte Regionen, sondern extrahiert alle gültigen MSERs, welche sich im Allgemeinen auch überlappen dürfen. Nachdem die MSER-Kandidaten durch wachsende Schwellenwerte entstehen, bestehen zwei sich überlappende Regionen immer aus einer größeren und einer kleineren Region, wobei die kleinere komplett in der größeren enthalten ist. Die Verschiedenartigkeit ('diversity') beschreibt, wie stark sich die Flächen der sich überlappenden Regionen unterscheiden. Sie wird als relative Flächendifferenz berechnet:

$$diversity_i = \frac{Flaeche_{i+1} - Flaeche_i}{Flaeche_i}$$

wobei
$Flaeche_i$ die Flaeche des aktuellen MSER-Kandidaten ist
$Flaeche_{i+1}$ die Flaeche des MSER-Kandidaten ist, der den aktuellen enthält

Es werden nur MSER-Kandidaten zurück gegeben, deren Verschiedenartigkeit ('diversity') mindestens dem Wert *'min_diversity'* entspricht. Der Parameter *'min_diversity'* kann daher verwendet werden, um die Anzahl sich überlappender Regionen zu steuern. Größere Werte für *'min_diversity'* reduzieren die Anzahl sich überlappender Regionen und kleinere Werte erhöhen diese Anzahl. Die jeweils größten MSER-Kandidaten, d.h. die Kandidaten, die nicht in einem anderen enthalten sind, werden aber in jedem Fall zurück gegeben.

	Index i des MSER-Kandidaten	**Schwellenwert**	**Fläche der Schwellenwert-Region**	**Verschiedenartigkeit ('diversity') des MSER-Kandidaten**
	0	0	320	0.75
Delta=1	1	4	560	0.59
	2	9	890	N/A
	0	0	320	0.75
Delta=2	1	4	560	0.43
	2	8	800	N/A

Zuerst wird der MSER-Kandidat mit der kleinsten Verschiedenartigkeit ('diversity') gelöscht, falls sein Wert kleiner als *'min_diversity'* ist. In unserem Beispiel wurde der Wert für *'min_diversity'* auf 0.5 gesetzt. Für Delta gleich 2 wird daher der MSER-Kandidat mit der Verschiedenartigkeit 0.43 gelöscht. Daraufhin werden die Werte für die

Verschiedenartigkeit der verbleibenden MSER-Kandidaten neu berechnet. Hierbei erhält der MSER-Kandidat 0 einen Wert von 1.5. Damit haben jetzt alle MSER-Kandidaten eine Verschiedenartigkeit die größer als 0.5 ist und werden als MSERs akzeptiert.

Das Segmentierungverfahren berechnet MSERs innerhalb jeder Zusammenhangskomponente des Definitionsbereiches des Eingabebildes.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **MSERDark** (output_object) ..... region-array ⇝ *object*
  Segmentierte dunkle MSERs.
- ▷ **MSERLight** (output_object) ..... region-array ⇝ *object*
  Segmentierte helle MSERs.
- ▷ **Polarity** (input_control) ..... string ⇝ *string*
  Polarität der segmentierten MSERs.
  Defaultwert: 'both'
  Werteliste: `Polarity` ∈ {'dark', 'light', 'both'}
- ▷ **MinArea** (input_control) ..... number(-array) ⇝ *integer* / real
  Minimale Fläche der MSERs.
  Defaultwert: 10
  Wertevorschläge: `MinArea` ∈ {1, 10, 100, 10000}
  Typischer Wertebereich: 1 ≤ MinArea
- ▷ **MaxArea** (input_control) ..... number(-array) ⇝ *integer* / real
  Maximale Fläche der MSERs.
  Defaultwert: []
  Wertevorschläge: `MaxArea` ∈ {1, 10, 100, 10000}
  Typischer Wertebereich: 1 ≤ MaxArea
- ▷ **Delta** (input_control) ..... number ⇝ *integer* / real
  Bereich von ± `Delta` Schwellenwerten, für den eine MSER stabil sein muss.
  Defaultwert: 15
  Wertevorschläge: `Delta` ∈ {5, 10, 20, 50}
  Typischer Wertebereich: 1 ≤ `Delta` ≤ 65535
- ▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: []
  Werteliste: `GenParamName` ∈ {'max_variation', 'min_diversity', 'may_touch_border', 'min_gray', 'max_gray'}
- ▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *string* / real / integer
  Werte der generischen Parameter.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {0.5, 0.8, 'true', 'false', 30, 50, 200, 230}

Beispiel

```
read_image (Image,'pellets')
segment_image_mser (Image, MSERDark, MSERLight, 'light', 1000, 10000, 3, [],[])
```

Ergebnis

Sind alle Eingabeparameter korrekt, gibt `segment_image_mser` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

select_shape, select_gray

Alternativen

auto_threshold, binary_threshold, char_threshold, local_threshold, watersheds, regiongrowing

Literatur

J. Matas, O. Chum, M. Urban, and T. Pajdla: „Robust wide baseline stereo from maximally stable extremal regions.“; Proc. of British Machine Vision Conference, pages 384-396; 2002.

Modul

Foundation

23.5 Schwellenwert-Verfahren

Dieses Kapitel beschreibt Operatoren der Schwellenwertsegmentierung.

Konzept der Schwellenwertoperatoren

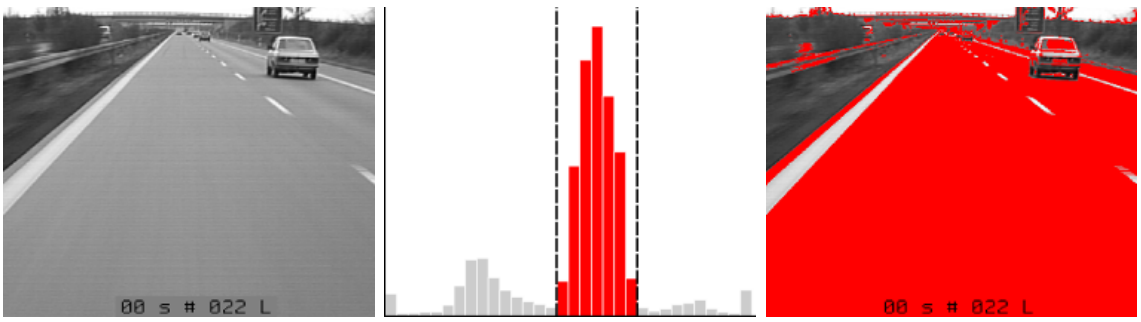
Eine Möglichkeit ein Bild zu segmentieren ist die Verwendung von Schwellenwertoperatoren. Regionen werden dabei durch Schwellenwertbedingungen bestimmt, die auf Grauwerten basieren. Je nach Anwendung können sich Aufgabenbereiche und Aufnahmesituationen stark unterscheiden, weswegen eine Reihe unterschiedlicher Operatoren zur Verfügung stehen.

Die folgenden Abschnitte geben einen Überblick über die vorhandenen Schwellenwertoperatoren. Zu diesem Zweck werden sie in histogrammbasierte und lokale Methoden eingeteilt und genauer betrachtet.

Histogrammbasierte Schwellenwertoperatoren

Unabhängig von ihrer Position im Bild werden Pixel bei der histogrammbasierten Segmentierung nur basierend auf ihren Grauwerten in Regionen eingeteilt. Die Schwellenwerte werden daher durch eine Anpassung an die Form des Histogramms bestimmt.

Threshold Mit dem Operator threshold kann der Nutzer alle Pixel auswählen, deren Wert in einem manuell bestimmten Grauwertintervall liegt.

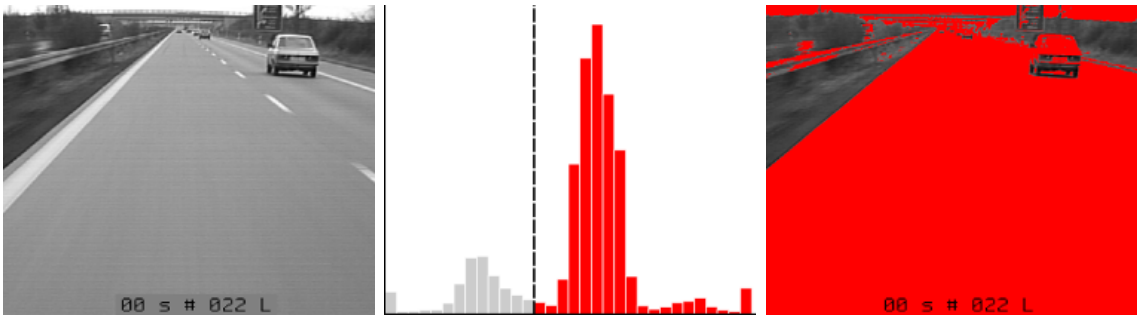


(1) (2) (3)

(1) Eingabebild, (2) Histogramm des Eingabebildes mit den manuell bestimmten Schwellenwerten MinGray und MaxGray, (3) Ergebnis der Segmentierung.

Der Operator fast_threshold arbeitet ebenfalls mit zwei manuell zu bestimmenden Schwellenwerten, nutzt allerdings einen anderen Algorithmus zur Berechnung.

Binary Threshold Um das Eingangsbild in eine helle und eine dunkle Region einzuteilen berechnet binary_threshold automatisch einen Schwellenwert, um Vorder- und Hintergrund möglichst eindeutig voneinander zu trennen.

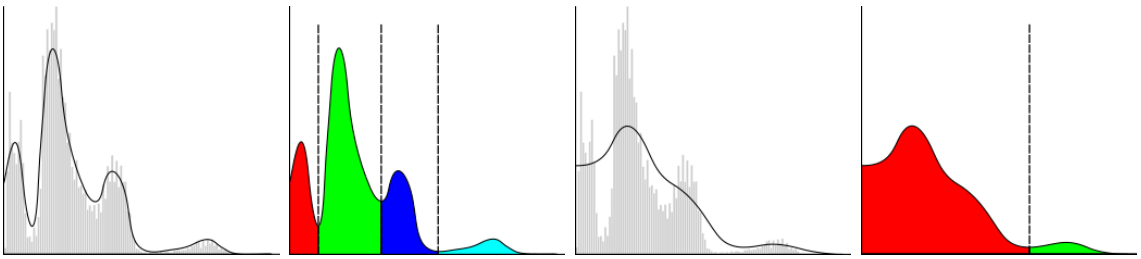


(1) (2) (3)

(1) Eingabebild, (2) Histogramm des Eingabebildes mit einem automatisch bestimmten binären Schwellenwert, (3) Ergebnis der Segmentierung.

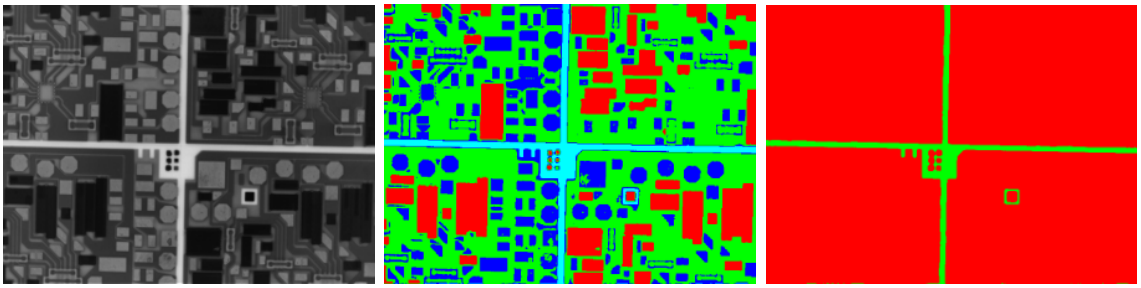
Die subpixelgenaue Grenze zwischen Vorder- und Hintergrund des Eingabebildes gibt der Operator `threshold_sub_pix` aus. Der Schwellenwert muss hierbei manuell gesetzt werden.

Automatic Threshold `auto_threshold` berechnet die lokalen Minima im Histogramm eines Bildes, um daraus die Schwellenwerte zu bestimmen. Durch die Glättung des Histogramms kann Einfluss auf die Häufung lokaler Minima und damit die Anzahl an Klassen im Ergebnisbild genommen werden.



(1) (2) (3) (4)

(1) Leicht geglättetes Histogramm, (2) höhere Anzahl an lokalen Minima, (3) stärkere Glättung des Histogramms, (4) Anzahl lokaler Minima reduziert.

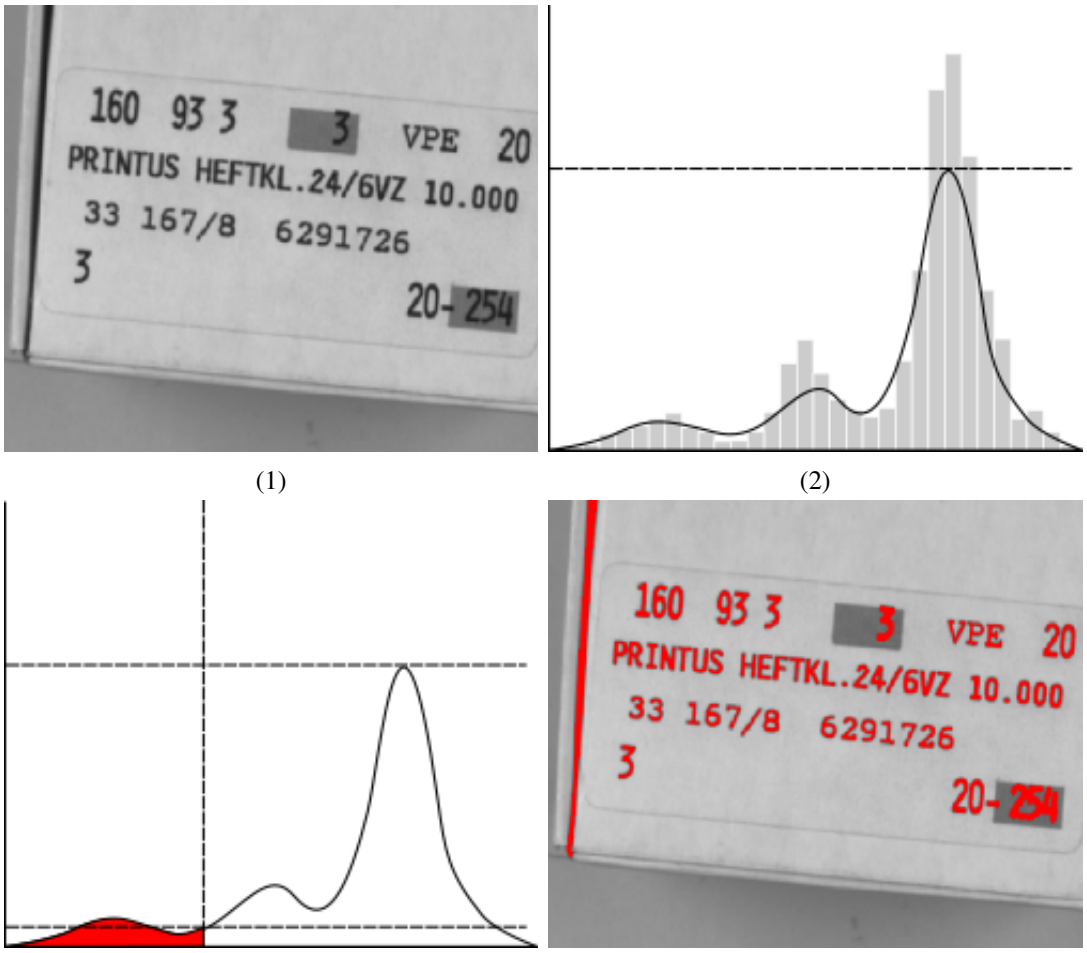


(1) (2) (3)

(1) Eingabebild, (2) vier Klassen aus leicht geglättetem Histogramm, (3) zwei Klassen aus stärker geglättetem Histogramm.

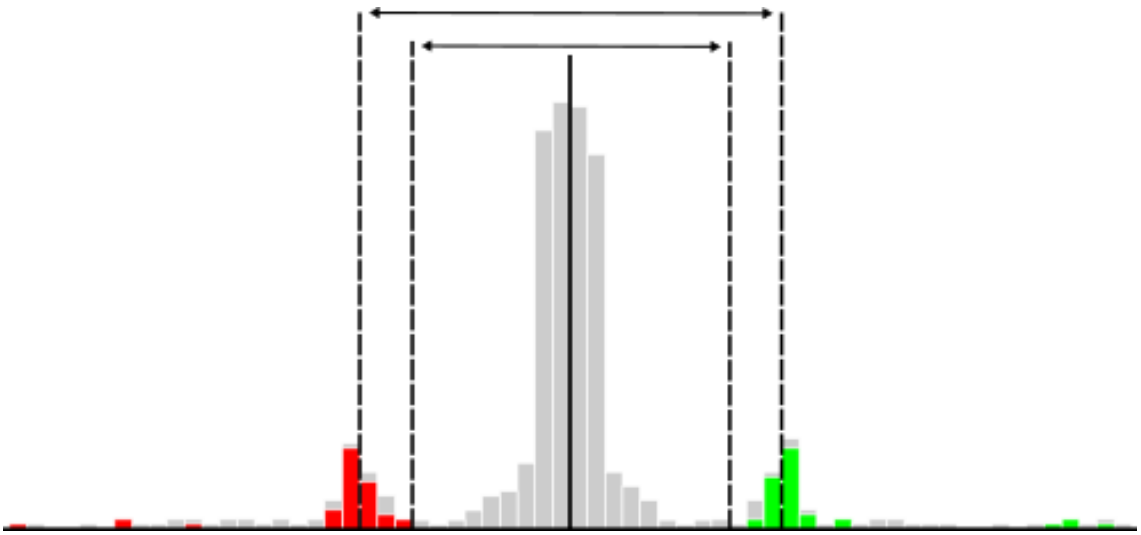
`histo_to_thresh` kann genutzt werden, um die automatisch berechneten Schwellenwerte ausgeben zu lassen.

Character Threshold Um dunklen Text auf hellem Hintergrund zu segmentieren, ist `char_threshold` ein nützliches Tool. Im Histogramm entspricht der Hintergrund dem globalen Maximum. Ohne geeignetes Minimum im unteren Grauwertbreich wird der Schwellenwert dann in Abhängigkeit von der Frequenz des maximalen Grauwerts mittels des Parameters `Percent` bestimmt.

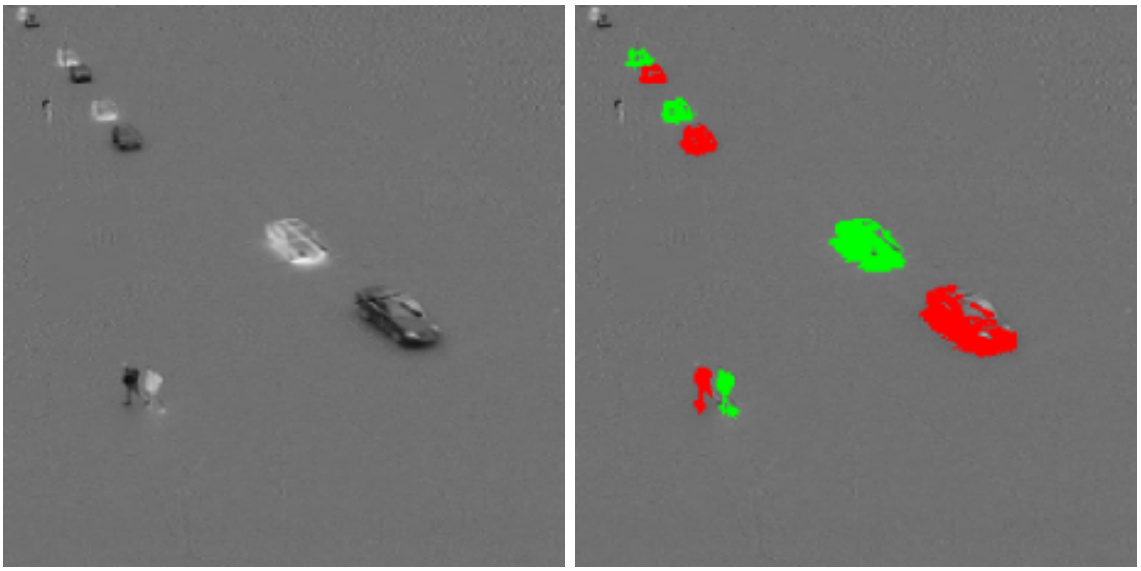


(1) Eingabebild, (2) geglättetes Histogramm mit globalem Maximum, (3) `Percent` bestimmt den Schwellenwert links des Maximums, (4) Ergebnisbild.

Dual-Threshold Subtrahiert man zwei Bilder voneinander oder nutzt einen Operator wie `laplace_of_gauss` zur Kantendetektion, so entstehen üblicherweise Bilder mit negativen Pixelwerten. Zur Segmentierung solcher Bilder unter Berücksichtigung der Regionengröße ist der Operator `dual_threshold` geeignet.



Histogramm eines Bildes mit negativen und positiven Pixelwerten mit den Schwellenwerten `Threshold` (innere Grenzen) und `MinGray` (äußere Grenzen). Pixel werden nur ausgewählt wenn Grauwert- und Größenbedingung erfüllt werden.



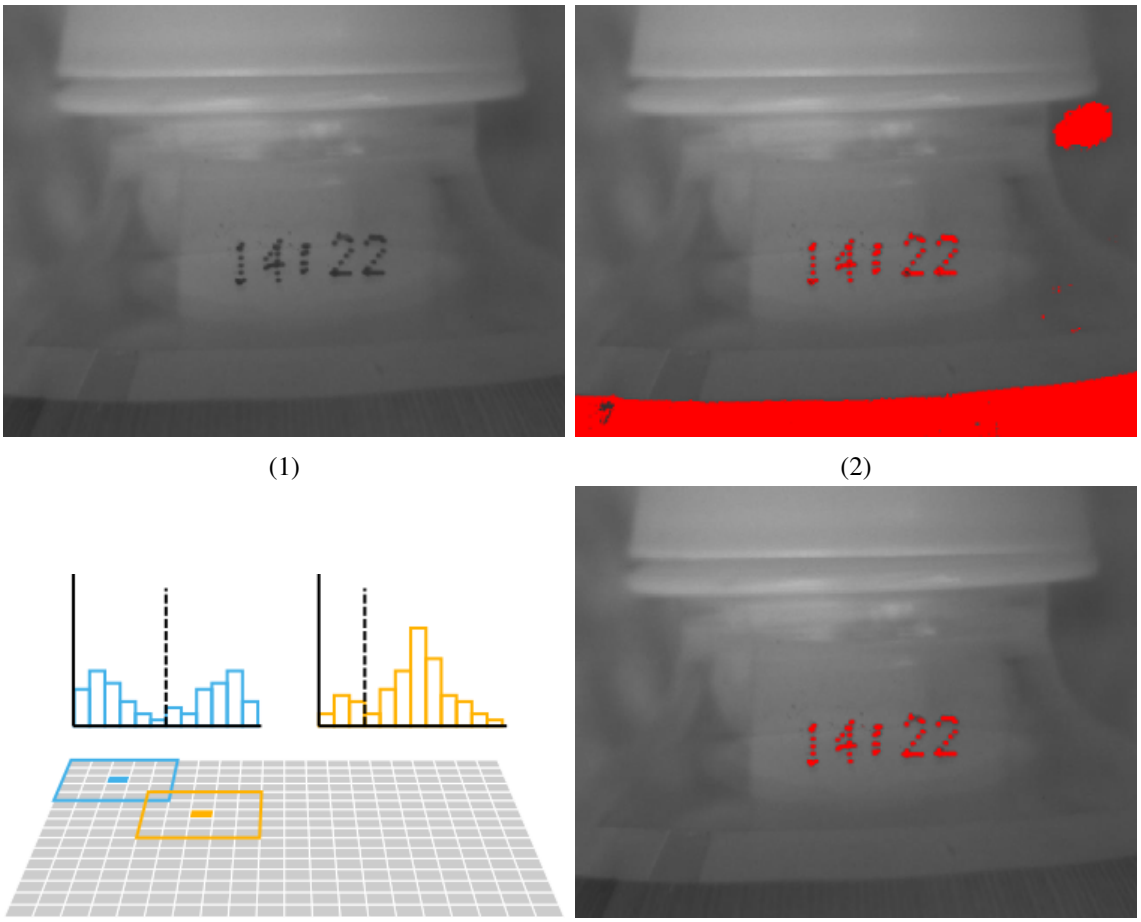
(1) (2)

(1) Mit `sub_image` erzeugtes Eingabebild, (2) Ergebnisbild nach Anwendung von `dual_threshold`.

Local Threshold Operators

Im Gegensatz zur histogrammbasierten Schwellenwertsegmentierung berücksichtigt ein lokaler Schwellenwertoperator auch die Position oder Nachbarschaft eines Pixels. Statt globalen Schwellenwerten, die auf jeden Bildpunkt angewandt werden, ist es oft sinnvoll den Schwellenwert lokal an Bildmerkmale anzupassen.

Local Threshold `local_threshold` berücksichtigt für jedes Pixel lokale Mittelwerte und Standardabweichungen. Die Größe der einbezogenen Nachbarschaft wird vom Nutzer festgelegt. Dieser Operator eignet sich besonders für die Segmentierung von Text bei inhomogenen Belichtungsverhältnissen oder Hintergrund.

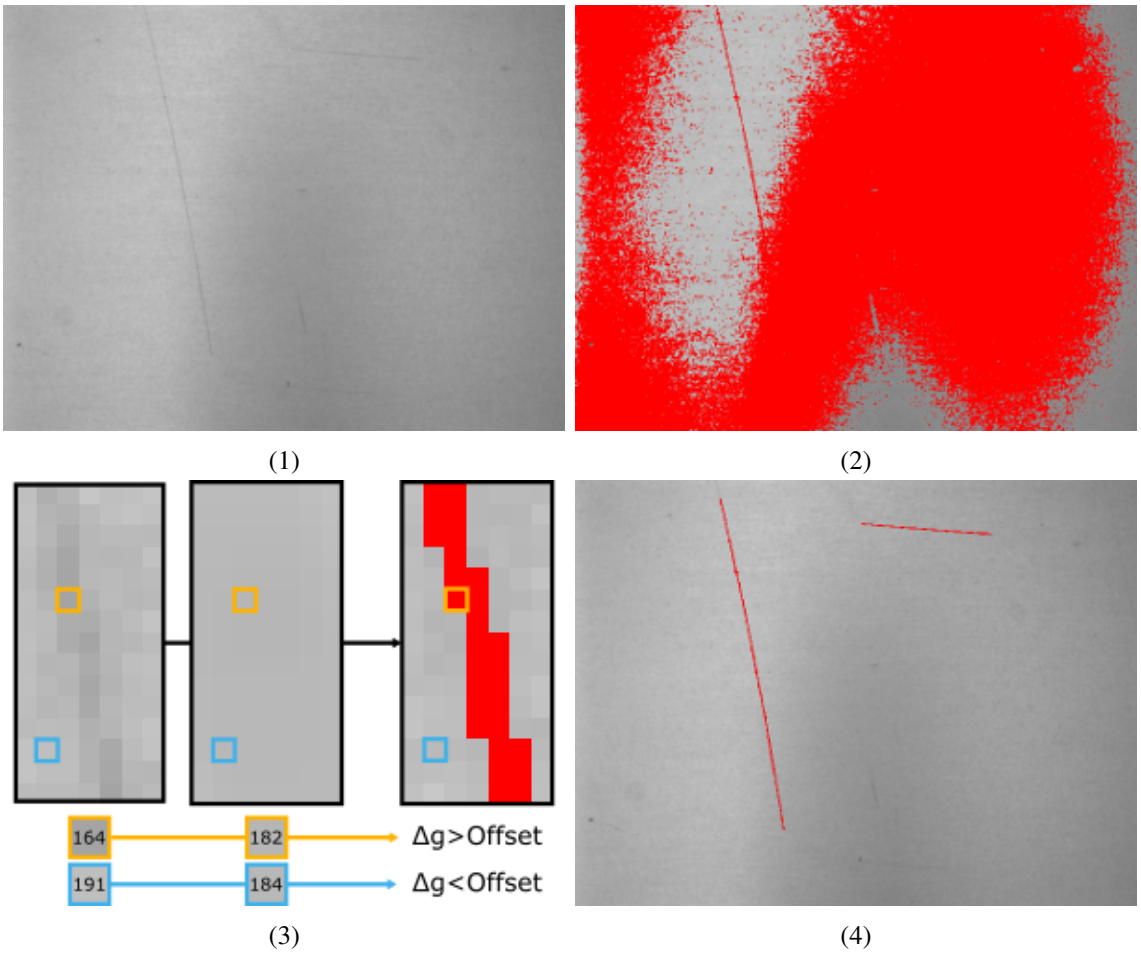


(1) (2) (3) (4)

(1) Eingabebild, (2) Segmentierung mit globalem Schwellenwert gibt nicht den Text allein aus, (3) Schwellenwert wird an lokale Bildmerkmale angepasst, (4) Text kann segmentiert werden.

Der Operator `var_threshold` arbeitet nach einem ähnlichen Prinzip, verwendet allerdings lokale Helligkeit und Standardabweichung als Kriterien zur Segmentierung.

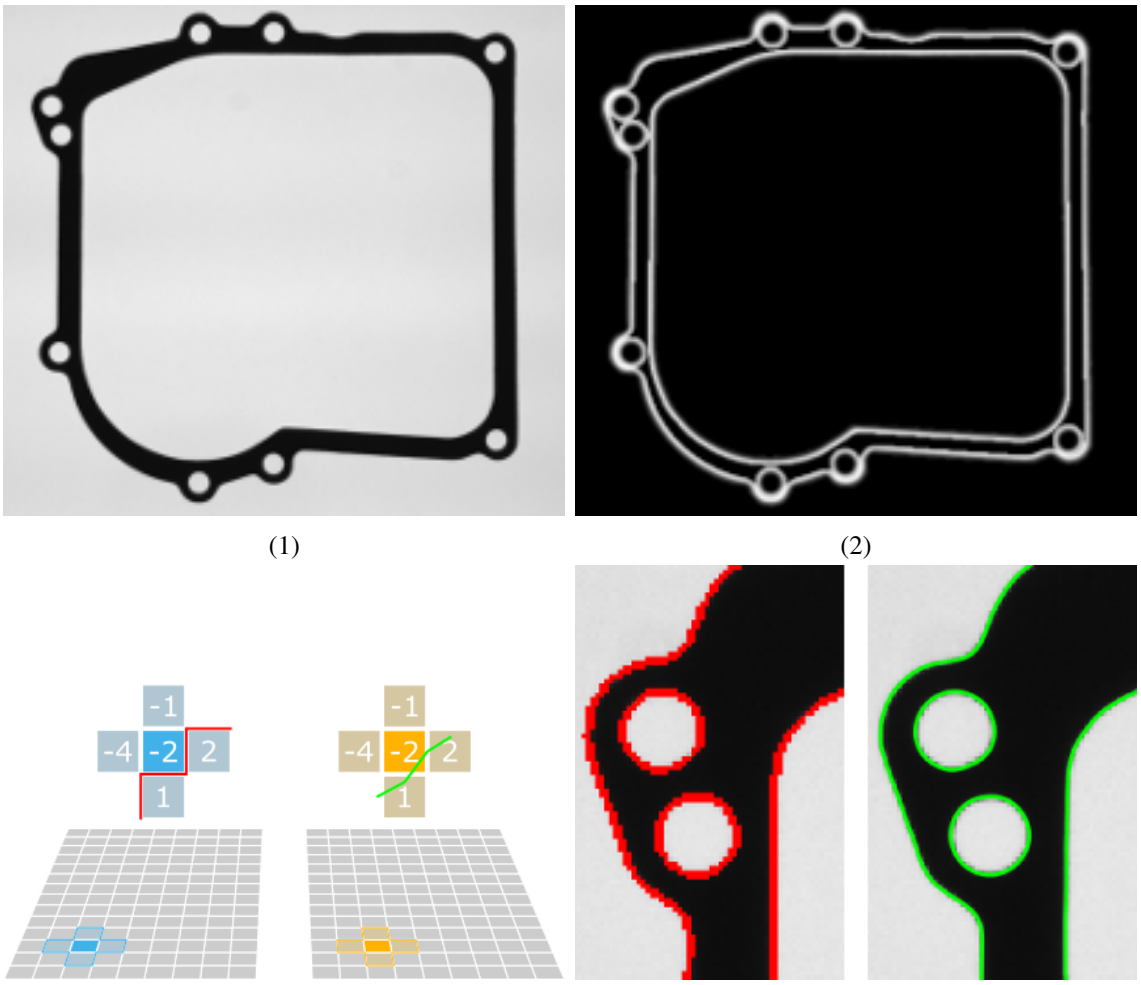
Dynamic Threshold `dyn_threshold` kann verwendet werden, um die Unterschiede zwischen Bildern zu untersuchen. Üblicherweise werden dabei das Eingabebild und eine gefilterte Version des Originals (z.B. das Mittelwertbild) pixelweise verglichen. Der Parameter `LightDark` wird dabei dazu verwendet festzulegen welche Veränderungen im Bild relevant für den Nutzer sind. Die Sensitivität des Operators kann mit dem Parameter `Offset` gesteuert werden.



(1) Eingabebild, (2) Segmentierung von Kratzern mit globalen Schwellenwerten ist aufgrund der unregelmäßigen Belichtungsverhältnisse nicht möglich, (3) Untersuchung der Grauwertdifferenzen korrespondierender Pixel in Originalbild (links) und Mittelwertbild (rechts) zur Segmentierung von Regionen in Abhängigkeit vom Parameter `Offset` (Vergrößerte Bildausschnitte), (4) Ergebnisbild nach Selektion gewünschter Regionen. Für weitere Details steht das Beispiel `surface_scratch.hdev` zur Verfügung.

Um die absoluten Differenzen zwischen zwei Bildern zu untersuchen steht der Operator `check_difference` zur Verfügung. Er ist besonders geeignet für die Detektion von Veränderung in aufeinanderfolgenden Bildaufnahmen.

Zero Crossing Threshold In Bildern, welche durch Operatoren wie `laplace_of_gauss` erzeugt werden, können Kanten an den Nulldurchgängen identifiziert werden. `zero_crossing` und `zero_crossing_sub_pix` liefern diese Kanten unter Berücksichtigung der Vierer-Nachbarschaft jedes Bildpunktes.



(1) Eingabebild, (2) Bildbereich wird reduziert und Laplace-of-Gaussian Filterung wird durchgeführt, (3) Kanten werden an den Nulldurchgängen pixelgenau (rot) bzw. subpixelgenau (grün) identifiziert, (4) Bildausschnitt des Resultats mit `zero_crossing` (rot) bzw. `zero_crossing_sub_pix` (grün).

auto_threshold (Image : Regions : Sigma :)

Segmentieren mit Schwellen aus dem Histogramm.

`auto_threshold` segmentiert ein einkanaliges Bild mittels mehrfacher Schwellenwertsegmentierung. Als erstes wird dabei das absolute Histogramm über den Grauwerten ermittelt. Dann werden im Histogramm relevante Minima gesucht, die schließlich nacheinander als Schwellen für die Schwellenwertsegmentierung dienen. Die Schwellen bestehen für Bilder vom Typ byte aus den Werten 0 und 255 und aus allen Minima des Histogramms (nach der Histogrammglättung mit einem Gaußfilter mit Standardabweichung `Sigma`). Für jedes Grauwertintervall wird *eine* Region erzeugt. Die Anzahl der Regionen ist die Anzahl der Minima + 1. Für Bilder vom Typ uint2 gilt das oben beschriebene analog. Allerdings ist es hier so, dass die höchste Schwelle 65535 beträgt. Weiterhin bezieht sich bei uint2-Bildern der Wert von `Sigma` (virtuell) auf ein Histogramm mit 256 Werten, obwohl intern Histogramme mit einer höheren Auflösung verwendet werden. Dies dient dazu, den Parameter `Sigma` bei der Umstellung zwischen Bildtypen nicht ändern zu müssen. Für float Bilder sind die Schwellenwerte der minimale und maximale Grauwert aus dem Bild sowie alle Minima die aus dem Histogramm extrahiert wurden. Hier bezieht sich der Parameter `Sigma` auf die ursprünglichen Grauwerte. Um so größer der Wert von `Sigma` ist, um so weniger Regionen werden extrahiert. Das Verfahren ist dann einsetzbar, wenn die gesuchten Regionen einheitliche Grauwerte ausweisen (homogene Regionen).

Parameter

▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
Eingabebild.

▷ **Regions** (output_object) . . . . . region-array ⇝ *object*
Regionen mit den Punkten innerhalb der Grauwertintervalle.

▷ **Sigma** (input_control) . . . . . number ⇝ *real* / integer
Sigma für Gaußglättung des Histogramms.
Defaultwert: 2.0
Wertevorschläge: `Sigma` ∈ {0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
Typischer Wertebereich: 0.0 ≤ `Sigma` ≤ 100.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.3
Restriktion: Sigma >= 0.0

Beispiel

```
read_image (Image, 'fabrik')
median_image (Image, Median, 'circle', 3, 'mirrored')
auto_threshold (Median, Seg, 2.0)
connection (Seg, Connected)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`anisotropic_diffusion`, `median_image`, `illuminate`

Nachfolger

`connection`, `select_shape`, `select_gray`

Alternativen

`binary_threshold`, `char_threshold`

Siehe auch

`gray_histo`, `gray_histo_abs`, `histo_to_thresh`, `smooth_funct_1d_gauss`, `threshold`

Modul

Foundation

binary_threshold (Image : Region : Method, LightDark : UsedThreshold)

Segmentieren mit verschiedenen globalen Schwellenwertoperationen.

`binary_threshold` segmentiert ein einkanaliges Bild `Image` mit einem automatisch bestimmten globalen Schwellenwert und gibt die segmentierte `Region` zurück. Dies ist z.B. für die Segmentierung von Buchstaben und Zeichen auf gleichmäßig beleuchtetem Hintergrund geeignet. `binary_threshold` gibt auch den verwendeten Schwellenwert in `UsedThreshold` zurück.

Der verwendete Schwellenwert wird mit der in `Method` angegebenen Methode bestimmt. Derzeit stehen die beiden folgenden Methoden zur Verfügung: *'max_separability'* und *'smooth_histo'*. Beide Verfahren sollten nur auf Bilder angewendet werden, die ein bimodales Histogramm haben.

Die Methode *'smooth_histo'* stellt die Funktionalität zur Verfügung, die vom Operator `bin_threshold` bereitgestellt wurde. Die Methode *'max_separability'* ist weniger empfindlich gegenüber dünnen und isolierten Spitzen im Histogramm. Außerdem tendiert diese Methode dazu, Werte für `UsedThreshold` zu bestimmen, die näher am Schwerpunkt des Histogramms liegen. In vielen Fällen ist die Methode *'max_separability'* schneller als *'smooth_histo'*.

Maximierung der Trennbarkeit

Bei der Wahl von *'max_separability'* als Verfahren in `Method` wird eine automatische Bestimmung eines globalen Schwellenwertes nach dem Verfahren von Otsu (siehe Referenzen) durchgeführt.

Der Algorithmus berechnet zuerst das Histogramm des Bildes und verwendet dann statistische Momente um die optimale Schwelle T^* zu bestimmen. Die Schwelle trennt die Pixel so in Vordergrund und Hintergrund, dass die Streuung zwischen den beiden Klassen maximiert wird.

- Für `LightDark` = *'light'* werden alle Pixel mit Grauwert größer gleich T^* ausgewählt.
- Für `LightDark` = *'dark'* werden alle Pixel mit Grauwert kleiner als T^* ausgewählt.

Das Verfahren 'max_separability' ist nur für byte- und uint2-Bilder verfügbar.

Verfahren mit Glättung des Histogramms

Bei der Wahl von *'smooth_histo'* als Verfahren in `Method` wird der optimale Schwellenwert (r, c) wie folgt bestimmt: Als erstes wird dabei das relative Histogramm über den Grauwerten ermittelt. Dann werden im Histogramm relevante Minima gesucht, die als Schwellen für die Schwellenwertsegmentierung dienen. Um die Anzahl der Minima zu reduzieren, wird das Histogramm wie bei `auto_threshold` mit einer Gaußmaske geglättet. Die Maske wird dabei so lange vergrößert, bis nur noch ein Minimum vorhanden ist.

- Für `LightDark` = *'light'* wird jedes Pixel $p(r, c)$ dessen Grauwert größer gleich der lokalen Schwelle $T(r, c)$ ist selektiert.
- Für `LightDark` = *'dark'* wird jedes Pixel $p(r, c)$ dessen Grauwert kleiner als die lokale Schwelle $T(r, c)$ ist selektiert.

Parameter

▷ **`Image`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2
Eingabebild.

▷ **`Region`** (output_object) ..... region(-array) ⇝ *object*
Segmentierte Ausgaberegion.

▷ **`Method`** (input_control) ..... string ⇝ *string*
Auswahl des Verfahrens.
Defaultwert: 'max_separability'
Werteliste: `Method` ∈ {'max_separability', 'smooth_histo'}

▷ **`LightDark`** (input_control) ..... string ⇝ *string*
Segmentierung des Vordergrunds oder des Hintergrunds.
Defaultwert: 'dark'
Werteliste: `LightDark` ∈ {'dark', 'light'}

▷ **`UsedThreshold`** (output_control) ..... number(-array) ⇝ *integer* / string
Angewandte Schwelle.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`connection`, `select_shape`, `select_gray`

Alternativen

`auto_threshold`, `char_threshold`, `local_threshold`

Siehe auch

`gray_histo`, `threshold`

Literatur

N. Otsu, „A threshold selection method from gray level histograms", IEEE Trans. Syst. Man. Cybern., Vol. SMC-9, 62-66 (1979)

Modul

Foundation

char_threshold (Image, HistoRegion : Characters : Sigma, Percent : Threshold)

Segmentieren mit Schwellen für Texte.

char_threshold ist darauf spezialisiert, in einkanaligen Bildern dunkle Schriftzeichen von hellem Hintergrund zu segmentieren. Die Vorgehensweise ist dabei wie folgt: Zuerst wird das Histogramm der Grauwerte des Bildes Image für die Punkte der Region HistoRegion ermittelt und zur Rauschunterdrückung mit einer Gaußmaske (Parameter Sigma) geglättet. In diesem Histogramm kennzeichnet das Maximum den Hintergrund (weißes Papier), während die Schriftzeichen ein kleines lokales Maximum bei niedrigen Grauwerten verursachen. Im Gegensatz zum Operator binary_threshold (mit Method='smooth_histo') wird nun nicht nach dem Minimum zwischen den beiden Maxima gesucht; der Schwellenwert wird stattdessen in Relation zum Maximum des Histogramms, also zum Hintergrund, nach folgender Bedingung bestimmt:

$$Histogramm[\text{Schwellenwert}] * 100.0 < Histogramm[\text{Maximum}] * (100.0 - \texttt{Percent})$$

Falls z.B. Percent = 95 gewählt wird, sucht der Operator den Grauwert, dessen Häufigkeit höchstens 5 Prozent der maximalen Häufigkeit beträgt. Da char_threshold annimmt, dass die Schriftzeichen dunkler sind als der Hintergrund, wird der Schwellenwert links vom Maximum gesucht.

Im Vergleich zu binary_threshold sollte char_threshold dann eingesetzt werden, wenn kein deutliches Minimum zwischen den beiden zu den Schriftzeichen bzw. zum Hintergrund gehörenden Maxima besteht oder die Schriftzeichen gar kein lokales Maximum verursachen. Dies tritt z.B. bei ungleichmäßiger Beleuchtung auf, oder wenn das Bild nur wenige Schriftzeichen enthält.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
  Eingabebild.
- ▷ **HistoRegion** (input_object) ..... region ⇝ *object*
  Region, in der das Histogramm berechnet wird.
- ▷ **Characters** (output_object) ..... region(-array) ⇝ *object*
  Dunkle Bereiche des Bildes (Buchstaben).
- ▷ **Sigma** (input_control) ..... number ⇝ *real*
  Sigma für Glättung des Histogramms.
  Defaultwert: 2.0
  Wertevorschläge: Sigma ∈ {0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich: 0.0 ≤ Sigma ≤ 50.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.2
- ▷ **Percent** (input_control) ..... number ⇝ *real* / integer
  Prozentwert für Grauwertunterschied.
  Defaultwert: 95
  Wertevorschläge: Percent ∈ {90, 92, 95, 96, 97, 98, 99, 99.5, 100}
  Typischer Wertebereich: 0.0 ≤ Percent ≤ 100.0 (lin)
  Minimale Schrittweite: 0.1
  Empfohlene Schrittweite: 0.5
- ▷ **Threshold** (output_control) ..... integer(-array) ⇝ *integer*
  Verwendeter Schwellenwert.

Beispiel

```
read_image (Image, 'letters')
char_threshold (Image, Image, Seg, 0.0, 5.0, Threshold)
connection (Seg, Connected)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

anisotropic_diffusion, median_image, illuminate

Nachfolger

connection, select_shape, select_gray

Alternativen

binary_threshold, auto_threshold, gray_histo, smooth_funct_1d_gauss, threshold

Modul

Foundation

check_difference (Image, Pattern : Selected : Mode, DiffLowerBound, DiffUpperBound, GrayOffset, AddRow, AddCol :)

Pixelweises Vergleichen zweier Bilder.

check_difference wählt aus dem Eingabebild die Bildpunkte ($g_o = g_{\texttt{Image}}$) aus, deren Grauwertabweichung von den entsprechenden Pixel in Pattern abhängig vom Parameter Mode innerhalb bzw. außerhalb des vorgegebenen Werteintervalls [DiffLowerBound, DiffUpperBound] liegt. Sei g_p der Grauwert aus Pattern, der gegenüber g_o um den Vektor (AddRow, AddCol) verschoben ist.

Dann werden im Modus *'diff_inside'* die Pixel g_o selektiert mit

$$g_o - g_p - \texttt{GrayOffset} \geq \texttt{DiffLowerBound} \quad \text{und}$$
$$g_o - g_p - \texttt{GrayOffset} \leq \texttt{DiffUpperBound}$$

und im Modus *'diff_outside'* die g_o mit

$$g_o - g_p - \texttt{GrayOffset} > \texttt{DiffLowerBound} \quad \text{oder}$$
$$g_o - g_p - \texttt{GrayOffset} < \texttt{DiffUpperBound}$$

Dieser Test wird für alle Bildpunkte aus dem Definitionsbereich von Image, geschnitten mit dem verschobenen Definitionsbereich von Pattern, durchgeführt. Alle Punkte, die die obige Bedingung erfüllen, werden in der Ausgaberegion gespeichert. Die beiden Bilder können unterschiedliche Größe haben. Typischerweise ist Pattern kleiner als Image.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
Eingabebild.

▷ **Pattern** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte
Vergleichsbild.

▷ **Selected** (output_object) ..... region(-array) ⇝ *object*
Punkte, in denen sich die Bilder ähneln/unterscheiden.

▷ **Mode** (input_control) ..... string ⇝ *string*
Modus: bestimme ähnliche oder verschiedene Pixel.
Defaultwert: 'diff_outside'
Wertevorschläge: Mode ∈ {'diff_inside', 'diff_outside'}

▷ **DiffLowerBound** (input_control) ..... number ⇝ *integer*
Untere Grenze der erlaubten Grauwertdifferenz.
Defaultwert: -5
Wertevorschläge: DiffLowerBound ∈ {0, -1, -2, -3, -5, -7, -10, -12, -15, -17, -20, -25, -30}
Typischer Wertebereich: -255 ≤ DiffLowerBound ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2
Restriktion: -255 <= DiffLowerBound && DiffLowerBound <= 255

▷ **DiffUpperBound** (input_control) ..... number ⇝ *integer*
Obere Grenze der erlaubten Grauwertdifferenz.
Defaultwert: 5
Wertevorschläge: `DiffUpperBound` ∈ {0, 1, 2, 3, 5, 7, 10, 12, 15, 17, 20, 25, 30}
Typischer Wertebereich: -255 ≤ `DiffUpperBound` ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2
Restriktion: -255 <= DiffUpperBound && DiffUpperBound <= 255

▷ **GrayOffset** (input_control) ..... number ⇝ *integer*
Korrekturgrauwert, der im Eingabebild abgezogen wird.
Defaultwert: 0
Wertevorschläge: `GrayOffset` ∈ {-30, -25, -20, -17, -15, -12, -10, -7, -5, -3, -2, -1, 0, 1, 2, 3, 5, 7, 10, 12, 15, 17, 20, 25, 30}
Typischer Wertebereich: -255 ≤ `GrayOffset` ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2
Restriktion: -255 <= GrayOffset && GrayOffset <= 255

▷ **AddRow** (input_control) ..... point.y ⇝ *integer*
Zeilenkoordinate, um die das Vergleichsbild verschoben wird.
Defaultwert: 0
Wertevorschläge: `AddRow` ∈ {-200, -100, -20, -10, 0, 10, 20, 100, 200}
Typischer Wertebereich: -32000 ≤ `AddRow` ≤ 32000 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **AddCol** (input_control) ..... point.x ⇝ *integer*
Spaltenkoordinate, um die das Vergleichsbild verschoben wird.
Defaultwert: 0
Wertevorschläge: `AddCol` ∈ {-200, -100, -20, -10, 0, 10, 20, 100, 200}
Typischer Wertebereich: -32000 ≤ `AddCol` ≤ 32000 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Komplexität

Sei A die Anzahl der gültigen Pixel, dann ist die Laufzeitkomplexität: $O(A)$.

Ergebnis

`check_difference` liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Nachfolger

`connection`, `select_shape`, `reduce_domain`, `select_gray`, `rank_region`, `dilation1`, `opening`

Alternativen

`sub_image`, `dyn_threshold`

Modul

Foundation

dual_threshold (Image : RegionCrossings : MinSize, MinGray, Threshold :)

Segmentieren mit Schwellen für Bilder mit Vorzeichen.

dual_threshold segmentiert das Eingabebild in eine Teilregion mit Grauwerten $\geq$Threshold („positive“ Bildbereiche) und eine mit Grauwerten $\leq -$Threshold („negative“ Bildbereiche). Nur „positive“ oder „negative“ Teilbereiche mit einer Fläche größer als MinSize werden dabei berücksichtigt. Teilbereiche, deren maximaler Grauwert betragsmäßig kleiner ist als MinGray, werden unterdrückt.

Die durchgeführte Segmentierung ist nicht vollständig, d.h. die „positiven“ und „negativen“ Ergebnisregionen überdecken zusammengenommen nicht unbedingt das ganze Eingabebild. Es bleiben alle Bildbereiche unberücksichtigt, deren Grauwerte zwischen $-$Threshold und Threshold (bzw. $-$MinGray und MinGray) liegen.

dual_threshold schließt sich typischerweise an die Anwendung von Laplace-Operatoren wie (laplace, laplace_of_gauss, derivate_gauss oder diff_of_gauss) oder eine Bildsubtraktion (sub_image) an. In den Nulldurchgängen des Laplace-Operators spiegeln sich Bildkanten wider, die sich als Trennlinien zwischen den „positiven“ und „negativen“ Bildbereichen bestimmen lassen. Dazu ruft man zuerst dual_threshold mit Threshold = 1 auf und erzeugt dann die komplementäre Regionen mittels complement. Der Parameter MinGray steuert dabei die Rauschinvarianz,während MinSize das Auflösungsvermögen der Kantendetektion steuert.

Bei Pixeltypen ohne Vorzeichen (byte) wird nur die positive Hälfte der Segmentierung durchgeführt. dual_threshold verhält sich also wie eine einfaches Schwellenwertverfahren (threshold) mit nachgeschaltetem connection und select_gray.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / int1 / int2 / int4 / real
  Eingabebild.
- ▷ **RegionCrossings** (output_object) ..... region-array ⇝ *object*
  „Positive“ und „negative“ Teilregionen.
- ▷ **MinSize** (input_control) ..... integer ⇝ *integer*
  Teilregionen mit weniger als MinSize Fläche werden unterdrückt.
  Defaultwert: 20
  Wertevorschläge: MinSize $\in$ {0, 10, 20, 50, 100, 200, 500, 1000}
  Typischer Wertebereich: $0 \leq$ MinSize $\leq$ 10000 (lin)
  Minimale Schrittweite: 1
  Empfohlene Schrittweite: 10
- ▷ **MinGray** (input_control) ..... real ⇝ *real*
  Teilregionen, deren maximaler Grauwert betragsmäßig kleiner als MinGray ist, werden unterdrückt.
  Defaultwert: 5.0
  Wertevorschläge: MinGray $\in$ {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 20.0}
  Typischer Wertebereich: $0.001 \leq$ MinGray $\leq$ 10000.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 10.0
  Restriktion: MinGray > 0
- ▷ **Threshold** (input_control) ..... real ⇝ *real*
  Zurückgeliefert werden Teilregionen, deren Grauwerte größer gleich (bzw. kleiner gleich) Threshold (-Threshold) sind.
  Defaultwert: 2.0
  Wertevorschläge: Threshold $\in$ {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 20.0}
  Typischer Wertebereich: $0.001 \leq$ Threshold $\leq$ 10000.0 (lin)
  Minimale Schrittweite: 1.0
  Empfohlene Schrittweite: 10.0
  Restriktion: Threshold >= 1 && Threshold <= MinGray

Beispiel

```
* Edge detection with the Laplace operator (and edge thinning)
diff_of_gauss(Image,Laplace,2.0,1.6)
* find "`positive"' and "`negative"' regions:
dual_threshold(Laplace,Region,20,2,1)
* The zero runnings are the complement to these image section:
complement(Region,ZeroCrossings)

* Simulation of dual_threshold
```

```
dual_threshold(Laplace,Result,MinS,MinG,Threshold)
threshold(Laplace,Tmp1,Threshold,999999)
connection(Tmp1,Tmp2)
select_shape(Tmp2,Tmp3,'area','and',MinS,999999)
select_gray(Laplace,Tmp3,Tmp4,'max','and',MinG,999999)
threshold(Laplace,Tmp5,-999999,-Threshold)
connection(Tmp5,Tmp6)
select_shape(Tmp6,Tmp7,'area','and',MinS,999999)
select_gray(Laplace,Tmp7,Tmp8,'min','and',-999999,-MinG)
concat_obj(Tmp4,Tmp8,Result)
```

Ergebnis

`dual_threshold` liefert normalerweise den Wert TRUE. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`min_max_gray`, `sobel_amp`, `binomial_filter`, `gauss_filter`, `reduce_domain`, `diff_of_gauss`, `sub_image`, `derivate_gauss`, `laplace_of_gauss`, `laplace`, `expand_region`

Nachfolger

`connection`, `dilation1`, `erosion1`, `opening`, `closing`, `rank_region`, `shape_trans`, `skeleton`

Alternativen

`threshold`, `dyn_threshold`, `check_difference`

Siehe auch

`connection`, `select_shape`, `select_gray`

Modul

Foundation

dyn_threshold (OrigImage, ThresholdImage : RegionDynThresh : Offset, LightDark :)

Segmentieren mit lokalem Schwellenwert.

`dyn_threshold` wählt aus den Eingabebildern die Bildpunkte (g_o = `g_OrigImage`) aus, die der Schwellenwertbedingung genügen. Sei g_t = `g_ThresholdImage` dann lautet diese Bedingung bei `LightDark` = *'light'*:

$$g_o \geq g_t + \texttt{Offset}$$

Bei `LightDark` = *'dark'* lautet sie:

$$g_o \leq g_t - \texttt{Offset}$$

Bei `LightDark` = *'equal'*:

$$g_t - \texttt{Offset} \leq g_o \leq g_t + \texttt{Offset}$$

Und bei `LightDark` = *'not_equal'*:

$$g_t - \texttt{Offset} > g_o \vee g_o > g_t + \texttt{Offset}$$

Typischerweise werden als Vergleichskomponenten das Original und eine geglättete Version eines Bildes verwendet (z.B. `mean_image`, `binomial_filter`, `gauss_filter`, etc.). Dann entspricht `dyn_threshold` in seiner Wirkung etwa der Anwendung von `threshold` auf ein hochpassgefiltertes Bild (`highpass_image`).

Es lassen sich Objektkanten finden, wobei die Größe (Durchmesser) der gesuchten Objekte durch die Maskengröße des Tiefpassfilters und die Steigung, bzw. Amplitude der Kante festgelegt wird.

Je größer der Maskendurchmesser gewählt wird, desto größer sind die gefundenen Regionen. Als Faustregel sollte die Maskengröße etwa doppelt so groß wie der Durchmesser der gesuchten Objekte sein. Es ist wichtig, den Parameter `Offset` nicht auf Null zu setzen, da dann zu viele kleine Regionen gefunden werden (Rauschen). Sinnvoll sind Werte, die betragsmäßig zwischen 5 und 40 liegen. Je größer `Offset` gewählt wird, desto weniger Punkte werden gefunden und umso kleiner werden die Regionen.

Alle Punkte eines Eingabebildes, die die obige Bedingung erfüllen, werden gemeinsam als eine Region abgespeichert. Die Zusammenhangskomponenten können gegebenenfalls mit `connection` berechnet werden.

Achtung

Wird `Offset` auf $-1 \ldots 1$ gesetzt, dann wird i.a. eine sehr verrauschte Region erzeugt, die viel Speicherplatz benötigt. Wird `Offset` zu groß gewählt (etwa > 60), dann kann es vorkommen, dass kein Punkt die Schwellenwertbedingung erfüllt (also eine leere Region berechnet wird). Wird `Offset` zu klein gewählt (etwa $<$ -60) dann können alle Punkte die Schwellenwertbedingung erfüllen. Es wird also eine maximale Region erzeugt.

Parameter

- ▷ **`OrigImage`** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Eingabebild.
- ▷ **`ThresholdImage`** (input_object) .... singlechannelimage(-array) ⇝ *object :* byte / int2 / uint2 / int4 / real
  Bild mit den lokalen Werten.
- ▷ **`RegionDynThresh`** (output_object) ..... region(-array) ⇝ *object*
  Segmentierte Region.
- ▷ **`Offset`** (input_control) ..... number ⇝ *real* / integer
  Schwellenwert für die Segmentierung.
  Defaultwert: 5.0
  Wertevorschläge: `Offset` ∈ {1.0, 3.0, 5.0, 7.0, 10.0, 20.0, 30.0}
  Typischer Wertebereich: -255.0 ≤ `Offset` ≤ 255.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 5
  Restriktion: -255 < Offset && Offset < 255
- ▷ **`LightDark`** (input_control) ..... string ⇝ *string*
  Helle,dunkle oder ähnliche Bereiche suchen.
  Defaultwert: 'light'
  Werteliste: `LightDark` ∈ {'dark', 'light', 'equal', 'not_equal'}

Beispiel

```
* Looking for regions with the diameter D
mean_image(Image,Mean,D*2+1,D*2+1)
dyn_threshold(Image,Mean,Seg,5,'light')
connection(Seg,Regions)
```

Komplexität

Sei A die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität $O(A)$

Ergebnis

`dyn_threshold` liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

- Automatisch parallelisiert auf Domainebene.

Vorgänger

`mean_image`, `smooth_image`, `binomial_filter`, `gauss_filter`

Nachfolger

`connection`, `select_shape`, `reduce_domain`, `select_gray`, `rank_region`, `dilation1`, `opening`, `erosion1`

Alternativen

`check_difference`, `threshold`

Siehe auch

`highpass_image`, `sub_image`

Modul

Foundation

`fast_threshold` (`Image` : `Region` : `MinGray`, `MaxGray`, `MinSize` :)

Schnelles Segmentieren mit globalen Schwellenwerten.

`fast_threshold` wählt aus den Eingabebildern die Bildpunkte aus, deren Grauwerte g der Schwellenwertbedingung

$$\texttt{MinGray} \leq g \leq \texttt{MaxGray}$$

genügen. Für eine schnellere Bearbeitung erfolgt die Auswahl der Punkte in zwei Schritten: Zunächst werden alle Punkte bestimmter Zeilen, welche durch ihren Abstand festgelegt werden, untersucht. Danach wird in der Umgebung $((2 * MinSize + 1) \times (2 * MinSize + 1))$ aller so gefundenen Punkte eine genauere Segmentierung durchgeführt.

Achtung

Auf Mehrkernprozessoren, die den SSE2 Befehlssatz unterstützen, ist `threshold` meist schneller als `fast_threshold`. `fast_threshold` sollte nur dann `threshold` vorgezogen werden, wenn diese Features nicht verfügbar sind, zum Beispiel in eingebetteten Systemen.

Parameter

▷ **`Image`** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / uint2 / direction / cyclic / real
Eingabebild.

▷ **`Region`** (output_object) . . . . . region(-array) ⇝ *object*
Segmentierte Region.

▷ **`MinGray`** (input_control) . . . . . number ⇝ *real* / integer
Untere Schwelle für die Grauwerte.
Defaultwert: 128
Wertevorschläge: `MinGray` ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}
Typischer Wertebereich: 0.0 ≤ `MinGray` ≤ 255.0 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5.0

▷ **`MaxGray`** (input_control) . . . . . number ⇝ *real* / integer
Obere Schwelle für die Grauwerte.
Defaultwert: 255.0
Wertevorschläge: `MaxGray` ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}
Typischer Wertebereich: 0.0 ≤ `MaxGray` ≤ 255.0 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5.0

▷ **MinSize** (input_control) . . . . . number ⇝ *integer*
Mindestgröße eines Objektes.
Defaultwert: 20
Wertevorschläge: MinSize ∈ {5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 100}
Typischer Wertebereich: 2 ≤ MinSize ≤ 200 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2

Komplexität

Sei A die Fläche der Ausgaberegion und $height$ die Höhe von Image, dann ist die Laufzeitkomplexität $O(A + height/\text{MinSize})$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert fast_threshold den Wert TRUE. Für das Verhalten bzgl. der Ein- und Ausgabebilder sind die Flags 'no_object_result', 'empty_region_result' und 'store_empty_region' einstellbar (siehe set_system). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

histo_to_thresh, min_max_gray, sobel_amp, binomial_filter, gauss_filter, reduce_domain, fill_interlace

Nachfolger

connection, dilation1, erosion1, opening, closing, rank_region, shape_trans, skeleton

Alternativen

threshold, gen_grid_region, dilation_rectangle1, dyn_threshold

Siehe auch

class_2dim_sup, hysteresis_threshold

Modul

Foundation

histo_to_thresh (: : Histogramm, Sigma : MinThresh, MaxThresh)

Berechnen von Grauwertschwellen aus einem Histogramm.

histo_to_thresh berechnet aus einem Histogramm Schwellen für eine Segmentierung mit threshold. Als Schwellen werden die Werte 0 und der Maximale Grauwert im Histogramm, sowie alle Minima des Histogramms verwendet. Vor der Berechnung der Schwellen wird das Histogramm mit einer Gaußfunktion geglättet.

histo_to_thresh kann die mit gray_histo erzeugten relativen und absoluten Histogramme verarbeiten. Allerdings sollten hierbei nur byte-Bilder verwendet werden, da sonst die zurückgelieferten Schwellen nicht ohne weiteres auf die entsprechenden Schwellen des Bildes umgerechnet werden können. Für die Bildtypen uint2 sollten die Histogramme mit gray_histo_abs berechnet werden, da dann die Umrechnung auf die Schwellen einfach durch Multiplikation mit der gewählten Quantisierung erfolgen kann. Bei uint2-Bildern ist zu beachten, dass die Quantisierung so gewählt werden sollte, dass das Histogramm noch aussagekräftige Information enthält. So hat z.B. ein 640×480 Bild mit 16 Bit Grauwertauflösung im Mittel nur $307200/65536 = 4.7$ Einträge in jeder Histogrammzelle, d.h. das Histogramm ist zu dünn besetzt, um sinnvolle Statistiken daraus ableiten zu können. Um aus einem solchen Histogramm sinnvolle Schwellen extrahieren zu können, müsste Sigma extrem groß gewählt werden, was zu sehr hohen Laufzeiten und numerischen Problemen führen würde. Die Quantisierung in gray_histo_abs sollte im Normalfall also so gewählt werden, dass das Histogramm maximal 1024 Einträge enthält. Bei Bildern mit mehr als 10 Bit pro Pixel muss die Quantisierung also größer als 1 gewählt werden. Das

mit gray_histo_abs berechnete Histogramm ist weiterhin auf die Teile zu beschränken, die signifikante Information tragen. So sollte z.B. bei einem Bild mit 12 Bit pro Pixel die Quantisierung auf 4 gesetzt werden. Von dem berechneten Histogramm (das in diesem Fall immer noch 16384 Werte enthält), sollten die ersten 1024 Werte als Eingabe für histo_to_thresh ausgewählt werden. Schließlich muss MinThresh noch mit 4 (der Quantisierung) multipliziert werden, während MaxThresh mit 4 multipliziert und um 3 (der Quantisierung minus 1) erhöht werden muss.

Parameter

- ▷ **Histogramm** (input_control) ..... histogram-array ⇝ *integer / real*
  Grauwerthistogramm.
- ▷ **Sigma** (input_control) ..... number ⇝ *real*
  Sigma für Glättung des Histogramms.
  Defaultwert: 2.0
  Wertevorschläge: Sigma ∈ {0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich: 0.5 ≤ Sigma ≤ 30.0 (lin)
  Minimale Schrittweite: 0.01
  Empfohlene Schrittweite: 0.2
- ▷ **MinThresh** (output_control) ..... integer-array ⇝ *integer*
  Minimale Schwellen.
- ▷ **MaxThresh** (output_control) ..... integer-array ⇝ *integer*
  Maximale Schwellen.

Beispiel

```
* Calculate thresholds from a byte image and threshold the image.
gray_histo (Image, Image, AbsoluteHisto, RelativeHisto)
histo_to_thresh (AbsoluteHisto, 4, MinThresh, MaxThresh)
threshold (Image, Region, MinThresh, MaxThresh)

* Calculate thresholds from a 12 bit uint2 image and threshold the image.
gray_histo_abs (Image, Image, 4, AbsoluteHisto)
AbsoluteHisto := AbsoluteHisto[0:1023]
histo_to_thresh (AbsoluteHisto, 16, MinThresh, MaxThresh)
MinThresh := MinThresh*4
MaxThresh := MaxThresh*4+3
threshold (Image, Region, MinThresh, MaxThresh)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gray_histo

Nachfolger

threshold

Siehe auch

auto_threshold, binary_threshold, char_threshold

Modul

Foundation

local_threshold (Image : Region : Method, LightDark, GenParamName, GenParamValue :)

Segmentieren mit lokalen Schwellenwertoperationen.

local_threshold segmentiert ein einkanaliges Bild Image mit dem in Method angegebenen Verfahren und gibt die segmentierte Region zurück. Der Operator bietet momentan das Verfahren *'adapted_std_deviation'* an. Dieses Verfahren ist ein lokales Schwellenwertverfahren und liefert gute Ergebnisse auf Bildern von Dokumenten.

'adapted_std_deviation'

Bei der Wahl von *'adapted_std_deviation'* als Verfahren in Method wird ein lokal adaptives Schwellenwertverfahren basierend auf lokalem Mittelwert und Standardabweichung durchgeführt. Dieses Verfahren kann zum Beispiel Bilder von Dokumenten segmentieren, die durch Veränderungen in der Beleuchtung, durch Rauschen oder andere Effekte beeinflusst wurden. Es erlaubt die Segmentierung von Text auf inhomogenem Hintergrund durch Berücksichtigung des lokalen Kontrastes. Für die Segmentierung des dunklen Vordergrunds (siehe Parameter LightDark) wird für ein Pixel an Position (r, c) die lokale Schwelle $T(r, c)$ in einem Fenster der Größe mask_size×mask_size (siehe *'mask_size'*) wie folgt berechnet:

$$T(r,c) = \mu(r,c)\left(1 + k\left(\frac{\sigma(r,c)}{R} - 1\right)\right)$$

Hier ist $\mu(r, c)$ der lokale Mittelwert innerhalb des Fensters und $\sigma(r, c)$ die entsprechende Standardabweichung. Der Parameter R (siehe *'range'*) ist der maximal erwartete Wert der Standardabweichung (R= 128 für byte-Bilder) und k (siehe *'scale'*) der Parameter, der kontrolliert wie stark sich der Schwellenwert $T(r, c)$ vom Mittelwert $\mu(r, c)$ unterscheidet.

Bei hohem Kontrast in der Nachbarschaft des Punktes (r, c) hat die Standardabweichung $\sigma(r, c)$ einen Wert nahe bei R, was zu einer Schwelle $T(r, c)$ nahe dem lokalen Mittelwert $\mu(r, c)$ führt. Bei niedrigem Kontrast ist die lokale Schwelle kleiner als der lokale Mittelwert. Für dunklen Text auf hellem Hintergrund der auch dunkle Regionen enthält, erlaubt dies die Segmentierung des Textes auch in den dunkleren Bereichen.

Der Parameter LightDark steuert, ob helle oder dunkle Strukturen segmentiert werden sollen.

- Für LightDark = *'dark'* werden dunkle Strukturen vor einem hellen Hintergrund segmentiert. Jedes Pixel $p(r, c)$ dessen Grauwert größer als die lokale Schwelle $T(r, c)$ ist wird selektiert.
- Für LightDark = *'light'* werden helle Strukturen vor einem dunklen Hintergrund segmentiert. Das Ergebnis ist im wesentlichen das gleiche, wie wenn man das Bild invertieren würde und dann den Parameter LightDark auf *'dark'* setzen würde.

Mit Hilfe der generischen Parameter GenParamName und GenParamValue können folgende Parameter des Verfahrens beeinflusst werden:

'mask_size': bestimmt die Maskengröße, also die Größe der Nachbarschaft, in der die lokale Schwelle bestimmt wird. Je kleiner die Maskengröße ist, umso dünner werden segmentierte Strukturen. *'mask_size'* sollte größer als die Strichbreite der zu segmentierenden Zeichen oder Strukturen gesetzt werden. Ist *'mask_size'* gerade, wird die nächst höhere ungerade Größe verwendet.
Vorgeschlagene Werte: *15* (default), *21*, *31*

'scale': bestimmt den Parameter k ($0 \leq k$), der kontrolliert wie stark der Schwellenwert vom lokalen Mittelwert abweicht. Kleinere Werte für *'scale'* führen dazu, dass auch Strukturen segmentiert werden, die einen geringeren Kontrast zum Hintergrund aufweisen. Größere Werte führen zur Unterdrückung von Artefakten.
Vorgeschlagene Werte: *0.2* (default), *0.3*, *0.5*

'range': bestimmt den maximalen Wert der Standardabweichung R. Dieser Wert sollte an den erwarteten Grauwertbereich angepasst werden. Als Faustregel kann der Wert für *'range'* auf $range = 0.5 \cdot (MaxGray - MinGray)$ gesetzt werden, wobei MinGray und MaxGray die minimalen und maximalen Grauwerte sind, die mit min_max_gray bestimmt werden können.
Vorgeschlagene Werte: *128* (default) für byte-Bilder, *32767.5* (default) für uint2-Bilder

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2
Eingabebild.

▷ **Region** (output_object) ..... region(-array) ⇝ *object*
Segmentierte Ausgaberegion.

▷ **Method** (input_control) ..... string ⇝ *string*
Auswahl des Verfahrens.
Defaultwert: 'adapted_std_deviation'
Werteliste: Method ∈ {'adapted_std_deviation'}

▷ **LightDark** (input_control) ..... string ⇝ *string*
Segmentierung des Vordergrunds oder des Hintergrunds.
Defaultwert: 'dark'
Werteliste: LightDark ∈ {'dark', 'light'}

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Liste der generischen Parameter.
Defaultwert: []
Werteliste: GenParamName ∈ {'mask_size', 'scale', 'range'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *integer* / real
Liste der Werte für die generischen Parameter.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0.2, 15, 30, 128.0}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

connection, select_shape, select_gray

Alternativen

auto_threshold, binary_threshold, char_threshold

Siehe auch

gray_histo, threshold

Literatur

J. Sauvola, M. Pietikäinen, „Adaptive document image binarization", Pattern Recognition, 33, 225-236 (2000)

Modul

Foundation

threshold (Image : Region : MinGray, MaxGray :)

Segmentieren mit globalen Schwellenwerten.

threshold wählt aus den Eingabebildern die Bildpunkte aus, deren Grauwerte g der Schwellenwertbedingung

$$\texttt{MinGray} \leq g \leq \texttt{MaxGray}$$

genügen.

Alle Punkte eines Eingabebildes, die die Bedingung erfüllen, werden gemeinsam als eine neue Region abgespeichert. Wird mehr als ein Grauwertbereich übergeben (Tupel von Werten für MinGray und MaxGray), dann wird für jedes dieser Intervalle eine Region erzeugt. Für Vektorfeldbilder wird der Schwellenwert nicht auf Grauwerte sondern auf die Länge der Vektoren angewandt. Um die Grenzen nach unten oder oben offen zu lassen, kann statt eines Wertes auch *'min'* beziehungsweise *'max'* für MinGray und MaxGray gesetzt werden.

Achtung

Für ganzzahlige Bildtypen werden Fließkommawerte in MinGray und MaxGray abgeschnitten.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / vector_field
Eingabebild.

▷ **Region** (output_object) ..... region(-array) ⇝ *object*
Segmentierte Region.

▷ **MinGray** (input_control) ..... number(-array) ⇝ *real* / integer / string
Untere Schwelle für die Grauwerte oder *'min'*.
Defaultwert: 128.0
Wertevorschläge: MinGray ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0, 'min'}

▷ **MaxGray** (input_control) ..... number(-array) ⇝ *real* / integer / string
Obere Schwelle für die Grauwerte oder *'max'*.
Defaultwert: 255.0
Wertevorschläge: MaxGray ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0, 'max'}
Restriktion: MaxGray >= MinGray

Beispiel

```
read_image(Image,'fabrik')
sobel_dir(Image,EdgeAmp,EdgeDir,'sum_abs',3)
threshold(EdgeAmp,Seg,50,255)
skeleton(Seg,Rand)
connection(Rand,Lines)
select_shape(Lines,Edges,'area','and',10,1000000)
```

Komplexität

Sei A die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität $O(A)$.

Ergebnis

`threshold` liefert den Wert TRUE, falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`histo_to_thresh`, `min_max_gray`, `sobel_amp`, `binomial_filter`, `gauss_filter`, `reduce_domain`, `fill_interlace`

Nachfolger

`connection`, `dilation1`, `erosion1`, `opening`, `closing`, `rank_region`, `shape_trans`, `skeleton`

Alternativen

`class_2dim_sup`, `hysteresis_threshold`, `dyn_threshold`, `binary_threshold`, `char_threshold`, `auto_threshold`, `dual_threshold`

Siehe auch

`zero_crossing`, `background_seg`, `regiongrowing`

Modul

Foundation

threshold_sub_pix (Image : Border : Threshold :)

Subpixel-genaues Extrahieren von Grauwert-Höhenlinien in einem Bild

`threshold_sub_pix` extrahiert die Grauwert-Höhenlinien der Höhe `Threshold` des Eingabebildes `Image` subpixel-genau. Die extrahierten Höhenlinien werden als XLD-Konturen in `Border` zurückgegeben. Im Gegensatz zum Operator `threshold` liefert `threshold_sub_pix` keine Fläche zurück, sondern die Linien, die Bereiche mit Grauwert kleiner `Threshold` von Bereichen mit Grauwert größer `Threshold` trennen.

Bei der Extraktion wird das Eingabebild als Oberfläche interpretiert, wobei zwischen den Pixelmittelpunkten bilinear interpoliert wird. Konsistent mit der so entstehenden Oberfläche werden einzelne Grauwert-Höhenlinien für jedes Pixel extrahiert und zu topologisch „sauberen“ Konturen verknüpft. Das bedeutet, dass die Höhenlinienkonturen an Kreuzungspunkten sauber aufgetrennt werden. Falls im Bild flächenhafte Bereiche mit einem konstanten Grauwert von `Threshold` auftreten, wird nur der Rand solcher Gebiete als Höhenlinie zurückgeliefert.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
  Eingabebild.
- ▷ **`Border`** (output_object) ..... xld_cont-array ⇝ *object*
  Extrahierte Grauwert-Höhenlinien.
- ▷ **`Threshold`** (input_control) ..... number ⇝ *real* / integer
  Schwellenwert für die Grauwert-Höhenlinien.
  Defaultwert: 128
  Wertevorschläge: `Threshold` ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}

Beispiel

```
read_image(Image,'fabrik')
threshold_sub_pix(Image,Border,35)
dev_display(Border)
```

Ergebnis

`threshold_sub_pix` liefert normalerweise den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`threshold`

Siehe auch

`zero_crossing_sub_pix`

Modul

2D Metrology

`var_threshold` (`Image` : `Region` : `MaskWidth`, `MaskHeight`, `StdDevScale`, `AbsThreshold`, `LightDark` :)

Segmentieren mit Schwellen durch lokale Analyse von Mittelwert und Standardabweichung.

Mit `var_threshold` ist es möglich, Pixel des Eingabebildes `Image` zu selektieren, die

- eine hohe lokale Standardabweichung (für positive `StdDevScale`), oder eine niedrige lokale Standardabweichung (für negative `StdDevScale`) aufweisen

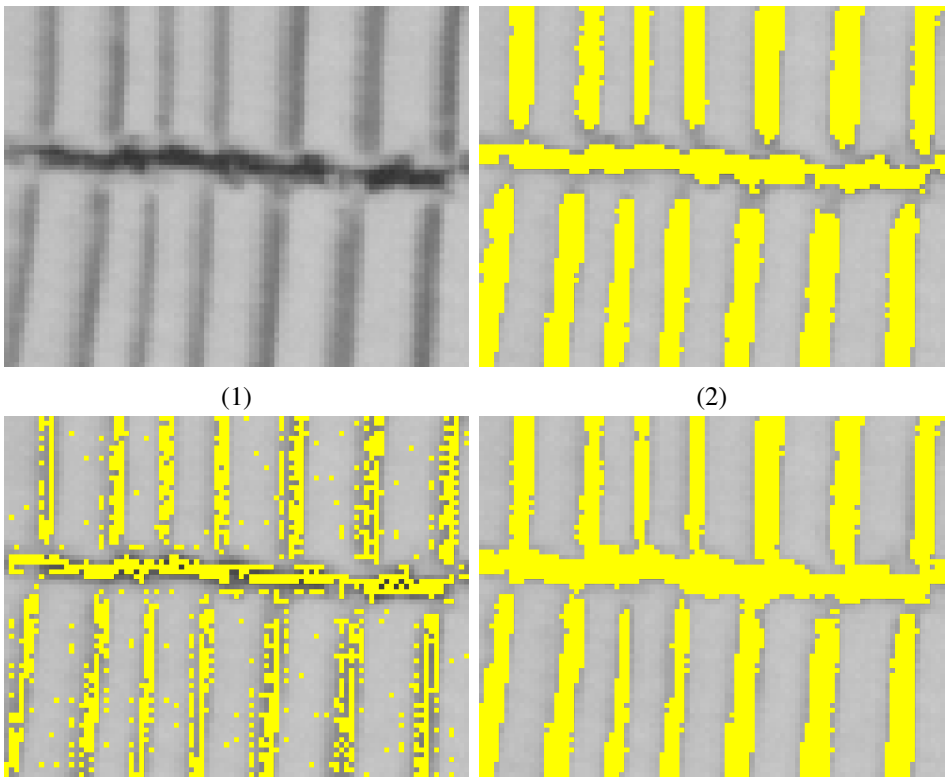
und

- lokal hell oder dunkel sind, gemäß `LightDark`.

Damit ist es möglich, Regionen in einem unregelmäßigen, verrauschten oder ungleichmäßig ausgeleuchteten Hintergrund zu finden.

Hinweise zu den Eingabeparametern

MaskWidth, MaskHeight Die Größe der Filtermaske, die durch `MaskWidth` und `MaskHeight` festgelegt wird, bestimmt die maximale Größe der Objekte, die segmentiert werden. Wenn die Maske allerdings zu groß ist, können Objekte, die dicht nebeneinander sind, miteinander verbunden werden.

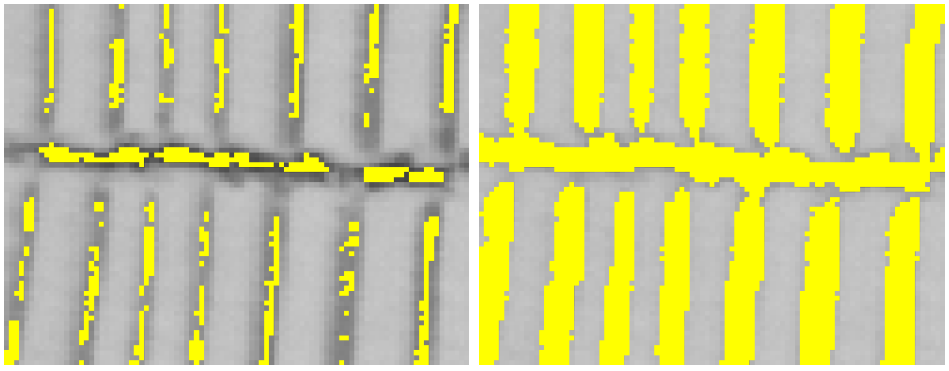


(1) Ursprüngliches Bild; das Ziel ist es, die vertikalen Linien zu zählen. (2) `MaskWidth` := *12*, `MaskHeight` := *12*, `StdDevScale` := *0.1*, alle vertikalen Linien werden korrekt segmentiert. (3) Falls die Maskengröße zu gering gewählt wird (*3*), werden die gewünschten Regionen nicht korrekt ausgewählt. (4) Falls die Maske zu groß ist (*40*), werden Objekte, die dicht nebeneinander sind, miteinander verbunden.

Wenn `MaskWidth` oder `MaskHeight` gerade Werte enthalten, werden die nächstgrößeren ungeraden Werte genutzt. Insgesamt kann *3* als geringster sinnvoller Wert angesehen werden.

StdDevScale Die lokale Standardabweichung wird als Maß für das Rauschen im Bild genutzt. Sie kann durch `StdDevScale` auf die gewünschte Empfindlichkeit skaliert werden. Ein höherer Wert bedeutet, dass Pixel selektiert werden, die sich stark von ihrer Umgebung unterscheiden.

Ein sinnvoller Wertebereich für den Parameter `StdDevScale` liegt zwischen -1.0 und 1.0. Vorgeschlagen wird der Wert 0.2. Wenn der Wert zu groß oder zu klein gewählt wird, kann eine komplett volle oder leere Region zurückgegeben werden.

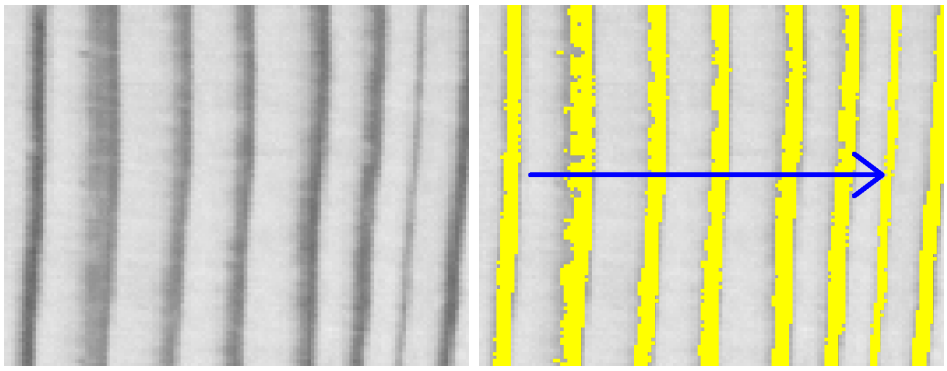


(1) Falls `StdDevScale` zu groß gewählt wird (*1.3*), ist der Operator zu 'wählerisch'; nur Pixel, die ihrer Umgebung sehr ähnlich sind, werden ausgewählt. (2) Falls `StdDevScale` zu gering ist (*-0.3*), werden zu viele Pixel ausgewählt, die ihrer Umgebung nur bedingt ähnlich sind.

AbsThreshold In homogenen Bereichen eines Bildes ist die Standardabweichung gering; der Einfluss von einzelnen Grauwerten ist daher sehr hoch. Um die Empfindlichkeit des Operators in homogenen Bereichen zu reduzieren, kann `AbsThreshold` angepasst werden. Damit können kleine Grauwertänderungen in homogenen Umgebungen ignoriert werden. Beachten Sie, falls Sie `StdDevScale` negativ wählen, sollte `AbsThreshold` entsprechend auch negativ sein.

LightDark Wenn *'light'* oder *'dark'* ausgewählt sind, werden jeweils alle Pixel selektiert, die heller bzw. dunkler als ihre Umgebung sind. *'equal'* selektiert alle Pixel, die von keiner der beiden Einstellungen ausgewählt werden; also alle Pixel, die ihrer Umgebung recht ähnlich sind. *'not_equal'* vereint alle Pixel von *'light'* und *'dark'*; es werden also alle Pixel selektiert, die sich von ihrer Umgebung unterscheiden.

Die Berechnung



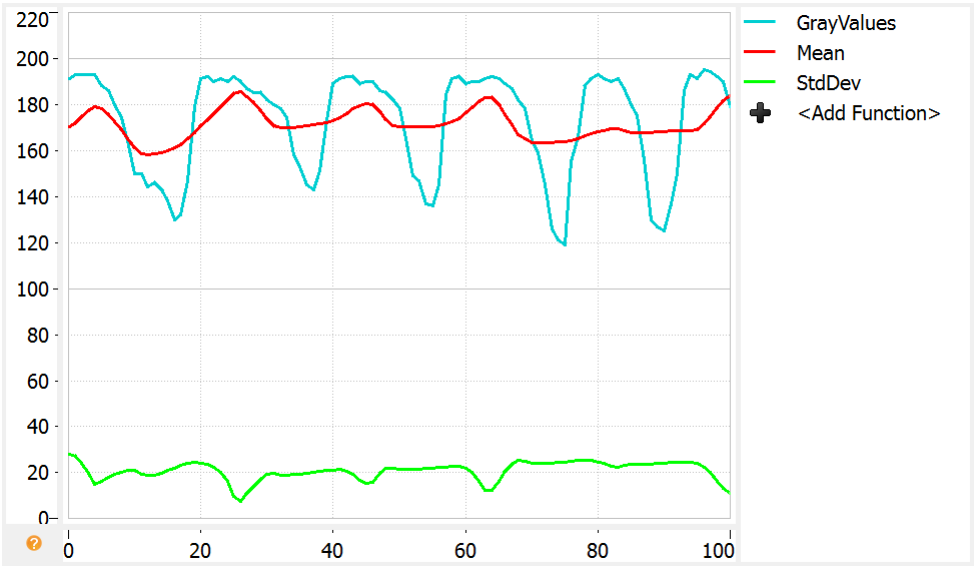
(1) (2)

(1) Ursprüngliches Bild. (2) Segmentiertes Bild (*0.6*, `MaskWidth` := *15*, `MaskHeight` := *15*, `AbsThreshold` := *10*). Die folgenden Bilder visualisieren beispielhaft, wie das Resultat entlang des blauen Pfeils entstanden ist.

`var_threshold` selektiert die Regionen des Eingabebildes `Image`, in denen die Bildpunkte eine Schwellenwertbedingung erfüllen. Der Schwellenwert wird aus dem Mittelwert und der Standardabweichung in einer lokalen Maske der Größe `MaskWidth` x `MaskHeight` um jeden Bildpunkt (x, y) berechnet.

Sei

- $g(x, y)$ der Grauwert des Eingabebildes `Image` an der Position (x, y),
- $m(x, y)$ der korrespondierende Mittelwert, und
- $d(x, y)$ die Standardabweichung der Grauwerte in der Maske um den Bildpunkt.



Der ursprüngliche Grauwert (GrayValue), der korrespondierende Mittelwert (Mean), und die Standardabweichung (StdDev) der Grauwerte in der Maske um diese Bildpunkte.

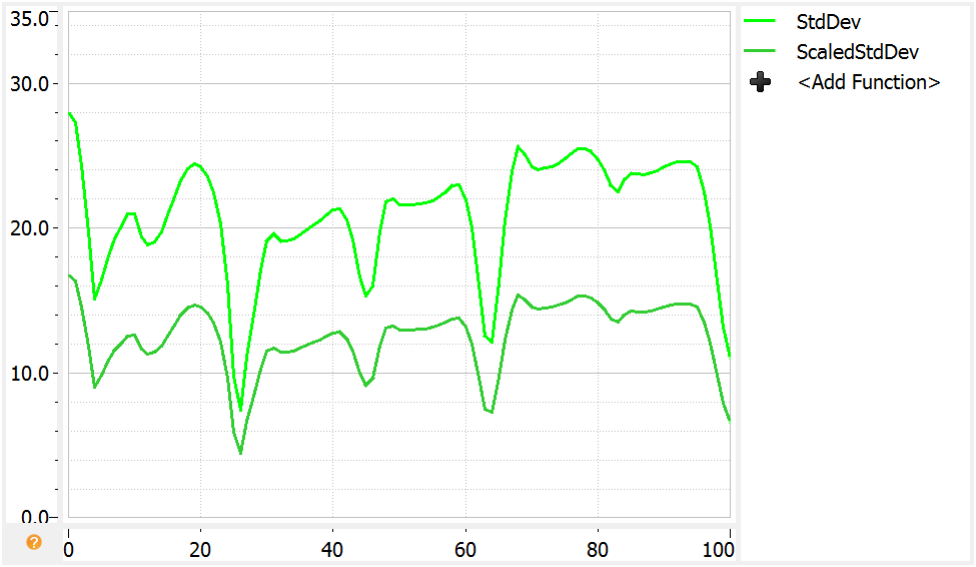
Der variable Schwellenwert $v(x, y)$ ist dann definiert als

$v(x, y) = max(\texttt{StdDevScale} * d(x, y), \texttt{AbsThreshold})$ für $\texttt{StdDevScale} \geq 0$

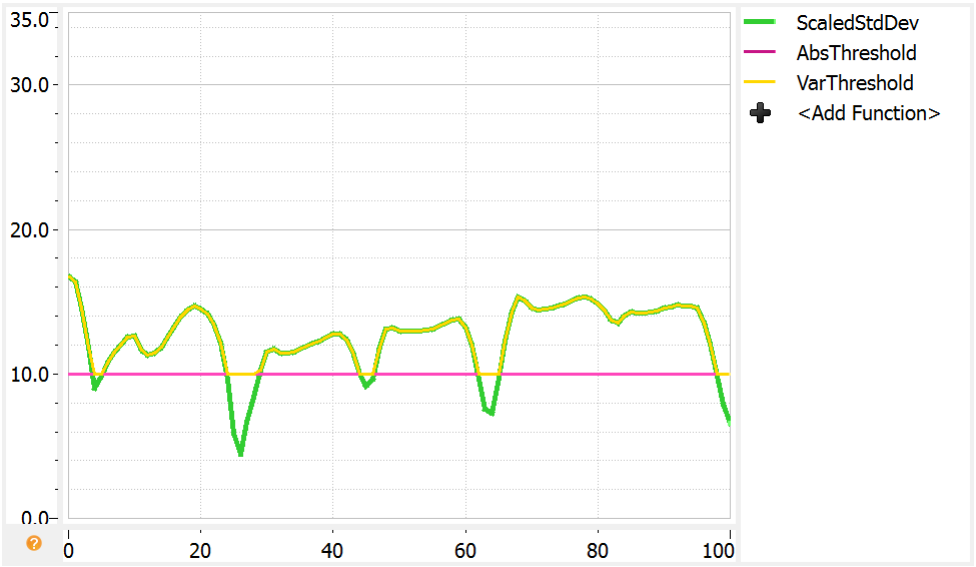
oder

$v(x, y) = min(\texttt{StdDevScale} * d(x, y), \texttt{AbsThreshold})$ für $\texttt{StdDevScale} < 0$.

Interpretation: Für eine positive `StdDevScale` wird jeder Pixel analysiert. Es wird überprüft, ob der benutzerdefinierte Wert von `AbsThreshold` oder die skalierte Standardabweichung größer ist. Der größere Wert wird als variabler Schwellenwert $v(x, y)$ genutzt. Für eine negative `StdDevScale` wird der jeweils kleinere Wert genutzt.



Die Standardabweichung kann mit `StdDevScale` skaliert werden.



Der variable Schwellenwert wird aus der skalierten Standardabweichung und `AbsThreshold` berechnet.

Welche Pixel basierend auf dem variablen Schwellenwert ausgewählt werden, bestimmt der Parameter `LightDark`:

`LightDark` = *'light'*:

$g(x, y) \geq m(x, y) + v(x, y)$.

Interpretation: Falls ein Pixel um $v(x, y)$ heller ist als seine Umgebung, wird er selektiert.

`LightDark` = *'dark'*:

$g(x, y) \leq m(x, y) - v(x, y)$.

Interpretation: Falls ein Pixel um $v(x,y)$ dunkler ist als seine Umgebung, wird er selektiert.

`LightDark` = *'equal'*:

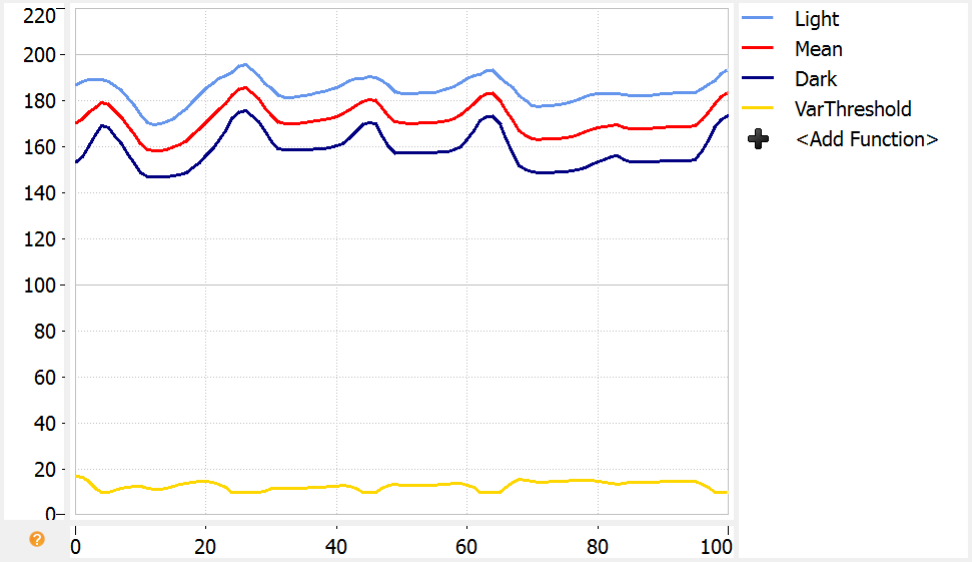
$m(x,y) - v(x,y) \leq g(x,y) \leq m(x,y) + v(x,y).$

Interpretation: Es werden genau die Pixel ausgewählt, die weder von *'light'* noch von *'dark'* ausgewählt werden, also die Pixel, die ihrer Umgebung recht ähnlich sind.

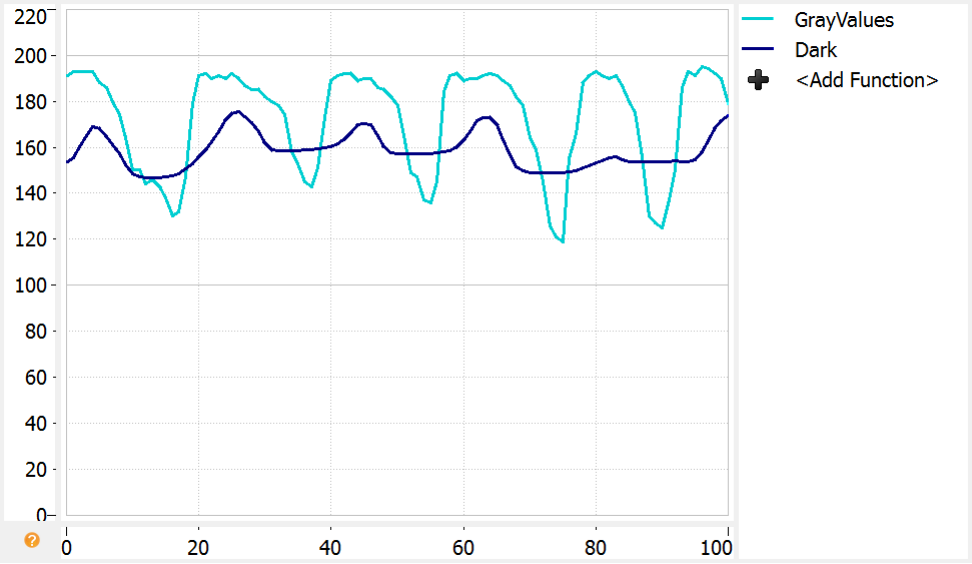
`LightDark` = *'not_equal'*:

$m(x,y) - v(x,y) > g(x,y) \quad \vee \quad g(x,y) > m(x,y) + v(x,y).$

Interpretation: Es werden alle Pixel von *'light'* und *'dark'* ausgewählt, also alle Pixel, die sich um $v(x,y)$ von ihrer Umgebung unterscheiden.



'light' und *'dark'* basieren auf dem korrespondierenden Mittelwert und dem variablen Schwellenwert.



Zum Beispiel werden, wenn *'dark'* ausgewählt ist, alle Pixel selektiert, bei denen die 'Dark'-Kurve über der 'GrayValues'-Kurve ist.

Achtung

Dieser Operator liefert eventuell unerwartete Resultate zurück, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage(-array) ⇝ *object :* byte / int2 / int4 / uint2 / real
  Eingabebild.
- ▷ **Region** (output_object) . . . . . region(-array) ⇝ *object*
  Segmentierte Regionen.
- ▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
  Breite des Maskenfensters für Mittelwert- und Standardabweichungsberechnung.
  Defaultwert: 15
  Wertevorschläge: `MaskWidth` ∈ {9, 11, 13, 15}
  Restriktion: MaskWidth >= 1
- ▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
  Höhe des Maskenfensters für Mittelwert- und Standardabweichungsberechnung.
  Defaultwert: 15
  Wertevorschläge: `MaskHeight` ∈ {9, 11, 13, 15}
  Restriktion: MaskHeight >= 1
- ▷ **StdDevScale** (input_control) . . . . . number ⇝ *real* / integer
  Faktor für die Standardabweichung der Grauwerte.
  Defaultwert: 0.2
  Wertevorschläge: `StdDevScale` ∈ {-0.2, -0.1, 0.1, 0.2}
- ▷ **AbsThreshold** (input_control) . . . . . number ⇝ *real* / integer
  Minimale Grauwertdifferenz zum Mittelwert.
  Defaultwert: 2
  Wertevorschläge: `AbsThreshold` ∈ {-2, -1, 0, 1, 2}
- ▷ **LightDark** (input_control) . . . . . string ⇝ *string*
  Schwellenwerttyp.
  Defaultwert: 'dark'
  Werteliste: `LightDark` ∈ {'dark', 'light', 'equal', 'not_equal'}

Komplexität

Sei A die Größe der Eingaberegion, dann beträgt die Laufzeit $O(A)$.

Ergebnis

`var_threshold` gibt TRUE zurück, wenn alle Parameter korrekt angegeben sind. Das Verhalten bezüglich der Eingabebilder und Ausgaberegionen kann mit den Werten von *'no_object_result'*, *'empty_region_result'*, und *'store_empty_region'* im Operator `set_system` kontrolliert werden. Wenn nötig wird eine Ausnahmebehandlung vorgenommen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Alternativen

`dyn_threshold`, `threshold`

Literatur

W.Niblack, "An Introduction to Digital Image Processing", Page 115-116, Englewood Cliffs, N.J., Prentice Hall, 1986

Modul

Foundation

`zero_crossing ( Image : RegionCrossing : : )`

Extrahieren von Nulldurchgängen in einem Bild.

zero_crossing liefert als Ausgaberegion die Nulldurchgänge im übergebenen Eingabebild zurück. Ein Pixel wird dabei als Nulldurchgang markiert (d.h. in die Ausgaberegion RegionCrossing aufgenommen), wenn sein Grauwert (in Image) gleich Null ist oder mindestens einer seiner Nachbarn in 4er Nachbarschaft ein anderes Vorzeichen aufweist.

Diese Routine wird sinnvollerweise nach einer Kantenfilterung, die die zweite Ableitung bestimmt (z.B. laplace_of_gauss), gegebenenfalls gefolgt von einer Glättungsoperation aufgerufen. In diesem Fall markieren die Nulldurchgänge Kanten(kandidaten).

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* int1 / int2 / int4 / real
  Eingabebild.
- ▷ **RegionCrossing** (output_object) ..... region(-array) ⇝ *object*
  Nulldurchgänge.

Ergebnis

zero_crossing liefert normalerweise den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.
- Automatisch parallelisiert auf Domainebene.

Vorgänger

laplace, laplace_of_gauss, derivate_gauss

Nachfolger

connection, skeleton, boundary, select_shape, fill_up

Alternativen

threshold, dual_threshold, zero_crossing_sub_pix

Modul

Foundation

zero_crossing_sub_pix (Image : ZeroCrossings : :)

Subpixel-genaues Extrahieren von Nulldurchgängen in einem Bild

zero_crossing_sub_pix extrahiert die Nulldurchgänge des Eingabebildes Image subpixel-genau. Die extrahierten Nulldurchgänge werden als XLD-Konturen in ZeroCrossings zurückgegeben. zero_crossing_sub_pix kann somit zur subpixel-genauen Kantenextraktion verwendet werden, wenn als Eingabebild ein Laplace-gefiltertes Bild übergeben wird (siehe laplace, laplace_of_gauss, derivate_gauss).

Bei der Extraktion wird das Eingabebild als Oberfläche interpretiert, wobei zwischen den Pixelmittelpunkten bilinear interpoliert wird. Konsistent mit der so entstehenden Oberfläche werden einzelne Nulldurchgangslinien für jedes Pixel extrahiert und zu topologisch „sauberen“ Konturen verknüpft. Das bedeutet, dass die Nulldurchgangskonturen an Kreuzungspunkten sauber aufgetrennt werden. Falls im Bild flächenhafte Bereiche mit einem konstanten Grauwert von 0 auftreten, wird nur der Rand solcher Gebiete als Nulldurchgang zurückgeliefert.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* int1 / int2 / int4 / real
  Eingabebild.
- ▷ **ZeroCrossings** (output_object) ..... xld_cont-array ⇝ *object*
  Extrahierte Nulldurchgänge.

Beispiel

```
* Detection zero crossings of the Laplacian-of-Gaussian
* of an aerial image
read_image(Image,'mreut')
derivate_gauss(Image,Laplace,3,'laplace')
zero_crossing_sub_pix(Laplace,ZeroCrossings)
dev_display(ZeroCrossings)

* Detection of edges, i.e, zero crossings of the Laplacian-of-Gaussian
* that have a large gradient magnitude, in an aerial image
read_image(Image,'mreut')
Sigma := 1.5
* Compensate the threshold for the fact that derivate_gauss(...,'gradient')
* calculates a Gaussian-smoothed gradient, in which the edge amplitudes
* are too small because of the Gaussian smoothing, to correspond to a true
* edge amplitude of 20.
Threshold := 20/(Sigma*sqrt(2*3.1415926))
derivate_gauss(Image,Gradient,Sigma,'gradient')
threshold(Gradient,Region,Threshold,255)
reduce_domain(Image,Region,ImageReduced)
derivate_gauss(ImageReduced,Laplace,Sigma,'laplace')
zero_crossing_sub_pix(Laplace,Edges)
dev_display(Edges)
```

Ergebnis

`zero_crossing_sub_pix` liefert normalerweise den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`laplace`, `laplace_of_gauss`, `diff_of_gauss`, `derivate_gauss`

Alternativen

`zero_crossing`

Siehe auch

`threshold_sub_pix`

Modul

2D Metrology

23.6 Topographie

critical_points_sub_pix (Image : : Filter, Sigma, Threshold : RowMin, ColumnMin, RowMax, ColumnMax, RowSaddle, ColumnSaddle)

Subpixelgenaue Detektion von kritischen Punkten in einem Bild.

`critical_points_sub_pix` extrahiert subpixelgenau kritische Punkte, also lokale Maxima, lokale Minima und Sattelpunkte, aus dem Bild `Image`. Zur Detektion der kritischen Punkte wird das Eingangsbild in jedem Punkt durch ein quadratisches Polynom in x und y approximiert und dieses auf Extremstellen und Sattelpunkte hin untersucht. Um das Polynom aufzustellen, werden die partiellen Ableitungen abhängig von `Filter` entweder

mit verschiedenen Ableitungen der Gaußfunktion oder nach dem Facet-Modell berechnet. Im ersten Fall bestimmt `Sigma` die Größe der verwendeten Gaußkerne, im zweiten Fall erfolgt vor der Berechnung der partiellen Ableitungen eine Glättung des Eingabebildes mit einem Gaußfilter der Größe `Sigma`. Der Modus *'facet'* ist damit etwas schneller, liefert aber auch etwas ungenauere Resultate. Sind beide Eigenwerte der Hessematrix im untersuchten Punkt ihrem Betrage nach größer als `Threshold`, dann liegt ein Extrem- oder Sattelpunkt vor. Die Eigenwerte entsprechen der Krümmung der Grauwertoberfläche. Sind beide Eigenwerte negativ, so liegt ein lokales Maximum vor, sind beide positiv, ein lokales Minimum; bei unterschiedlichen Vorzeichen liegt ein Sattelpunkt vor.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
  Eingabebild.
- ▷ **`Filter`** (input_control) ..... string ⇝ *string*
  Verfahren zur Berechnung der partiellen Ableitungen.
  Defaultwert: 'facet'
  Werteliste: `Filter` ∈ {'facet', 'gauss'}
- ▷ **`Sigma`** (input_control) ..... real ⇝ *real*
  Sigma der Gaußfunktion. Ist `Filter` auf 'facet' gesetzt, kann `Sigma` auf 0.0 gesetzt werden, um eine Glättung des Eingangsbildes zu vermeiden.
  Wertevorschläge: `Sigma` ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
  Restriktion: Sigma >= 0.0
- ▷ **`Threshold`** (input_control) ..... real ⇝ *real*
  Mindestbetrag der Eigenwerte der Hessematrix.
  Defaultwert: 5.0
  Wertevorschläge: `Threshold` ∈ {2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0}
  Restriktion: Threshold >= 0.0
- ▷ **`RowMin`** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der gefundenen Minima.
- ▷ **`ColumnMin`** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der gefundenen Minima.
- ▷ **`RowMax`** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der gefundenen Maxima.
- ▷ **`ColumnMax`** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der gefundenen Maxima.
- ▷ **`RowSaddle`** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der gefundenen Sattelpunkte.
- ▷ **`ColumnSaddle`** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der gefundenen Sattelpunkte.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `critical_points_sub_pix` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`gen_cross_contour_xld`, `disp_cross`

Alternativen

`local_min_sub_pix`, `local_max_sub_pix`, `saddle_points_sub_pix`

Siehe auch

local_min, local_max, plateaus, plateaus_center, lowlands, lowlands_center

Modul

Foundation

local_max (Image : LocalMaxima : :)

Detektieren aller Punkte, deren Grauwerte die lokale Maxima sind.

local_max wählt all die Punkte aus Image aus, deren Grauwert größer ist als die Grauwerte aller benachbarten Pixel, und gibt sie in LocalMaxima zurück. Die zu berücksichtigende Umgebung kann mit set_system(::'neighborhood',<4/8>) eingestellt werden.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Eingabebild.

▷ **LocalMaxima** (output_object) ..... region(-array) ⇝ *object*
Gefundene lokale Maxima als Region.
Parameteranzahl: LocalMaxima == Image

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  using namespace Halcon;
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);

  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray maxi = cres.LocalMax ();

  win.SetColored (12);
  maxi.Display (win);
  win.Click ();

  return (0);
}
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

binomial_filter, gauss_filter, smooth_image

Nachfolger

get_region_points, connection

Alternativen

nonmax_suppression_amp, plateaus, plateaus_center

Siehe auch

monotony, topographic_sketch, corner_response, texture_laws

Modul

Foundation

local_max_sub_pix (Image : : Filter, Sigma, Threshold : Row, Column)

Subpixelgenaue Detektion von lokalen Maxima in einem Bild.

local_max_sub_pix extrahiert subpixelgenau lokale Maxima aus einem Bild Image. Zur Detektion der lokalen Maxima wird das Eingangsbild in jedem Punkt durch ein quadratisches Polynom in x und y approximiert und dieses auf lokale Maxima hin untersucht. Um das Polynom aufzustellen, werden die partiellen Ableitungen abhängig von Filter entweder mit verschiedenen Ableitungen der Gaußfunktion oder nach dem Facet-Modell berechnet. Im ersten Fall bestimmt Sigma die Größe der verwendeten Gaußkerne, im zweiten Fall erfolgt vor der Berechnung der partiellen Ableitungen eine Glättung des Eingabebildes mit einem Gaußfilter der Größe Sigma. Der Modus *'facet'* ist damit etwas schneller, liefert aber auch etwas ungenauere Resultate. Sind beide Eigenwerte der Hessematrix im untersuchten Punkt kleiner als -Threshold, dann liegt ein lokales Maximum vor. Die Eigenwerte entsprechen der Krümmung der Grauwertoberfläche.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
Eingabebilder.

▷ **Filter** (input_control) ..... string ⇝ *string*
Verfahren zur Berechnung der partiellen Ableitungen.
Defaultwert: 'facet'
Werteliste: Filter ∈ {'facet', 'gauss'}

▷ **Sigma** (input_control) ..... real ⇝ *real*
Sigma der Gaußfunktion. Ist Filter auf 'facet' gesetzt, kann Sigma auf 0.0 gesetzt werden, um eine Glättung des Eingangsbildes zu vermeiden.
Wertevorschläge: Sigma ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
Restriktion: Sigma >= 0.0

▷ **Threshold** (input_control) ..... real ⇝ *real*
Mindestbetrag der Eigenwerte der Hessematrix.
Defaultwert: 5.0
Wertevorschläge: Threshold ∈ {2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0}
Restriktion: Threshold >= 0.0

▷ **Row** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinaten der gefundenen Maxima.

▷ **Column** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinaten der gefundenen Maxima.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert

local_max_sub_pix den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

gen_cross_contour_xld, disp_cross

Alternativen

critical_points_sub_pix, local_min_sub_pix, saddle_points_sub_pix

Siehe auch

local_max, plateaus, plateaus_center

Modul

Foundation

local_min (Image : LocalMinima : :)

Detektieren aller Punkte die lokale Minima sind.

local_min wählt alle Punkte aus Image aus, deren Grauwert kleiner ist als die Grauwerte aller benachbarten Pixel, und gibt sie in LocalMinima zurück. Die zu berücksichtigende Umgebung kann mit set_system('neighborhood',<4/8>) eingestellt werden.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Zu verarbeitendes Bild.

▷ **LocalMinima** (output_object) ..... region(-array) ⇝ *object*
Gefundene lokale Minima als Regionen.
Parameteranzahl: LocalMinima == Image

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);
```

```
  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray mins = cres.LocalMin ();

  win.SetColored (12);
  mins.Display (win);
  win.Click ();

  return (0);
}
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

binomial_filter, gauss_filter, smooth_image

Nachfolger

get_region_points, connection

Alternativen

gray_skeleton, lowlands, lowlands_center

Siehe auch

monotony, topographic_sketch, corner_response, texture_laws

Modul

Foundation

local_min_sub_pix (Image : : Filter, Sigma, Threshold : Row, Column)

Subpixelgenaue Detektion von lokalen Minima in einem Bild.

local_min_sub_pix extrahiert subpixelgenau lokale Minima aus dem Bild Image. Zur Detektion der lokalen Minima wird das Eingangsbild in jedem Punkt durch ein quadratisches Polynom in x und y approximiert und dieses auf lokale Minima hin untersucht. Um das Polynom aufzustellen, werden die partiellen Ableitungen abhängig von Filter entweder mit verschiedenen Ableitungen der Gaußfunktion oder nach dem Facet-Modell berechnet. Im ersten Fall bestimmt Sigma die Größe der verwendeten Gaußkerne, im zweiten Fall erfolgt vor der Berechnung der partiellen Ableitungen eine Glättung des Eingabebildes mit einem Gaußfilter der Größe Sigma. Der Modus *'facet'* ist damit etwas schneller, liefert aber auch etwas ungenauere Resultate. Sind beide Eigenwerte der Hessematrix im untersuchten Punkt größer als Threshold, dann liegt ein lokales Minimum vor. Die Eigenwerte entsprechen der Krümmung der Grauwertoberfläche.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
Eingabebilder.

▷ **Filter** (input_control) ..... string ⇝ *string*
Verfahren zur Berechnung der partiellen Ableitungen.
Defaultwert: 'facet'
Werteliste: Filter ∈ {'facet', 'gauss'}

▷ **Sigma** (input_control) ..... real ⇝ *real*
Sigma der Gaußfunktion. Ist Filter auf 'facet' gesetzt, kann Sigma auf 0.0 gesetzt werden, um eine Glättung des Eingangsbildes zu vermeiden.
Wertevorschläge: Sigma ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
Restriktion: Sigma >= 0.0

▷ **Threshold** (input_control) ..... real ⇝ *real*
Mindestbetrag der Eigenwerte der Hessematrix.
Defaultwert: 5.0
Wertevorschläge: `Threshold` ∈ {2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0}
Restriktion: Threshold >= 0.0

▷ **Row** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinaten der gefundenen Minima.

▷ **Column** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinaten der gefundenen Minima.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `local_min_sub_pix` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

`gen_cross_contour_xld`, `disp_cross`

Alternativen

`critical_points_sub_pix`, `local_max_sub_pix`, `saddle_points_sub_pix`

Siehe auch

`local_min`, `lowlands`, `lowlands_center`

Modul

Foundation

lowlands (Image : Lowlands : :)

Detektieren aller Grauwert-Tiefebenen.

`lowlands` wählt alle Punkte aus `Image` aus, deren Grauwert nicht größer als der Grauwert ihrer Nachbarn (8er-Nachbarschaft) ist, und gibt sie in `Lowlands` zurück. Jede Tiefebene ergibt eine eigene Region.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Zu verarbeitendes Bild.

▷ **Lowlands** (output_object) ..... region-array ⇝ *object*
Gefundenen Tiefebenen als Regionen (für jede Tiefebene eine Region).

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  if (argc != 2)
```

```
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);

  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray mins = cres.Lowlands ();

  win.SetColored (12);
  mins.Display (win);
  win.Click ();

  return (0);
}
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

binomial_filter, gauss_filter, smooth_image

Nachfolger

area_center, get_region_points, select_shape

Alternativen

lowlands_center, gray_skeleton, local_min

Siehe auch

monotony, topographic_sketch, corner_response, texture_laws

Modul

Foundation

lowlands_center (Image : Lowlands : :)

Detektieren der Schwerpunkte aller Grauwert-Tiefebenen.

lowlands_center wählt alle Punkte aus Image aus, deren Grauwert nicht größer als der Grauwert ihrer Nachbarn (8-er Nachbarschaft) ist, und gibt sie in Lowlands zurück. Wenn mehrere solcher Punkte zusammenhängen (Tiefebene), wird der Schwerpunkt verwendet. Jede Tiefebene ergibt eine eigene Region.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Zu verarbeitendes Bild.

▷ **Lowlands** (output_object) ..... region-array ⇝ *object*
Schwerpunkte der gefundenen Tiefebenen als Regionen (für jede Tiefebene eine Region).

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
```

```
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);

  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray mins = cres.LowlandsCenter ();

  win.SetColored (12);
  mins.Display (win);
  win.Click ();

  return (0);
}
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

binomial_filter, gauss_filter, smooth_image

Nachfolger

area_center, get_region_points, select_shape

Alternativen

lowlands, gray_skeleton, local_min

Siehe auch

monotony, topographic_sketch, corner_response, texture_laws

Modul

Foundation

plateaus (Image : Plateaus : :)

Detektieren aller Grauwert-Plateaus.

plateaus wählt alle Punkte aus Image aus, deren Grauwert nicht kleiner als der Grauwert ihrer Nachbarn (8er Nachbarschaft) ist, und gibt sie in Plateaus zurück. Jedes Maximum ergibt eine eigene Region.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Eingabebild.

▷ **Plateaus** (output_object) ..... region-array ⇝ *object*
Gefundenen Plateaus als Regionen (für jedes Plateau eine Region).

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);

  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray maxi = cres.Plateaus ();

  win.SetColored (12);
  maxi.Display (win);
  win.Click ();

  return (0);
}
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

binomial_filter, gauss_filter, smooth_image

Nachfolger

area_center, get_region_points, select_shape

Alternativen

plateaus_center, nonmax_suppression_amp, local_max

Siehe auch

monotony, topographic_sketch, corner_response, texture_laws

Modul

Foundation

plateaus_center (Image : Plateaus : :)

Detektieren der Schwerpunkte aller Grauwert-Plateaus.

plateaus_center wählt alle Punkte aus Image aus, deren Grauwert nicht kleiner als der Grauwert ihrer Nachbarn (8-er Nachbarschaft) ist, und gibt sie in Plateaus zurück. Wenn mehrere solcher Punkte zusammenhängen (Plateau), wird der Schwerpunkt verwendet. Jeder Schwerpunkt eines Plateaus ergibt eine eigene Region.

Parameter

▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real
Eingabebild.

▷ **Plateaus** (output_object) ..... region-array ⇝ *object*
Schwerpunkte der gefundenen Plateaus als Regionen (für jedes Plateau eine Region).

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);

  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray maxi = cres.PlateausCenter ();

  win.SetColored (12);
  maxi.Display (win);
  win.Click ();

  return (0);
}
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

binomial_filter, gauss_filter, smooth_image

Nachfolger

area_center, get_region_points, select_shape

Alternativen

plateaus, nonmax_suppression_amp, local_max

Siehe auch

monotony, topographic_sketch, corner_response, texture_laws

Modul

Foundation

pouring (Image : Regions : Mode, MinGray, MaxGray :)

Segmentieren nach dem „Gießkannen“-Prinzip.

pouring sieht die Werte des Eingabebildes als topologisches (Grauwert-)Gebirge an, das von oben betrachtet wird. Dabei werden große (= helle) Grauwerte als höher gelegene Punkte interpretiert, wogegen kleine (= dunkle) Grauwerte entsprechend die „Täler“ bilden. pouring segmentiert die Eingabebilder in mehreren Teilschritten. Zuerst werden die relativ höchsten Grauwerte — im Folgenden als „lokale Maxima“ bezeichnet — ermittelt, d.h. die Pixel, die entweder allein oder in Form eines zusammenhängenden Plateaus größere Grauwerte besitzen, als die unmittelbaren Nachbarn. Die Bestimmung der lokalen Maxima basiert dabei auf Vergleichen in der 4-er Nachbarschaft. Im nächsten Teilschritt bilden die so gefundenen lokalen Maxima den Ausgangspunkt für die Bereichszuordnung, bei der sich die Segmente von den Maxima ausgehend bis in die „Talsohlen“ ausdehnen, d.h. solange absteigende Grauwertketten existieren (wie Wasser, das aus einer Gießkanne über die Kuppen gegossen wurde und nun in alle Richtungen nach unten abläuft). Auch hier wird die 4-er Nachbarschaft zugrunde gelegt, jedoch mit abgeschwächter Bedingung (kleiner gleich). Dabei kann es zur Überschneidungen in den „Talsohlen“ kommen, wenn die Bedingung für mehr als ein Segment erfüllt ist. Diese Schnittmengen werden vorerst keinem Segment zugeordnet, sondern im letzten Teilschritt unter den konkurrierenden Segmenten aufgeteilt. Die Aufteilung erfolgt durch pixelweises gleichmäßiges Ausdehnen aller beteiligten Segmente, bis alle Pixel eindeutig zugeordnet wurden. Mit Hilfe des Parameters Mode kann der Programmablauf gesteuert werden. In Abhängigkeit von diesem Eingabeparameter läuft entweder der gesamte Algorithmus ab, oder wird nach Vollendung von Teilschritten — z.B. dem Auffinden aller lokalen Maxima oder der isolierten Segmente ohne Aufteilung der Schnittmengen — beendet. Die möglichen Werte für Mode werden nachfolgend erläutert.

’all’ Diese Parameterbelegung stellt den normalen Modus dar. Es wird die komplette Segmentierungsroutine gestartet, bei der alle lokalen Maxima mit den zuzuordnenden Bereichen ermittelt werden. Überschneidungsbereiche werden aufgeteilt.

’maxima’ Die Segmentierungsroutine terminiert, sobald die lokalen Maxima der Eingabebilder ermittelt worden sind. Eine Segmentierung in zugehörige Bereiche wird dann nicht mehr durchgeführt. Die Ausgabe besteht aus den verschiedenen Maxima.

’regions’ Bei dieser Parameterbelegung führt die Segmentierungsroutine die Suche nach allen lokalen Maxima sowie den zugehörigen Bereichen, die eindeutig zugeordnet werden können, durch. Die Teilbereiche, in den Überschneidungen auftreten, d.h. die mehr als einem Maximum zugeordnet werden können, werden nicht weiter bearbeitet, also nicht aufgeteilt. Die Ausgabe besteht aus allen ermittelten isolierten Segmenten.

Um bei Ablauf der vollständigen Segmentierung die Aufteilung eines gleichmäßigen, vom restlichen Eingabebild verschiedenen Hintergrundes zu vermeiden, bieten die Parameter MinGray und MaxGray die Möglichkeit, Grauwertschwellen vorzugeben, so dass der Hintergrund ausmaskiert wird. Ein gewünschter unterer Grauwert wird in MinGray angegeben. Alle Bildpunkte, die einen kleineren Grauwert besitzen, werden dann weder zur Bestimmung der Maxima, noch zur weiteren Segmentierung herangezogen. Entsprechendes gilt für die Angabe einer oberen Grauwertschwelle in MaxGray: alle Pixel mit einem größeren Grauwert werden ebenfalls einfach ausmaskiert. Für eine komplette Segmentierung aller Bildpunkte des Eingabebildes setzt man MinGray auf 0 und MaxGray auf 255 und erreicht so eine Bearbeitung des kompletten Grauwertspektrums. Auf MinGray $<$ MaxGray ist zu achten.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object :* byte
  Eingabebild.
- ▷ **Regions** (output_object) . . . . . region-array ⇝ *object*
  Segmentierte Regionen.

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Gewünschter Modus für den Programmablauf.
Defaultwert: 'all'
Werteliste: `Mode` ∈ {'all', 'maxima', 'regions'}

▷ **MinGray** (input_control) . . . . . integer ⇝ *integer*
Alle Grauwerte, die echt unterhalb dieses Schwellenwertes liegen, werden ausmaskiert.
Defaultwert: 0
Wertevorschläge: `MinGray` ∈ {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110}
Typischer Wertebereich: 0 ≤ `MinGray` ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: MinGray >= 0

▷ **MaxGray** (input_control) . . . . . integer ⇝ *integer*
Alle Grauwerte, die echt oberhalb dieses Schwellenwertes liegen, werden ausmaskiert.
Defaultwert: 255
Wertevorschläge: `MaxGray` ∈ {100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 255}
Typischer Wertebereich: 0 ≤ `MaxGray` ≤ 255 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 10
Restriktion: MaxGray <= 255 && MaxGray > MinGray

Beispiel

```
* Segment a filtered image
read_image(Image,'particle')
mean_image(Image,Mean,11,11)
pouring(Mean,Seg,'all',0,255)
dev_display(Mean)
dev_set_colored(12)
dev_display(Seg)

* Segment an image while masking the dark background
read_image(Image,'particle')
mean_image(Image,ImageMean,15,15)
pouring(Mean,Seg,'all',90,255)
dev_display(Mean)
dev_set_colored(12)
dev_display(Seg)
```

Komplexität

Sei N die Anzahl der Bildpunkte des Eingabebildes und M die Anzahl der gefundenen Segmente, wobei das umschreibende Rechteck für ein Segment i insgesamt m_i Pixel beinhalte. Sei außerdem noch K_i die Komplexität der Lauflängenkodierung von Segment i, so beträgt die Laufzeitkomplexität

$$O(3 * N + sum_M(3 * m_i) + sum_M(K_i)) \ .$$

Ergebnis

`pouring` liefert normalerweise den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: mutually exclusive (läuft parallel zu anderen nicht-exklusiven Operatoren mit Ausnahme von sich selbst).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

binomial_filter, gauss_filter, smooth_image, mean_image

Alternativen

watersheds, local_max, watersheds_threshold, watersheds_marker

Siehe auch

histo_2dim, expand_region, expand_gray, expand_gray_ref

Modul

Foundation

saddle_points_sub_pix (Image : : Filter, Sigma, Threshold : Row, Column)

Subpixelgenaue Detektion von Sattelpunkten in einem Bild.

saddle_points_sub_pix extrahiert subpixelgenau Sattelpunkte aus einem Bild, also Punkte, in denen entlang einer Richtung die Bildintensität minimal ist, während sie gleichzeitig entlang einer anderen Richtung maximal ist. Zur Detektion der Sattelpunkte wird das Eingangsbild in jedem Punkt durch ein quadratisches Polynom in x und y approximiert und dieses auf Sattelpunkte hin untersucht. Um das Polynom aufzustellen, werden die partiellen Ableitungen abhängig von Filter entweder mit verschiedenen Ableitungen der Gaußfunktion oder nach dem Facet-Modell berechnet. Im ersten Fall bestimmt Sigma die Größe der verwendeten Gaußkerne, im zweiten Fall erfolgt vor der Berechnung der partiellen Ableitungen eine Glättung des Eingabebildes mit einem Gaußfilter der Größe Sigma. Der Modus *'facet'* ist damit etwas schneller, liefert aber auch etwas ungenauere Resultate. Sind beide Eigenwerte der Hessematrix im untersuchten Punkt ihrem Betrage nach größer als Threshold und weisen sie unterschiedliche Vorzeichen auf, dann liegt ein Sattelpunkt vor. Die Eigenwerte entsprechen der Krümmung der Grauwertoberfläche.

saddle_points_sub_pix eignet sich besonders, um Ecken in einem Bild aufzufinden, bei denen sich Felder unterschiedlicher Intensität in der Art der schwarzen und weißen Felder eines Schachbretts berühren. Durch den hohen Kontrast und die Formgebung wird das Auffinden und die genaue Positionsbestimmung solcher Ecken besonders begünstigt.

Achtung

Beachten Sie, dass Filteroperatoren eventuell unerwartete Resultate ausgeben, wenn ein Bild mit einer reduzierten Domäne als Input übergeben wird. Weitere Informationen können im Kapitel Filter gefunden werden.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / int1 / int2 / uint2 / int4 / real
  Eingabebilder.
- ▷ **Filter** (input_control) ..... string ⇝ *string*
  Verfahren zur Berechnung der partiellen Ableitungen.
  Defaultwert: 'facet'
  Werteliste: Filter ∈ {'facet', 'gauss'}
- ▷ **Sigma** (input_control) ..... real ⇝ *real*
  Sigma der Gaußfunktion. Ist Filter auf 'facet' gesetzt, kann Sigma auf 0.0 gesetzt werden, um eine Glättung des Eingangsbildes zu vermeiden.
  Wertevorschläge: Sigma ∈ {0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 3.0}
  Restriktion: Sigma >= 0.0
- ▷ **Threshold** (input_control) ..... real ⇝ *real*
  Mindestbetrag der Eigenwerte der Hessematrix.
  Defaultwert: 5.0
  Wertevorschläge: Threshold ∈ {2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0}
  Restriktion: Threshold >= 0.0
- ▷ **Row** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der gefundenen Sattelpunkte.
- ▷ **Column** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der gefundenen Sattelpunkte.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert

saddle_points_sub_pix den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Nachfolger

gen_cross_contour_xld, disp_cross

Alternativen

critical_points_sub_pix, local_min_sub_pix, local_max_sub_pix

Modul

Foundation

watersheds (Image : Basins, Watersheds : :)

Segmentieren nach dem Wasserscheiden-Prinzip.

watersheds zerlegt das Bild aufgrund der Topologie der Grauwerte. Diese werden als relative Höhenwerte interpretiert. Große Werte entsprechen „hohen Bereichen“ niedrige Werte entsprechen „tiefen Bereichen“. In dem hieraus entstehenden dreidimensionalen Grauwertgebirge werden Wasserscheiden gesucht. Wasserscheiden sind hellen Grate zwischen dunklen Becken. Der Parameter Basins enthält diese dunklen Becken. Watersheds enthält die Punkte der Wasserscheiden, die höchstens 1 Pixel breit sind. Watersheds ist immer eine Region pro Eingabebild, Basins dagegen enthält für jedes Becken eine eigene Region.

Es ist günstig, vorher einen Glättungsfilter zu verwenden, um die Anzahl der Regionen zu reduzieren (z.B. binomial_filter oder gauss_filter). Eine weitere Möglichkeit, die Anzahl der Ergebnisregionen zu reduzieren, besteht darin, benachbarte Becken aufgrund eines Schwellenwertkriteriums mit watersheds_threshold zu vereinigen (für weitere Informationen siehe watersheds_threshold).

Achtung

Wenn das Bild feine Strukturen enthält oder verrauscht ist, entstehen sehr viele Regionen und damit steigt die Laufzeit deutlich an.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage(-array) ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **Basins** (output_object) ..... region-array ⇝ *object*
  Segmentierte Becken.
- ▷ **Watersheds** (output_object) ..... region(-array) ⇝ *object*
  Wasserscheiden zwischen den Becken.

Beispiel

```
#include "HIOStream.h"
#if !defined(USE_IOSTREAM_H)
using namespace std;
#endif
#include "HalconCpp.h"
using namespace Halcon;

int main (int argc, char *argv[])
{
  HImage   image (argv[1]), gauss;
  HWindow  win;
```

```
  cout << "Gauss of original " << endl;
  gauss  = image.GaussImage (9);
  image.Display (win);

  HRegion        watersheds;
  HRegionArray   basins = gauss.Watersheds (&watersheds);

  win.SetColored (12);
  basins.Display (win);
  win.Click ();

  return (0);
}
```

Ergebnis

`watersheds` liefert normalerweise den Wert TRUE. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe `set_system`). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`binomial_filter`, `gauss_filter`, `smooth_image`, `invert_image`

Nachfolger

`expand_region`, `select_shape`, `reduce_domain`, `opening`

Alternativen

`watersheds_threshold`, `pouring`, `watersheds_marker`

Literatur

L. Vincent, P. Soille: „Watersheds in Digital Space: An Efficient Algorithm Based on Immersion Simulations“; IEEE Transactions on Pattern Analysis and Machine Intelligence; vol. 13, no. 6; pp. 583-598; 1991.

Modul

Foundation

watersheds_marker (Image, Markers : Basins : :)

Segmentieren nach dem Marker-basierten Wasserscheiden-Prinzip.

`watersheds_marker` segmentiert ein Bild basierend auf der Topologie der Grauwerte und gibt die resultierenden Regionen in `Basins` zurück. Das Bild wird dabei als „Grauwertgebirge“ interpretiert. Höhere Grauwerte entsprechen „Bergen“, während niedrigere Grauwerte „Tälern“ entsprechen. Das so interpretierte Bild wird geflutet, beginnend mit den Bereichen die in `Markers` definiert sind. Dabei wird jedes Pixel dem Marker zugeordnet, zu dem es den Pfad mit der kleinsten maximalen Höhe hat. Gibt es mehrere solche Marker, zu denen ein Pfad mit kleinster maximaler Höhe existiert, wird das Pixel zufällig einem davon zugeordnet.

Die resultierenden Becken (eines pro Eingangsbereich in `Markers`) werden in `Basins` zurückgegeben. Beachten Sie, dass einige dieser Bereiche leer sein können. Dies kann der Fall sein, wenn sich zwei Markerregionen im selben Becken des Grauwertgebirges befinden. Befindet sich ein Pixel in mehreren Marker-Startregionen, so wird es nur der letzten zugeordnet. Es wird allerdings dazu geraten, dass sich Markerregionen nicht überschneiden.

Die Überflutung wird nur für die Domäne von `Image` durchgeführt. Markierte Regionen außerhalb des Bereichs von `Image` werden ignoriert.

Es ist ratsam, einen Glättungsoperator auf dem Eingangsbild anzuwenden (z.B, binomial_filter oder gauss_filter) bevor watersheds_marker aufgerufen wird. Um ein Bild zu segmentieren ist es oft sinnvoll, zunächst die Kanten des Bildes zu berechnen (z.B. mit edges_image oder edges_color) und das sich daraus ergebende Amplitudenbild in watersheds_marker zu übergeben.

Achtung

Wenn das Bild feine Strukturen enthält oder verrauscht ist, werden intern sehr viele Regionen verarbeitet, wodurch die Laufzeit deutlich ansteigt.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Eingabebild.
- ▷ **Markers** (input_object) ..... region-array ⇝ *object*
  Initiale Marker von denen aus das Bild geflutet wird.
- ▷ **Basins** (output_object) ..... region-array ⇝ *object*
  Segmentierte Becken.

Beispiel

```
read_image (Image, 'printer_chip/printer_chip_01')
edges_image (Image, ImaAmp, ImaDir, 'canny', 1, 'nms', 20, 40)

* Compute background marker
full_domain (ImaAmp, ImageFull)
erosion_circle (Image, RegionErosion, 2.5)
difference (Image, RegionErosion, Boundary)

* Compute foreground markers
threshold (Image, Region, 200, 255)
erosion_circle (Region, RegionErosion1, 7.5)
connection (RegionErosion1, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 1500, 99999)

* Apply marker-based watersheds
concat_obj (Boundary, SelectedRegions, MarkerRegions)
watersheds_marker (ImageFull, MarkerRegions, Basins)

* Display results
dev_display (Image)
dev_set_color (['#FF000055', '#00FF0055', '#0000FF55', '#FF00FF55', '#FFFF0055', '#00FFF
dev_display (MarkerRegions)
dev_set_draw ('fill')
dev_display (Basins)
```

Ergebnis

watersheds_marker liefert normalerweise den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

binomial_filter, gauss_filter, smooth_image, invert_image, edges_image

Nachfolger

expand_region, select_shape, reduce_domain, opening

Alternativen

watersheds, pouring, watersheds_threshold

Modul

Foundation

watersheds_threshold (Image : Basins : Threshold :)

Segmentieren nach dem Wasserscheiden-Prinzip unter Verwendung eines Schwellenwertes.

Der Operator watersheds_threshold segmentiert Regionen, die durch Wasserscheiden voneinander getrennt sind. Die Mindesthöhe der Wasserscheiden kann durch Threshold festgelegt werden.

Im ersten Schritt berechnet watersheds_threshold die Wasserscheiden und die zugehörigen dunklen Becken ohne Berücksichtigung eines Schwellenwertes, was einem Aufruf von watersheds entspricht (weitere Informationen sind bei watersheds zu finden). Im zweiten Schritt werden die Becken sukzessive vereinigt, falls sie durch eine Wasserscheide voneinander getrennt sind, die kleiner ist als Threshold. Seien B_1 und B_2 die Minima zweier benachbarter Becken und W der kleinste Grauwert der Wasserscheide, die die beiden Becken voneinander trennt. Die Wasserscheide wird eliminiert und die beiden Becken werden vereinigt, wenn gilt:

$$\max\{W - B_1, W - B_2\} < \texttt{Threshold}.$$

Die so erhaltenen Becken werden in Basins zurückgeliefert.

Wird für Threshold der Wert *0* übergeben, liefert watersheds_threshold ähnliche Ergebnisse wie watersheds. Allerdings werden nur die expandierten Becken, nicht aber die Wasserscheiden selbst zurückgegeben. Wird Threshold dagegen auf den maximalen Grauwertbereich von Image gesetzt, so gibt es keine zwei Becken, die durch eine Wasserscheide der Höhe Threshold voneinander getrennt sind. Folglich werden alle Becken zu einem vereinigt, d.h. Basins enthält nur eine Region.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Zu segmentierendes Eingabebild.
- ▷ **Basins** (output_object) ..... region-array ⇝ *object*
  Gefundene Segmente (dunkle Becken).
- ▷ **Threshold** (input_control) ..... number ⇝ *integer* / real
  Schwellenwert für die Wasserscheiden.
  Defaultwert: 10
  Wertevorschläge: Threshold ∈ {0, 5, 10, 20, 30, 50}
  Restriktion: Threshold >= 0

Ergebnis

watersheds liefert normalerweise den Wert TRUE. Für das Verhalten bzgl. des Eingabebildes und der Ausgaberegionen sind die Flags *'no_object_result'*, *'empty_region_result'* und *'store_empty_region'* einstellbar (siehe set_system). Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

binomial_filter, gauss_filter, smooth_image, invert_image

Nachfolger

expand_region, select_shape, reduce_domain, opening

Alternativen

watersheds, pouring, watersheds_marker

Modul

Foundation

Kapitel 24

Steuerung

assign (: : Input : Result)

Weist einer Variablen einen Wert zu.

assign weist einer Variablen einen neuen Wert zu.

Im Freitexteditor wird eine Zuweisung mithilfe des Zuweisungsoperators ':=' eingegeben, zum Beispiel:

```
u := sin(x) + cos(y)
```

Dies entspricht der folgenden Zuweisung in C:

```
u = sin(x) + cos(y);
```

Wird eine Zuweisung im Operatorfenster eingegeben, muss assign in die Operatorauswahlbox eingetragen werden. Daraufhin öffnet sich der Parameterbereich, in dem der Parameter Input dem zu berechnenden Ausdruck, d.h. der rechten Seite der Zuweisung entspricht. Der Parameter Result enthält den Namen der Variablen, also die linke Seite der Zuweisung.

Achtung

Zusätzlich zu dem in der Parameterdokumentation genannten Parametertyp control unterstützt assign auch ikonische Variablen und Vektorvariablen. Die Parametertypen der beiden Parameter Input und Result müssen dabei übereinstimmen. Für die Zuweisung von ikonischen Variablen wird intern der Operator copy_obj verwendet.

Parameter

- ▷ **Input** (input_control) ..... real(-array) ⇝ *real* / integer / string
  Neuer Wert.
  Defaultwert: 1
- ▷ **Result** (output_control) ..... real(-array) ⇝ *real* / integer / string
  Zu ändernde Variable.

Beispiel

```
Tuple1 := [1,0,3,4,5,6,7,8,9]
Val := sin(1.2) + cos(1.2)
Tuple2 := []
```

Ergebnis

Ist der Ausdruck korrekt, dann liefert assign den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

insert

Modul

Foundation

assign_at (: : Index, Value : Result)

Zuweisung eines oder mehrerer Werte an ein oder mehrere Tupelelemente.

assign_at weist einem oder mehreren Tupelelementen einen neuen Wert zu, oder es weist mehrere Werte elementweise den angegebenen Tupelelementen zu. Alle anderen Elemente des Ausgabetupels behalten ihren Wert. Werden Indizes übergeben, die außerhalb der aktuellen Größe des Ausgabtupels liegen, wird das Tupel entsprechend vergrößert und die nicht gesetzten Werte werden mit einem Defaultwert initialisiert.

Im Freitexteditor wird der assign_at-Operator einfach mithilfe des Zuweisungsoperators ':=' und des Indexoperators '[ ]' erstellt. Als Index Parameter ist jeder Ausdruck erlaubt, der eine beliebige Anzahl von positiven Integerwerten ergibt. Als Value Parameter muss entweder genau ein Wert übergeben werden, oder es müssen genauso viele Werte wie Indizes übergeben werden:

```
Areas[Radius-1]     := Area
Areas[0,4,|Rad|-1] := 0
FileNames[0,2,4]    := ['f1','f2','f3']
```

Der Operator assign_at ersetzt und erweitert die modifizierende Variante des bisherigen insert Operators.

Parameter

- ▷ **Index** (input_control) ..... integer(-array) ⇝ *integer*
  Indizes der Tupelelemente, die durch die übergebenen Werte ersetzt werden sollen.
  Defaultwert: 0
  Wertevorschläge: Index ∈ {0, 1, 2, 3, 4, 5, 6}
  Minimale Schrittweite: 1
- ▷ **Value** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Werte, die im Tupel gesetzt werden sollen.
  Defaultwert: 1
- ▷ **Result** (output_control) ..... tuple(-array) ⇝ *real* / integer / string
  Ergebnistupel mit den übergebenen Werten.

Ergebnis

Ist der Ausdruck korrekt, dann liefert assign_at den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

assign, tuple_replace

Modul

Foundation

break (: : :)

Verlassen einer Schleife oder eines switch-Blocks.

Mit Hilfe der break-Anweisung ist es möglich, an einer beliebigen Stelle eine for-, while- oder repeat..until-Schleife zu verlassen. Darüber hinaus wird die break-Anweisung verwendet, um einen switch-Block zu verlassen, insbesondere am Ende eines case-Zweiges. Dabei bezieht sich die break-Anweisung immer auf den kleinsten umschließenden Block, nur dieser kann durch ein break in verschachtelten Blöcken verlassen werden. Das Programm wird mit der nächsten Anweisung hinter dem korrespondierenden Blockende fortgesetzt.

break-Anweisung außerhalb von Schleifen- oder switch-Blöcken sind ungültig.

Beispiel

```
read_image (Image, 'monkey')
threshold (Image, Region, 160, 180)
connection (Region, Regions)
Number := |Regions|
```

```
AllRegionsValid := 1
* check if for all regions area <=30
for i := 1 to Number by 1
  select_obj (Regions, ObjectSelected, i)
  area_center (ObjectSelected, Area, Row, Column)
  if (Area > 30)
    AllRegionsValid := 0
    break
  endif
endfor
```

Ergebnis

`break` (als Operatoren) liefern immer TRUE zurück.

Alternativen

`continue`

Siehe auch

`for`, `while`, `repeat`, `until`, `switch`, `case`

Modul

Foundation

`case` `( : : Constant : )`

Sprungmarke in einem `switch`-Block.

`case` definiert eine Sprungmarke in einem `switch`-Block. Diese Marke wird angesprungen, wenn der in der `switch`-Anweisung berechnete Wert mit der in der `case`-Anweisung im Parameter `Constant` definierten Integerkonstante übereinstimmt. Der `Constant`-Parameter muss zwingend eine Integerkonstante enthalten. Ausdrücke und andere Datentypen sind hier nicht erlaubt. Soll für mehrere unterschiedliche `switch`-Werte der gleiche Programmcode ausgeführt werden, können direkt vor dem auszuführenden Codeblock mehrere `case`-Anweisungen mit unterschiedlichen Konstanten direkt aufeinander folgen.

Wie in C, C++ und C# leitet die `case`-Anweisung keinen Block ein, der beim nächsten `case` oder `default` automatisch verlassen wird, sondern stellt lediglich eine Sprungmarke dar, die bei passendem `switch`-Wert angesprungen wird. Um einen `case`-Zweig am Ende zu verlassen und nicht unbeabsichtigt über die nächste `case`- oder `default`-Anweisung zu laufen, muss der `switch`-Block mit Hilfe einer `break`-Anweisung verlassen werden.

Parameter

▷ **`Constant`** (input_control) . . . . . integer ⇝ *integer*
Integerkonstante, die bestimmt, bei welchem `switch`-Wert die Anweisung angesprungen wird.
Defaultwert: 1

Ergebnis

`case` (als Operator) liefert immer TRUE zurück.

Alternativen

`elseif`

Siehe auch

`switch`, `default`, `endswitch`, `if`

Modul

Foundation

`catch` `( : : : Exception )`

Fängt im vorangehenden `try`-Block geworfene Exceptions.

Die Operatoren try, catch, endtry und throw erlauben es, in HDevelop-Programme Code für eine dynamische Fehlerbehandlung ähnlich zu den Programmiersprachen C++ oder C# einzubauen. Grundlegende Informationen zur Fehlerbehandlung in HDevelop finden sich bei den Operatoren try, throw und dev_set_check sowie im „HDevelop User's Guide".

Der Operator catch trennt den zu überwachenden Programmbereich von dem Code, der im Fehlerfall auszuführen ist. Tritt im vorhergehenden try-catch-Block kein Fehler auf, wird der gesamte catch-endtry-Block übersprungen. Im Fehlerfall dagegen wird von dem Operator, in dem der Fehler aufgetreten ist, (bzw. vom throw-Operator) direkt zum catch des umschließenden try-catch-Blocks gesprungen. Im Ausgabeparameter Exception wird dem catch-Operator ein Tupel mit den Fehlerdaten bzw. bei throw ein vom Benutzer weitestgehend frei definierbares Tupel übergeben.

Das wichtigste Fehlerdatum ist der Fehlercode. Daher wird dieser im Exception-Tupel immer an erster Stelle zurückgeliefert und kann direkt mit *'Exception'*[0] abgefragt werden. Alle anderen Daten müssen über den Operator dev_get_exception_data abgefragt werden, da sich die Reihenfolge und der Umfang der bereitgestellten Informationen unter Umständen in zukünftigen Versionen ändern und sich beim exportierten Code von Sprache zu Sprache unterscheiden kann. Insbesondere ist zu beachten, dass im exportierten Code nicht alle Informationen verfügbar sind, sowie dass es Informationen gibt, die möglicherweise erst bei einem expliziten Zugriff darauf bereitgestellt werden (wie z.B. Fehlertexte).

Beim Fehlercode handelt es sich bei Operatoraufrufen immer um einen HALCON-Fehlercode (< 10000) bzw. beim Aufruf von Operatoren aus einem Extension Package um einen anwenderdefinierten Fehlercode (> 10000). Die HALCON-Fehlercodes werden im Anhang des „Extension Package Programmer's Manual" aufgelistet. Wird mit throw ein benutzerdefiniertes Exception-Tupel geworfen, sollte dieses als erstes Element immer einen Fehlercode $\geq$ 30000 als Kennzeichnung enthalten. Alle weiteren Elemente können vollkommen frei gesetzt werden.

Tritt in HDevelop oder der HDevEngine ein Operatorfehler auf, werden im Exception-Tupel die folgenden Informationen über den Fehler und den Ort seines Auftretens zur Verfügung gestellt:

- der HALCON-Fehlercode,
- ein zusätzlicher HDevelop-spezifischer Fehlercode, der darüber informiert, ob der Fehler innerhalb des HALCON-Operators generiert wurde (Fehlercode = 21000), oder ob er bereits bei der Übergabe der Parameter, d.h. bei der Auswertung der Parameterausdrücke aufgetreten ist. Im zweiten Fall wird mit diesem Fehlercode die Art des Fehlers genauer spezifiziert. Die relevanten HDevelop-Fehlercodes werden beim Operator dev_get_exception_data aufgelistet.
- der HALCON-Fehlertext,
- ein passender HDevelop-spezifischer Fehlertext,
- die Zeilennummer, in der der Fehler aufgetreten ist,
- der Name des Operators (tritt der Fehler in einer geschützten Prozedur auf, wird '--protected--' zurückgeliefert),
- die Aufruftiefe ('main' hat die Tiefe 1),
- der Name der Prozedur.

Für eine automatische Fehlerbehandlung ist vor allem der HALCON-Fehlercode von Bedeutung. Die weiteren Daten haben in erster Linie informativen Charakter. Achtung: Im exportierten Code stehen die Informationen über den Fehlerort nicht oder nur eingeschränkt zur Verfügung.

Achtung

Der Export der Operatoren try, catch, endtry und throw ist nur in die Sprachen C++, C# und VisualBasic/.NET möglich, da nur sie das Werfen und Fangen von Exceptions über Prozedurgrenzen hinweg unterstützen. Beim Export nach C können diese Operatoren nicht unterstützt werden.

Parameter

▷ **Exception** (output_control) . . . . . exception-array ⇝ *integer* / string
Tupel mit den Fehlerdaten.

Ergebnis

catch liefert immer TRUE zurück.

Nachfolger

dev_get_exception_data

Siehe auch

try, endtry, throw, dev_get_exception_data, dev_set_check

Modul

Foundation

comment (: : Comment :)

Kommentar von einer Zeilenlänge.

Kommentar von einer Zeilenlänge. Als Parameter, d.h. als Kommentar sind alle Zeichen zulässig. Sie müssen nicht, wie bei einem String, in Hochkommata gesetzt werden. Zeilenumbrüche, die in dieser Zeile vorkommen, werden in mehrere Kommentarzeilen umgesetzt.

Im Freitexteditor werden Kommentarzeilen durch einen Stern ('*') als erstes druckbares Zeichen gekennzeichnet.

Dieser Operator hat keinen Einfluss auf den Programmablauf.

Parameter

▷ **Comment** (input_control) . . . . . string ⇝ *string*
Beliebige Zeichenfolge.

Beispiel

```
* This is a program with comments
* 'this is a string as comment'
* here are numbers: 4711, 0.815
stop ()
```

Ergebnis

comment wird nicht ausgeführt.

Modul

Foundation

continue (: : :)

Überspringt den aktuellen Schleifendurchlauf.

continue überspringt den umschließenden for, while oder repeat..until Schleifendurchlauf. Die Ausführung des Programms wird bei der Bedingungszeile der umschließenden Schleife oder bei der nächsten Zeile nach continue fortgesetzt, falls keine umschließende Schleife existiert.

Ergebnis

continue (als Operatoren) liefern immer TRUE zurück.

Alternativen

break

Siehe auch

for, while, repeat, until

Modul

Foundation

convert_tuple_to_vector_1d (: : InputTuple,
SubTupleLength : ResultVector)

Verteilt die Elemente eines Tupels auf einen Vektor.

Mit Hilfe des Befehls convert_tuple_to_vector_1d wird ein Tupel in einen Vektor umgewandelt. Dabei werden nacheinander aus jeweils SubTupleLength Werten aus dem Eingabetupel InputTuple Teiltupel gebildet und diese im Ergebnisvektor ResultVector abgelegt.

Parameter

- ▷ **InputTuple** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Eingabetupel.
- ▷ **SubTupleLength** (input_control) . . . . . number ⇝ *integer*
  Gewünschte Länge der Tupel im Ausgabevektor.
  Defaultwert: 1
- ▷ **ResultVector** (output_control) . . . . . number-vector1 ⇝ *real* / integer / string
  Ausgabevektor.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert convert_tuple_to_vector_1d den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Siehe auch

convert_vector_to_tuple

Modul

Foundation

convert_vector_to_tuple (: : InputVector : ResultTuple)

Verkettet die Elemente eines Vektors zu einem Tupel.

convert_vector_to_tuple wandelt ein Vektor in ein Tupel um. Dabei werden die einzelnen Elemente des Eingabevektors InputVector zu einem einzelnen Ausgabetupel ResultTuple aneinandergereiht. Ist InputVector zwei- oder mehrdimensional, werden die Vektorelemente in einer Tiefensuche aufgesammelt. So wird z.B. aus dem Eingabevektor *'1,2,3,4'* das Ausgabetupel *[1,2,3,4]*.

Parameter

- ▷ **InputVector** (input_control) . . . . . number-vector ⇝ *real* / integer / string
  Eingabevektor.
- ▷ **ResultTuple** (output_control) . . . . . real(-array) ⇝ *real* / integer / string
  Ausgabetupel.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert convert_vector_to_tuple den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Siehe auch

convert_tuple_to_vector_1d

Modul

Foundation

default (: : :)

Alternative einer Mehrfachverzweigung.

default leitet in einem switch-Block den Alternativzweig ein, der angesprungen wird, wenn keine der vorherigen case-Anweisungen zum berechneten Wert der switch-Anweisung passt.

Ergebnis

default (als Operator) liefert immer TRUE zurück.

Alternativen

case, elseif, else

Siehe auch

switch, case, endswitch, if

Modul

Foundation

else (: : :)

Alternative einer bedingten Anweisung.

`else` setzt einen durch `if` oder `elseif` eingeleiteten Block als Alternativzweig fort. Falls die Bedingungen aller zugehörigen mit `if` oder `elseif` eingeleiteten Blöcke den Wert *'false'* (*0*) ergeben, also keiner der zugehörigen `if` oder `elseif` Blöcke ausgeführt wurde, wird der durch `else` eingeleitete Block ausgeführt.

Ergebnis

`else` (als Operator) liefert immer TRUE zurück.

Alternativen

`if`, `elseif`

Siehe auch

`until`, `for`, `while`

Modul

Foundation

elseif (: : Condition :)

Bedingte Anweisung mit Alternative.

`elseif` ist eine bedingte Anweisung, die an einen durch `if` oder `elseif` eingeleiteten Block als Alternativzweig anschließt. Der Parameter `Condition` muss einen booleschen oder Integerwert ergeben.

Falls `Condition` *'true'* (also einen Wert ungleich *0*) ergibt, wird der folgende Programmabschnitt bis zum nächsten zugehörigen `elseif`, `else` oder `endif` ausgeführt. Daraufhin wird die Ausführung hinter der schließenden Anweisung `endif` fortgesetzt.

Falls `Condition` *'false'* (*0*) ergibt, wird die Ausführung ab der nächsten zugehörigen Anweisung `elseif`, `else` oder `endif` fortgesetzt.

Parameter

▷ **Condition** (input_control) . . . . . integer ⇝ *integer*
Bedingung für die if-Anweisung.
Defaultwert: 1

Ergebnis

Enthält die Bedingung einen korrekten Ausdruck, dann liefert `elseif` (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`if`

Siehe auch

`else`, `for`, `while`, `until`

Modul

Foundation

endfor (: : :)

Abschluss einer for-Schleife.

`endfor` bildet den Abschluss einer `for`-Schleife.

Ergebnis

`endfor` liefert immer TRUE zurück.

Siehe auch

for

Modul

Foundation

endif (: : :)

Abschluss einer if-Anweisung.

endif bildet den Abschluss eines if, elseif oder else Blocks.

Ergebnis

endif liefert immer TRUE zurück.

Siehe auch

if

Modul

Foundation

endswitch (: : :)

Abschluss einer mit switch eingeleiteten Mehrfachvererbung.

endswitch bildet den Abschluss einer mit switch eingeleiteten Mehrfachvererbung.

Ergebnis

endswitch liefert immer TRUE zurück.

Siehe auch

switch

Modul

Foundation

endtry (: : :)

Beendet einen Fehlerbehandlungsblock.

Die Operatoren try, catch, endtry und throw erlauben es, in HDevelop-Programme Code für eine dynamische Fehlerbehandlung ähnlich zu den Programmiersprachen C++ oder C# einzubauen. Grundlegende Informationen zur Fehlerbehandlung in HDevelop finden sich bei den Operatoren try, throw und dev_set_check sowie im „HDevelop User's Guide".

Der Operator endtry beendet einen Fehlerbehandlungsblock, der zuvor mit den Operatoren try und catch eingeleitet wurde.

Achtung

Der Export der Operatoren try, catch, endtry und throw ist nur in die Sprachen C++, C# und VisualBasic/.NET möglich, da nur sie das Werfen und Fangen von Exceptions über Prozedurgrenzen hinweg unterstützen. Beim Export nach C können diese Operatoren nicht unterstützt werden.

Ergebnis

endtry liefert immer TRUE zurück.

Siehe auch

try, catch, throw, dev_get_exception_data, dev_set_check

Modul

Foundation

endwhile (: : :)

Abschluss einer while-Schleife.

`endwhile` bildet den Abschluss einer `while`-Schleife.

Ergebnis

`endwhile` liefert immer TRUE zurück.

Siehe auch

`while`

Modul

Foundation

executable_expression (: : Expression :)

Führt eine allein stehende Operation aus.

Die HDevelop-Sprache kennt einige Operationen, die allein stehend ausgeführt werden. Mit Hilfe des Befehls `executable_expression` lassen sich derartige Operationen im Operatorfenster von HDevelop eingeben. Im Freitexteditor hingegen können allein stehende Operationen direkt eingegeben werden.

Derzeit gibt es die folgenden allein stehenden Vektoroperationen:

- `.clear()`
- `.insert()`
- `.remove()`

Nähere Informationen zu diesen Operationen entnehmen Sie bitte dem HDevelop User's Guide.

Auch wenn `Expression` formal ein Kontrollparameter ist, können auch allein stehende ikonische Vektoroperationen ausgeführt werden.

Parameter

▷ **Expression** (input_control) . . . . . number-vector ⇝ *real* / integer / string
Ausführbare Operation.

Beispiel

```
read_image (Image1, 'fin1')
read_image (Image2, 'fin2')
ImageVector.insert(1, Image1).insert(2, Image2)
* process vector
ImageVector.clear()
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `executable_expression` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Modul

Foundation

exit (: : :)

Beenden von HDevelop.

`exit` beendet HDevelop. Der Operator ist äquivalent zu dem Menüpunkt `Datei ▷ Beenden`. Intern und beim Export zu C++ wird die C-Funktion `exit(0)` verwendet.

Beispiel

```
read_image (Image, 'fabrik')
intensity (Image, Image, Mean, Deviation)
open_file ('intensity.txt', 'output', FileHandle)
fwrite_string (FileHandle, Mean + ' ' + Deviation)
close_file (FileHandle)
exit ()
```

Ergebnis

`exit` liefert 0 (o.k.) an die Umgebung, aus der heraus HDevelop gestartet wurde (das Betriebssystem), zurück.

Siehe auch

`stop`

Modul

Foundation

export_def (: : Position, Declaration :)

Fügt beliebigen Text in den Codeexport ein.

`export_def` erlaubt es, beliebige Texte beim Export von Programmen oder Prozeduren in die Ausgabedatei einzufügen.

Der Parameter `Position` steuert, wo der Text, der in `Declaration` gesetzt wird, in die Exportdatei eingefügt werden soll. Dabei werden die folgenden Optionen unterstützt:

'in_place' **- #** Der Text wird an der gleichen Stelle in die exportierte Prozedur eingefügt, an der er auch im HDevelop-Code steht.

'at_file_begin' **- #^^** Der Text wird ganz am Anfang der exportierten Datei eingefügt.

'before_procedure' **- #^** Der Text wird unmittelbar vor der Prozedur, in der der `export_def` Befehl steht, in die exportierte Datei eingefügt.

'after_procedure' **- #$** Der Text wird unmittelbar nach der Prozedur, in der der `export_def` Befehl steht, in die exportierte Datei eingefügt.

'at_file_end' **- #$$** Der Text wird ganz am Ende der exportierten Datei eingefügt.

Im Programmlisting wird der Befehl `export_def` nicht in der normalen Operatorschreibweise ausgegeben, sondern durch eine spezielle Zeichenfolge eingeleitet. Das erste Zeichen ist das Exportzeichen `#`, das entsprechend obiger Auflistung von einer Positionskennung gefolgt werden kann. Soll ein `export_def` Befehl im Freitexteditor eingegeben werden, ist zu beachten, dass vor dem Exportzeichen `#` kein Leerzeichen stehen darf.

Um die Lesbarkeit zu verbessern kann zwischen der Exportkennzeichnung und dem zu exportierenden Text ein Leerzeichen stehen, das nicht mit exportiert wird. Zusätzliche Leerzeichen werden dagegen mit exportiert.

Texte, die innerhalb der aktuellen Prozedur ausgegeben werden, werden entsprechend der aktuellen Einrücktiefe eingerückt. Dabei gibt es eine Ausnahme: Fängt der zu exportierende Text mit `#` an, wird der Text nicht eingerückt:

```
for Index := 1 to 5 by 1
# #ifdef MY\_SWITCH
# int cnt = 100;
  * ein optionaler Codeblock
# #endif
endfor
```

Wird exportiert zu:

```
proc (...)
{
```

```
  ...
  for (...)
  {
#ifdef MY\_SWITCH
    int cnt = 100;
    // ein optionaler Codeblock
#endif
  }
  ...
}
```

Der Befehl `export_def` kann wie jeder normale Operator aktiviert oder deaktiviert werden. Deaktivierte Exportbefehle werden nicht exportiert.

Parameter

▷ **`Position`** (input_control) . . . . . string ⇝ *string*
Position, an der Exporttext in die Ausgabedatei eingefügt werden soll.
Werteliste: `Position` ∈ {'in_place', 'at_file_begin', 'before_procedure', 'after_procedure', 'at_file_end'}

▷ **`Declaration`** (input_control) . . . . . string ⇝ *string*
Zu exportierender Text.

Ergebnis

`export_def` wird nicht ausgeführt.

Siehe auch

`comment`

Modul

Foundation

`for` `( : : Start, End, Step : Index )`

Eröffnet einen Schleifenblock, der i.d.R. mit einer festen Anzahl an Durchläufen ausgeführt wird.

Schreibweise in HDevelop: `for Index := Start to End by Step`

`for` eröffnet einen Schleifenblock, der i.d.R. mit einer festen Anzahl an Durchläufen ausgeführt wird. Der `for`-Block erstreckt sich bis zur zugehörigen `endfor`-Anweisung.

Die Anzahl der Durchläufe wird durch die Werte für `Start`, `End` und die Schrittweite `Step` festgelegt. Diese Parameter können neben konstanten Werten auch durch Ausdrücke oder Variablen initialisiert werden. Es ist zu beachten, dass die Schleifenparameter nur am Anfang, beim Betreten der `for`-Schleife ausgewertet werden. Innerhalb der `for`-Schleife haben Änderung an den Variablen, die zur Initialisierung der Schleife verwendet wurden keinen Einfluss auf die Anzahl an Durchläufen.

Die übergebenen Schleifenparameter müssen vom Typ `integer` oder `real` sein. Sind alle Eingabevariablen vom Typ `integer`, so ist die Variable `Index` ebenfalls vom Typ `integer`. In allen anderen Fällen ist die Variable `Index` vom Typ `real`.

Vor jedem Schleifendurchlauf wird der Wert der Laufvariablen `Index` mit dem Endwert verglichen. Ist – bei einer positiven Schrittweite – der Wert größer als der (bei der Initialisierung gesetzte) Endwert, wird die `for`-Schleife beendet und das Programm hinter der korrespondierenden `endfor`-Anweisung fortgesetzt. Bei einer negativen Schrittweite muss die `Index`-Variable kleiner als der Endwert sein, damit die Schleife abgebrochen wird.

Achtung: Ist die Schrittweite vom Typ `real` kann es passieren, dass – wenn auf Gleichheit zwischen Laufvariable und Endwert getestet wird – der letzte Schleifendurchlauf aufgrund von Rundungsfehlern nicht mehr ausgeführt wird. So wird auf manchen Systemen die folgende Schleife nicht wie erwartet vier mal durchlaufen (mit den Werten 1.3, 1.4, 1.5 und 1.6), sondern nur drei mal, da die Indexvariable nach drei Additionen mit 0.1 etwas größer als 1.6 ist:

```
I:=[]
for Index := 1.3 to 1.6 by 0.1
  I := [I,Index]
endfor
```

Nach dem Abarbeiten der Anweisungen des Schleifenkörpers, d.h. nach dem Erreichen der korrespondierenden endfor-Anweisung oder einer continue-Anweisung, wird die (bei der Initialisierung gesetzte) Schrittweite auf den aktuellen Wert der Index-Variable aufaddiert, und die Schleifenbedingung wie oben beschrieben erneut ausgewertet. Das Ergebnis der Auswertung bestimmt, ob die Schleife durch einen Sprung auf die erste Anweisung des Schleifenkörpers erneut ausgeführt wird, oder ob die Schleife durch einen Sprung auf die erste Anweisung hinter der korrespondierenden endfor-Anweisung beendet wird.

Der Aufruf einer break-Anweisung innerhalb des Schleifenkörpers bewirkt das sofortige Verlassen der for-Schleife. Das Programm wird hinter der korrespondierenden endfor-Anweisung fortgesetzt. Eine continue-Anweisung dagegen führt dazu, dass der Rest des Schleifenkörpers ignoriert wird und das Program direkt mit dem Anpassen der Laufvariablen und der Auswertung der Schleifenbedingung fortgesetzt wird.

Achtung: Das Verhalten bezüglich der Index-Variable hat sich mit HALCON 11 geändert: In älteren Versionen wurden Änderungen an der Index-Variable innerhalb des Schleifenkörpers von der for-Anweisung ignoriert. Dafür wurde die Index-Variable von der for-Anweisung vor dem Auswerten der Schleifenbedingung auf einen internen Iterationswert zurückgesetzt. Dieser wurde entsprechend der folgenden Formel bestimmt:

```
Index := Start + count * Step
```

wobei count die Anzahl der bereits absolvierten Schleifendurchläufe ist.

Damit Programme und Prozeduren, die mit einer älteren HALCON-Version (vor HALCON 11) erstellt und in einer neueren HALCON-Version geladen wurden, nicht automatisch ihr Verhalten ändern, werden beim Laden von älteren Programmen die for-Schleifen analysiert. Wird festgestellt, dass im Schleifenkörper die Index-Variable modifiziert wird, wird die for-Anweisung in einen Kompatibilitätsmodus versetzt und entsprechend markiert. Eine so markierte for-Schleife wird entsprechend der alten Semantik ausgeführt und im Listing mit einem Warndreieck und dem Zusatz __use_internal_index__ am Zeilenende versehen. Im Operatorfenster erscheint ebenfalls eine Warnung und eine Checkbox, die es erlaubt, den Kompatibilitätsmodus für die for-Schleife aufzuheben und die Schleife entsprechend der neuen Semantik auszuführen. Im Freitexteditor kann das Gleiche durch Löschen des Zusatzes __use_internal_index__ erreicht werden. Damit sich das Programm danach nicht anders verhält als vorher, ist es allerdings notwendig, auch den Schleifenkörper so zu modifizieren, dass innerhalb der Schleife nicht mehr die Laufvariable verändert wird, sondern mit einer lokalen Kopie von dieser gearbeitet wird.

Es wird in jedem Fall empfohlen, darauf zu verzichten, die Laufvariable innerhalb der Schleife zu verändern, da dadurch zum einen der Programmcode schwieriger verständlich wird, und zum anderen ein solches Programm sich auf älteren HALCON-Versionen anders verhalten würde.

Wird die for-Schleife durch eine stop-Anweisung oder durch Drücken des Knopfes Stop angehalten und die Ausführungsmarke manuell umgesetzt, so wird die for-Schleife in der aktuellen Iteration fortgeführt, solange die Ausführungsmarke nur innerhalb der for-Schleife oder auf die endfor-Anweisung gesetzt wird. Wird die Ausführungsmarke auf oder vor die for-Anweisung gesetzt und die Ausführung fortgesetzt, so wird die Schleife neu initialisiert und erneut komplett durchlaufen.

Parameter

- ▷ **Start** (input_control) . . . . . number ⇝ *integer* / real
  Startwert des Schleifenzählers.
  Defaultwert: 1
- ▷ **End** (input_control) . . . . . number ⇝ *integer* / real
  Endwert des Schleifenzählers.
  Defaultwert: 5
- ▷ **Step** (input_control) . . . . . number ⇝ *integer* / real
  Änderung bei jedem Schleifendurchgang
  Defaultwert: 1
- ▷ **Index** (output_control) . . . . . number ⇝ *integer* / real
  Laufvariable.

Beispiel

```
read_image (Image, 'fabrik')
threshold (Image, Region, 128, 255)
connection (Region, ConnectedRegions)
```

```
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 150, 99999)
area_center (SelectedRegions, Area, Row, Column)
dev_close_window ()
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
dev_display (Image)
dev_display (SelectedRegions)
dev_set_color ('white')
for Index := 0 to |Area| - 1 by 1
  set_tposition (WindowHandle, Row[Index], Column[Index])
  write_string (WindowHandle, 'Area=' + Area[Index])
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `for` (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`while`, `until`

Siehe auch

`repeat`, `break`, `continue`, `endfor`

Modul

Foundation

global (: : Declaration :)

Deklaration einer globalen Variablen.

Mit Hilfe des Befehls `global` ist es möglich, eine globale Variable zu deklarieren. Globale Variablen sind in allen Prozeduren sichtbar, in denen sie entsprechend deklariert wurden.

Alle Variablen, die innerhalb einer Prozedur nicht explizit als global deklariert werden, sind in jedem Fall lokal, auch wenn in einer anderen Prozedur eine globale Variable mit dem gleichen Namen deklariert wurde.

Der Parameter `Declaration` beinhaltet die vollständige Variablendeklaration, bestehend aus dem optionalen Schlüsselwort *'def'*, dem Typnamen *'object'* oder *'tuple'*, dem optionalen Schlüsselwort *'vector'* (unter Angabe der gewünschten Dimension in runden Klammern) und dem Variablennamen.

Ikonische Variablen werden mit dem Schlüsselwort *'object'* deklariert, für Steuervariablen wird *'tuple'* verwendet.

Das optionale Schlüsselwort *'def'* erlaubt es, eine bestimmte Variablendeklaration explizit als Definition zu kennzeichnen. In den meisten Fällen ist das nicht notwendig, da in HDevelop die globale Variableninstanz automatisch angelegt wird, wenn die Variable irgendwo deklariert wurde. Werden allerdings die Prozeduren, die globale Variablen verwenden, in eine Programmiersprache exportiert, und erfolgt der Export nicht in eine große Datei, mit allen Prozeduren zusammen, sondern wird jede Prozedur in eine eigene Dateien exportiert, dann ist es notwendig, eine der globalen Variablendeklarationen entsprechend als Definition zu kennzeichnen. In eine Gruppe von Prozedurdateien, die zu einer Bibliothek oder einer Anwendung verlinkt werden sollen, muss es genau eine Definition geben, um sowohl Fehler wegen undefinierter Symbole als auch wegen Mehrfachdefinitionen zu vermeiden.

Im Programmlisting wird die Variablendeklaration nicht in Operatorschreibweise sondern ohne Klammern angegeben, da es sich hierbei um eine Deklaration und nicht um einen Operatoraufruf handelt. Dabei wird die folgende Syntax verwendet:

```
global [def] {object|tuple} [vector(<Dimension>)] <Variablenname>
```

Parameter

▷ **Declaration** (input_control) . . . . . string ⇝ *string*
Variablendeklaration bestehend aus optionalem *'def'*, Typ und Variablennamen
Wertevorschläge: `Declaration` ∈ {'object', 'tuple', 'def object', 'def tuple', 'object vector(1)', 'tuple vector(1)', 'def object vector(1)', 'def tuple vector(1)'}

Ergebnis

`global` wird nicht ausgeführt.

Modul

Foundation

if (: : Condition :)

Bedingte Anweisung.

`if` ist eine bedingte Anweisung, die den Beginn eines `if`-Blocks markiert. Der Parameter `Condition` muss einen booleschen oder Integerwert ergeben.

Falls `Condition` *'true'* (also einen Wert ungleich *0*) ergibt, wird der folgende Programmabschnitt bis zum nächsten zugehörigen `elseif`, `else` oder `endif` ausgeführt. Daraufhin wird die Ausführung hinter der schließenden Anweisung `endif` fortgesetzt.

Falls `Condition` *'false'* (*0*) ergibt, wird die Ausführung ab der nächsten zugehörigen Anweisung `elseif`, `else` oder `endif` fortgesetzt.

Parameter

▷ **Condition** (input_control) ..... integer ⇝ *integer*
Bedingung für die if-Anweisung.
Defaultwert: 1

Ergebnis

Enthält die Bedingung einen korrekten Ausdruck, dann liefert `if` (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`elseif`, `else`

Siehe auch

`for`, `while`, `until`

Modul

Foundation

import (: : ProcedureSource :)

Importieren von einer oder mehreren externen Prozeduren.

Mit Hilfe des Befehls `import` ist es möglich aus einem HDevelop Programm heraus weitere, externe Prozeduren zu importieren. Die importierten Prozeduren stehen dabei nur derjenigen Prozedur zur Verfügung, die auch den `import` Befehl enthält, allen anderen nicht.

`import` Befehle dürfen an jeder beliebigen Stelle in einer Prozedur auftauchen. Die importierten Prozeduren stehen erst ab dem `import` Befehl zur Verfügung und können durch nachfolgende `import` Befehle mit anderen, gleichnamigen Prozeduren überschrieben werden.

```
proc()
* unaufgelöster Prozeduraufruf

import ./the_one_dir
proc()
* wird nach ./the_one_dir/proc.hdvp aufgelöst

import ./the_other_dir
proc()
* wird nach ./the_other_dir/proc.hdvp aufgelöst
```

Der Parameter `ProcedureSource` gibt die Quelle der zu importierenden Prozeduren an. Diese kann entweder ein Dateiverzeichnis oder eine Prozedurbibliotheksdatei bezeichnen. In beiden Fällen kann die Angabe sowohl

als absoluter als auch als relativer Pfad erfolgen. In letzterem Falle sieht HDevelop den Pfad als relativ zu dem Verzeichnis an, das die Prozedur mit dem `import` Befehl enthält. Enthält der Pfad Leerzeichen, so muss er in Anführungszeichen stehen, andernfalls ist die Programmzeile ungültig.

Im Gegensatz zu den System-, Benutzer- und Sitzungsprozedurpfaden werden beim `import` Befehl ausschließlich die angegebenen Dateiverzeichnisse nach Prozeduren durchsucht, nicht jedoch dessen Unterverzeichnisse.

Dabei ist zu beachten, dass `import` ein nicht ausführbarer Befehl ist und `ProcedureSource` deswegen schon beim Laden der Prozedur ausgewertet werden muss. `ProcedureSource` muss also ein konstanter Ausdruck sein, und es ist insbesondere nicht möglich, eine Textvariable in `ProcedureSource` zu übergeben.

Allerdings darf `ProcedureSource` auch Umgebungsvariablen enthalten, die von HDevelop entsprechend aufgelöst werden. Umgebungsvariablen müssen, unabhängig von der verwendeten Plattform, in jedem Fall in Windows Syntax angegeben werden, also z.B. %VARIABLE%.

`import` überprüft nicht, ob der Pfad `ProcedureSource` überhaupt existiert oder ob er eine Prozedurbibliothek oder ein Verzeichnis beschreibt. `import` Befehle mit nicht existierenden oder wirkungslosen Pfaden bleiben daher in jedem Fall gültige Programmzeilen.

Importierte Pfade werden in HDevelop in den Prozedur Einstellungen gesondert aufgeführt. Selbstverständlich können diese Pfade aus den Prozedur Einstellungen heraus nicht geändert noch deaktiviert werden. Prozeduren, die ausschließlich durch `import` Anweisungen verfügbar sind, werden mit einem speziellen Icon gekennzeichnet.

Im Programmlisting wird der `import` Befehl nicht in Operatorschreibweise sondern ohne Klammern angegeben, da es sich hierbei um eine Deklaration und nicht um einen Operatoraufruf handelt.

Parameter

▷ **`ProcedureSource`** (input_control) . . . . . string ⇝ *string*
Verzeichnis oder Prozedurbibliotheksdatei, in denen die zu importierenden Prozeduren gespeichert sind

Ergebnis

`import` wird nicht ausgeführt.

Modul

Foundation

`insert ( : : Input, Value, Index : Result )`

Zuweisung eines Wertes an ein Tupelelement.

`insert` ist veraltet und wird nur aus Gründen der Rückwärtskompatibilität zur Verfügung gestellt.. Die modifizierende Version des `insert` Operators wurde durch den Operator `assign_at` ersetzt. Dieser verwendet im Freitexteditor die gleiche Notation wie `insert`, so dass im Freitexteditor automatisch der richtige Operator verwendet wird. Die nicht-modifizierende Variante des `insert` Operators wurde durch den Operator `tuple_replace` ersetzt.

`insert` weist einem spezifischen Tupelelement einen neuen Wert zu.

Im Freitexteditor wird eine solche Operation einfach mithilfe des Zuweisungsoperators ':=' und des Indexoperators '[ ]' für die Ergebnisvariable eingetragen, zum Beispiel:

```
Areas[Radius-1] := Area
```

Wird eine Einfügeoperation im Operatorfenster eingegeben, muss `insert` in die Operatorauswahlbox eingetragen werden. Daraufhin öffnet sich der Parameterbereich, in dem der Parameter `Value` dem zu berechnenden Ausdruck entspricht, der dem Element `Index` des Tupels `Input` zugewiesen wird. Der Parameter `Result` enthält den Namen der Variablen, der das Ergebnis zugewiesen wird.

Nur für den Fall, dass das mit dem Parameter `Input` übergebene Eingabetupel mit dem als `Result` übergebenen Ausgabetupel identisch ist, kann der Operator `insert` mit der oben dargestellten Schreibweise im Freitexteditor eingegeben werden. In diesem Fall wird das Eingabetupel modifiziert. Die korrekte Operatorschreibweise sieht dann folgendermaßen aus:

```
insert (Areas, Area, Radius-1, Areas)
```

Wenn die Tupel `Input` und `Result` unterschiedlich sind, wird das Eingabetupel nicht modifiziert. In diesem Fall kann im Programmeditor nur die Operatorschreibweise verwendet werden:

```
insert (Areas, Area, Radius-1, Result)
```

Dies entspricht der Schreibweise:

```
Result := Areas
Result[Radius-1] := Area
```

Es ist zu beachten, dass der Operator `insert` kein neues Tupelelement hinzufügt, wenn an der angegebenen Position schon ein Wert gespeichert ist. Vielmehr wird der Wert an der Position `Index` ersetzt. Daher muss für `Value` genau ein Wert (oder ein Ausdruck, der genau einen Wert liefert) übergeben werden.

Wenn der durch `Index` übergebene Wert größer ist als die aktuelle Tupelgröße, wird das Tupel auf die erforderliche Größe gebracht. Die Tupelelemente zwischen dem ursprünglich letzten und dem neuen Element sind undefiniert.

Parameter

- ▷ **`Input`** (input_control) . . . . . real(-array) ⇝ *real* / integer / string
  Tupel, in das ein Wert eingefügt werden soll.
  Defaultwert: []
- ▷ **`Value`** (input_control) . . . . . real ⇝ *real* / integer / string
  Wert, der eingefügt werden soll.
  Defaultwert: 1
  Typischer Wertebereich: $0 \leq$ `Value` ≤ 1000000
- ▷ **`Index`** (input_control) . . . . . integer ⇝ *integer*
  Position an der der Wert eingefügt werden soll.
  Defaultwert: 0
  Wertevorschläge: `Index` $\in \{0, 1, 2, 3, 4, 5, 6\}$
  Minimale Schrittweite: 1
- ▷ **`Result`** (output_control) . . . . . real(-array) ⇝ *real* / integer / string
  Ergebnistupel mit eingefügtem Wert.

Ergebnis

Ist der Ausdruck korrekt, dann liefert `insert` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`assign`

Modul

Foundation

`par_join ( : : ThreadID : )`

Wartet auf Unterthreads die mit Hilfe der `par_start` Kennzeichnung am Anfang einer Programmzeile gestartet wurden.

Der Operator `par_join` erlaubt es, auf Prozeduren und Operatoren zu warten, die mit Hilfe von `par_start` in einem separaten Unterthread parallel zur aufrufenden Prozedur gestartet wurden. Es wird auf die Unterthreads gewartet, deren Thread-IDs als Tupel im Parameter `ThreadID` übergeben wurden.

Achtung: `par_start` ist kein eigener Operator, sondern lediglich ein Kennzeichner, der dem eigentlichen Prozeduraufruf in der gleichen Programmzeile vorangestellt wird: `par_start <ThreadID> :` gefolgt von dem eigentlichen Aufruf.

Parameter

- ▷ **`ThreadID`** (input_control) . . . . . thread_id(-array) ⇝ *integer*
  IDs der Unterthreads, auf die gewartet werden soll.

Beispiel

```
* start two procedures in separate sub threads
par_start <ThreadID1> : producer_proc()
par_start <ThreadID2> : consumer_proc()
* wait until both procedures have finished
par_join ([ThreadID1, ThreadID2])
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `par_join` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Modul

Foundation

repeat (: : :)

Beginn einer repeat..until-Schleife.

`repeat` eröffnet eine `repeat`..`until`-Schleife.

Ergebnis

`repeat` liefert immer TRUE zurück.

Alternativen

`for`, `while`

Siehe auch

`until`, `break`, `continue`

Modul

Foundation

return (: : :)

Beenden eines Prozeduraufrufs.

`return` beendet den aktuellen Prozeduraufruf und kehrt zur aufrufenden Prozedur zurück. Die Ausführung des Programms wird bei der nächsten aktiven Programmzeile nach dem Prozeduraufruf in der aufrufenden Prozedur fortgesetzt. Falls die aktuelle Prozedur die Hauptprozedur 'main' ist, so wird der Programmablauf beendet, und der Programmzähler springt an das Programmende. Es ist zu beachten, dass jede Prozedur außer der Hauptprozedur mindestens eine erreichbare `return` Operatorzeile besitzen muss, um von einem Aufruf der Prozedur zurückkehren zu können.

Ergebnis

`return` liefert immer TRUE zurück.

Modul

Foundation

stop (: : :)

Anhalten des Programmablaufes.

Der `stop`-Operator unterbricht den kontinuierlichen Programmablauf. Dabei bleibt der `PC` auf dem `stop`-Operator (und nicht der nächsten ausführbaren Zeile) stehen, damit der Grund für die Programmunterbrechung sofort zu erkennen ist, auch wenn hinter dem `stop`-Operator zahlreiche Kommentar- oder andere nicht ausführbare Zeilen folgen.

Der Operator ist äquivalent zum `Anhalten`-Befehl (F9) in der Menüleiste. Das Programm kann mit dem `Ausführen`-Befehl (F5) direkt fortgesetzt werden.

Über den Einstellungsdialog kann das Laufzeitverhalten durch Eingabe eines Zeitparameters umdefiniert werden, so dass die Programmausführung nicht mehr stoppt, sondern die eingestellte Zeit wartet. Innerhalb dieser Zeit kann das Programm mit F9 unterbrochen oder mit einem der Ausführungsbefehle fortgesetzt werden. Dies wird durch ein Icon in der ersten Spalte des Programmfensters gekennzeichnet.

Achtung

Dieser Operator wird beim Codeexport ignoriert.

Beispiel

```
read_image (Image, 'fabrik')
regiongrowing (Image, Regions, 3, 3, 6, 100)
count_obj (Regions, Number)
dev_update_window ('off')
for i := 1 to Number by 1
  select_obj (Regions, RegionSelected, i)
  dev_clear_window ()
  dev_display (RegionSelected)
  stop ()
endfor
```

Ergebnis

Hält das Programm an einem `stop`-Operator an, bleibt der Rückgabewert des vorherigen Operators erhalten. Wird das Programm danach fortgesetzt, liefert `stop` immer TRUE zurück.

Siehe auch

`exit`

Modul

Foundation

switch (: : ControlExpression :)

Einleiten einer Mehrfachverzweigung.

`switch` leitet einen Block mit einer Mehrfachverzweigung bzw. Fallunterscheidung ein. Der Parameter `ControlExpression` muss einen Integerwert ergeben. Der berechnete Wert bestimmt, zu welcher `case`-Marke innerhalb des `switch`..`endswitch`-Blocks gesprungen wird. Jede `case`-Anweisung enthält genau eine Integerkonstante. Stimmen der in `ControlExpression` berechnete Wert und die `case`-Konstante überein, wird die entsprechende Marke angesprungen. Optional kann innerhalb eines `switch`-Blocks als letzte Sprungmarke eine `default`-Anweisung angegeben werden. Diese wird angesprungen, wenn für den in `ControlExpression` berechneten Wert keine passende `case`-Anweisung gefunden wurde.

Wie in den Programmiersprachen C, C++ und C# stellt die `case`-Anweisung eine Sprungmarke dar und leitet anders als `elseif` nicht einen abgeschlossenen Block ein, der automatisch beim nächsten `case` verlassen wird.

Um nach den Anweisungen eines `case`-Zweiges den `switch`-Block zu verlassen, muss wie in C und C++ eine `break`-Anweisung eingebaut werden. Diese kann auch an beliebigen Stellen innerhalb eines `case`-Zweiges angegeben werden. Wird in der Programmausführung ein `break` erreicht, wird von dort direkt hinter die den `switch`-Block abschließende `endswitch`-Anweisung gesprungen.

Sollen für mehrere Integerwerte die gleichen Programmzeilen ausgeführt werden, können mehrere `case`-Anweisungen unmittelbar aufeinander folgen.

Parameter

▷ **ControlExpression** (input_control) ..... integer ⇝ *integer*
Integerausdruck, der die anzuspringende Sprungmarke bestimmt.

Beispiel

```
TestStr := ''
for Index := 1 to 8 by 1
```

```
  TestStr := TestStr + '<'
  switch (Index)
  case 1:
    TestStr := TestStr + '1'
    break
  case 2:
    TestStr := TestStr + '2'
    * intentionally fall through to 3
  case 3:
    TestStr := TestStr + '3'
    * intentionally fall through to 4
  case 4:
    TestStr := TestStr + '4'
    break
  case 5:
  case 6:
    * common case branch for 5 and 5
    TestStr := TestStr + '56'
    break
  case 7:
    * continue for loop
    TestStr := TestStr + '7'
    continue
  default:
    TestStr := TestStr + 'd'
    break
  endswitch
  TestStr := TestStr + '>'
endfor
```

Ergebnis

Enthält die Bedingung einen korrekten Ausdruck, dann liefert `switch` (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`if`, `elseif`, `else`

Siehe auch

`case`, `default`, `endswitch`, `if`

Modul

Foundation

throw (: : Exception :)

Werfen einer benutzerdefinierten oder Weitergeben einer mit `catch` abgefangenen Exception.

Die Operatoren `try`, `catch`, `endtry` und `throw` erlauben es, in HDevelop-Programme Code für eine dynamische Fehlerbehandlung ähnlich zu den Programmiersprachen C++ oder C# einzubauen. Grundlegende Informationen zur Fehlerbehandlung in HDevelop finden sich außerdem bei den Operatoren `try` und `dev_set_check` sowie im „`HDevelop User's Guide`".

Der Operator `throw` erlaubt es, an einer beliebigen Stelle im Programm eine Exception zu werfen. Diese kann dann mit dem `catch`-Operator eines umgebenen `try`-`catch`-Blocks gefangen werden. Dadurch ist es dem Entwickler möglich, eigene Fehler- oder Ausnahmezustände zu definieren, bei denen die normale Programmausführung – auch über Prozedurgrenzen hinweg – abgebrochen und in einen Fehlerbehandlungszweig gesprungen wird, beispielsweise um reservierte Ressourcen freizugeben.

Bei einer solchen anwenderdefinierten Exception kann ein nahezu beliebiges Tupel als `Exception`-Objekt geworfen werden, es sollte lediglich das erste Element des Tupels auf einen Fehlercode ≥ 30000 gesetzt werden. Können unterschiedliche anwenderdefinierte Ausnahmezustände auftreten, können diese über verschiedene Fehlercodes (≥ 30000) im ersten Element oder über zusätzliche Elemente unterschieden werden.

Darüber hinaus erlaubt der throw-Operator eine mit catch gefangene Exception weiter zu werfen. Das ist beispielsweise dann sinnvoll, wenn innerhalb eines inneren try-catch-endtry-Blocks, z.B. innerhalb einer Prozedur, die anderen Anwendungen bereitgestellt wird, nur auf bestimmte Fehler reagiert werden soll oder kann, und andere Fehler an den Aufrufer weitergereicht werden müssen.

In diesem Fall kann das mit catch gefangene Exception-Tupel einfach als Exception-Parameter übergeben und direkt weitergereicht werden. Darüber hinaus ist es aber auch möglich, eigene Elemente an das Exception-Tupel hinten anzuhängen. Diese können dann in dem äußeren catch-Block mit Hilfe des Operators dev_get_exception_data als *'user_data'* abgefragt werden:

```
try
  ...
catch (Exception)
  ...
  UserData := ...
  throw ([Exception, UserData])
endtry
```

Achtung

Der Export der Operatoren try, catch, endtry und throw ist nur in die Sprachen C++, C# und VisualBasic/.NET möglich, da nur sie das Werfen und Fangen von Exceptions über Prozedurgrenzen hinweg unterstützen. Beim Export nach C können diese Operatoren nicht unterstützt werden.

Parameter

▷ **Exception** (input_control) . . . . . exception-array ⇝ *integer* / string
Tupel mit den Fehlerdaten oder eigenen Fehlercodes.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert throw (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Siehe auch

try, catch, endtry, dev_get_exception_data, dev_set_check

Modul

Foundation

try (: : :)

Startet einen Block, in dem Exceptions erkannt und abgefangen werden können.

Die Operatoren try, catch, endtry und throw erlauben es, in HDevelop-Programme Code für eine dynamische Fehlerbehandlung ähnlich zu den Programmiersprachen C++ oder C# einzubauen. Zusammen bilden die ersten drei Operatoren zwei Codeblöcke: der erste (try .. catch) enthält den zu überwachenden Code und der zweite (catch .. endtry) den Code, der im Fehlerfall ausgeführt wird.

Der Operator try leitet den ersten Block ein, der die zu überwachenden Programmzeilen enthält. Tritt bei der Ausführung der nachfolgenden Operatoren oder Prozeduren ein Fehler oder anderer Ausnahmezustand (Exception) auf, oder wird vom Benutzer mit dem Operator throw explizit eine Exception geworfen, wird der try-Block sofort (bzw. nach Anzeigen eines Fehlerdialogs) beendet und die Programmausführung beim nächsten catch-Operator fortgesetzt. Tritt die Exception in einer aus dem try-Block direkt oder indirekt aufgerufenen Prozedur auf, wird diese Prozedur sofort (bzw. ebenfalls erst nach Anzeigen eines Fehlerdialogs) verlassen - einschließlich all der Prozeduren, die auf dem Callstack oberhalb der Prozedur, die den Fehlerbehandlungscode enthält, liegen.

Ob vor dem Werfen der Exception ein Dialog angezeigt wird oder nicht, steuert die HDevelop-Einstellung *'Fehlermeldungen unterdrücken (HDevelop Exceptions direkt werfen)'* im Einstellungsdialog unter Bearbeiten->Einstellungen->Allgemeine Optionen->Erfahrener Anwender. Dieser Fehlerdialog erlaubt es auch, die Programmausführung auf dem Operator, in dem der Fehler aufgetreten ist, d.h. noch vor dem Werfen der Exception anzuhalten, um beispielsweise einen möglicherweise fehlerhaften Operatoraufruf im Programm zu korrigieren.

Der überwachte Programmblock endet mit dem Operator catch. Tritt im try-Block keine Exception auf, wird der nachfolgende catch-Block übersprungen und die Ausführung nach dem Operator endtry fortgesetzt.

Es ist möglich try-catch-endtry-Blöcke beliebig ineinander zu schachteln - innerhalb einer Prozedur oder auch über mehrere Prozeduraufrufe hinweg. Wird dabei in einem inneren try-catch-Block eine Exception abgefangen, ist die Fehlerbehandlung mit dem Abarbeiten des entsprechenden catch-endtry-Blockes abgeschlossen, und die Exception ist für die äußeren try-catch-Blöcke nicht mehr sichtbar, es sei denn, die Exception wird aus dem catch-Block heraus mit dem throw-Operator explizit neu geworfen.

Tritt in einem HALCON-Operator ein Fehler auf, werden bestimmte Informationen über diesen Fehler, wie z.B. Fehlercode und Fehlertext in einem Exception-Tupel zusammengefasst und an den catch-Operator übergeben. Von dort können sie mit Hilfe des Operators dev_get_exception_data ausgelesen werden, um sie entsprechend auszuwerten oder auszugeben. Mehr Informationen zu den gespeicherten Fehlerdaten, den Zugriff darauf und Besonderheiten beim Export findet man bei diesen beiden Operatoren. Beim Operator throw ist beschrieben, was beim Werfen eines benutzerdefinierten Exception-Tupels zu beachten ist.

Wurde mit dem Operator dev_set_check('give_error') oder über den Dialog Bearbeiten->Einstellungen->Laufzeiteinstellungen, Checkbox: Fehler zurückgeben, die Fehlerbehandlung ausgeschaltet, wird bei einem Operator-Fehler keine Exception geworfen, sondern das Programm ganz normal mit dem nächsten Operator fortgesetzt. Im Gegensatz dazu wirft der throw-Operator immer eine Exception, unabhängig vom Zustand der *'give_error'*-Einstellung. Das gleiche gilt auch für Fehler, die bei der Auswertung von Parameterausdrücken auftreten.

Achtung

Der Export der Operatoren try, catch, endtry und throw ist nur in die Sprachen C++, C# und VisualBasic/.NET möglich, da nur sie das Werfen und Fangen von Exceptions über Prozedurgrenzen hinweg unterstützen. Beim Export nach C können diese Operatoren nicht unterstützt werden.

Beispiel

```
try
  read_image (Image, 'may_be_not_available')
catch (Exception)
  if (Exception[0] == 5200)
    dev_get_exception_data (Exception, 'error_message', ErrMsg)
    set_tposition (3600, 24, 12)
    write_string (3600, ErrMsg)
    return ()
  else
    * rethrow the exception
    throw ([Exception,'unknown exception in myproc'])
  endif
endtry
```

Ergebnis

try liefert immer TRUE zurück.

Alternativen

dev_set_check

Siehe auch

catch, endtry, throw, dev_get_exception_data, dev_set_check

Modul

Foundation

until (: : Condition :)

Führe so lange den Rumpf der Schleife aus, wie die Bedingung falsch ist.

until beendet eine repeat..until-Schleife. Die repeat..until-Schleife wird so lange durchlaufen, wie der Parameter Condition *'false'* (*0*) ergibt. Der Rumpf der Schleife wird mindestens einmal durchlaufen, da die Bedingung erst am Ende des Rumpfes überprüft wird.

Parameter

▷ **Condition** (input_control) . . . . . integer ⇝ *integer*
Bedingung für die Schleife.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `until` (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`for`, `while`

Siehe auch

`repeat`, `if`, `elseif`, `else`, `break`, `continue`

Modul

Foundation

while (: : Condition :)

Eröffnet einen Schleifenblock, der solange ausgeführt wird, wie die Bedingung wahr ist.

`while` führt den Schleifenblock bis zur zugehörigen Anweisung `endwhile` aus, solange der Parameter `Condition` *'true'* (oder eine Zahl ungleich *0*) ergibt.

Ergibt die Bedingung *'false'* (*0*), wird die Programmausführung nach der zugehörigen `endwhile`-Anweisung fortgesetzt.

Parameter

▷ **Condition** (input_control) . . . . . integer ⇝ *integer*
Bedingung für die Schleife.

Beispiel

```
dev_update_window ('off')
dev_close_window ()
dev_open_window (0, 0, 512, 512, 'black', WindowID)
read_image (Image, 'particle')
dev_display (Image)
stop ()
threshold (Image, Large, 110, 255)
dilation_circle (Large, LargeDilation, 7.5)
dev_display (Image)
dev_set_draw ('margin')
dev_set_line_width (3)
dev_set_color ('green')
dev_display (LargeDilation)
dev_set_draw ('fill')
stop ()
complement (LargeDilation, NotLarge)
reduce_domain (Image, NotLarge, ParticlesRed)
mean_image (ParticlesRed, Mean, 31, 31)
dyn_threshold (ParticlesRed, Mean, SmallRaw, 3, 'light')
opening_circle (SmallRaw, Small, 2.5)
connection (Small, SmallConnection)
dev_display (Image)
dev_set_colored (12)
dev_display (SmallConnection)
stop ()
dev_set_color ('green')
dev_display (Image)
dev_display (SmallConnection)
```

```
Button := 1
while (Button == 1)
  dev_set_color ('green')
  get_mbutton (WindowID, Row, Column, Button)
  dev_display (Image)
  dev_display (SmallConnection)
  dev_set_color ('red')
  select_region_point (SmallConnection, SmallSingle, Row, Column)
  dev_display (SmallSingle)
  NumSingle := |SmallSingle|
  if (NumSingle == 1)
    intensity (SmallSingle, Image, MeanGray, DeviationGray)
    area_center (SmallSingle, Area, Row, Column)
    dev_set_color ('yellow')
    set_tposition (WindowID, Row, Column)
    write_string (WindowID, 'Area='+Area+', Int='+MeanGray)
  endif
endwhile
dev_set_line_width (1)
dev_update_window ('on')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `while` (als Operator) den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt und ein Fehlercode zurückgeliefert.

Alternativen

`for`, `until`

Siehe auch

`repeat`, `break`, `continue`, `if`, `elseif`, `else`

Modul

Foundation

Kapitel 25

System

25.1 Betriebssystem

count_seconds (: : : Seconds)

Ermitteln von vergangener Zeit.

count_seconds dient der Zeitmessung. Jeder Aufruf des Operators liefert einen Zeitwert. Die Differenz der Werte zweier aufeinanderfolgender Aufrufe liefert die seither vergangene Zeit in Sekunden. Welche Art der Zeit gemessen wird, kann mit set_system('clock_mode',...) festgelegt werden.

Achtung

Die Zeitmessung ist nicht exakt und hängt von der Auslastung des Rechners ab.

Parameter

▷ **Seconds** (output_control) . . . . . real ⇝ *real*
Prozesszeit seit dem Programmstart.

Beispiel

```
count_seconds(Start)
* program segment to be measured
count_seconds(End)
Seconds := End - Start
```

Ergebnis

count_seconds liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

set_system

Modul

Foundation

get_system_time (: : : MSecond, Second, Minute, Hour, Day, YDay, Month, Year)

Auslesen der Systemzeit.

get_system_time liest die aktuelle Systemzeit aus.

Parameter

- ▷ **MSecond** (output_control) ..... integer ⇝ *integer*
  Millisekunden (0..999).
- ▷ **Second** (output_control) ..... integer ⇝ *integer*
  Sekunden (0..59).
- ▷ **Minute** (output_control) ..... integer ⇝ *integer*
  Minuten (0..59).
- ▷ **Hour** (output_control) ..... integer ⇝ *integer*
  Stunden (0..23).
- ▷ **Day** (output_control) ..... integer ⇝ *integer*
  Monatstag (1..31).
- ▷ **YDay** (output_control) ..... integer ⇝ *integer*
  Jahrestag (1..366).
- ▷ **Month** (output_control) ..... integer ⇝ *integer*
  Monat (1..12).
- ▷ **Year** (output_control) ..... integer ⇝ *integer*
  Jahr (xxxx).

Ergebnis

get_system_time liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

get_image_time

Modul

Foundation

system_call (: : Command :)

Ausführen eines Systemaufrufs.

system_call führt den in Command übergebenen Systemaufruf aus.

Unter Windows kann durch ein Voranstellen von *'start /b'* erreicht werden, dass nicht auf die gestartete Applikation gewartet wird. Unter unixartigen Systemen kann eine interaktive Shell ('csh -i') gestartet werden, indem ein leerer String in Command übergeben wird.

Parameter

- ▷ **Command** (input_control) ..... string ⇝ *string*
  Befehl für das Betriebssystem.
  Defaultwert: 'ls'

Ergebnis

Unter Windows liefert system_call immer den Wert TRUE, außer es gab ein Problem beim Erzeugen des neuen Prozesses. Unter anderen Betriebssystemen liefert system_call immer den Wert TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

count_seconds

Siehe auch

wait_seconds, count_seconds

Modul

Foundation

wait_seconds (: : Seconds :)

Verzögern der Programmausführung.

wait_seconds verzögert die Programmausführung um Seconds Sekunden.

wait_seconds verwendet den *'performance_counter'* für die Messung von Zeitintervallen. Der *'performance_counter'* ist im Rahmen der Beschreibung des Parameters *'clock_mode'* des Operators set_system erläutert.

Parameter

▷ **Seconds** (input_control) . . . . . real ⇝ *real*
Anzahl Sekunden, um die die Programmausführung ausgesetzt wird.
Defaultwert: 10
Restriktion: Seconds >= 0

Ergebnis

wait_seconds liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

system_call

Siehe auch

system_call, count_seconds

Modul

Foundation

25.2 Compute-Devices

activate_compute_device (: : DeviceHandle :)

Aktivieren eines Compute Devices.

activate_compute_device aktiviert das Compute Device DeviceHandle für den aktuellen HALCON Thread. Alle folgenden Aufrufe von HALCON Operatoren in diesem Thread werden auf DeviceHandle ausgeführt, sofern sie über eine entsprechende Implementierung (get_operator_info) verfügen. Mit deactivate_compute_device kann DeviceHandle wieder deaktiviert werden.

Gegenwärtig kann pro Thread nur ein Compute Device aktiv sein. Ist vor dem Aufruf von activate_compute_device bereits ein anderes Device aktiv, wird dieses automatisch deaktiviert.

Parameter

▷ **DeviceHandle** (input_control) . . . . . compute_device ⇝ *handle*
Compute Device Handle.

Beispiel

```
open_compute_device (DeviceIdentifier, DeviceHandle)
read_image (Image, 'rings_and_nuts')
*
* Gaussian convolution on a compute device
activate_compute_device (DeviceHandle)
derivate_gauss (Image, DerivGauss, 5, 'none')
*
* Gaussian convolution on the CPU
deactivate_compute_device (DeviceHandle)
derivate_gauss (Image, DerivGauss, 5, 'none')
```

Ergebnis

Ist `DeviceHandle` gültig, gibt `activate_compute_device` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_compute_device`

Siehe auch

`deactivate_compute_device`, `deactivate_all_compute_devices`

Modul

Foundation

deactivate_all_compute_devices (: : :)

Deaktivieren aller Compute devices.

`deactivate_all_compute_devices` deaktiviert alle Compute Devices für den aktuellen HALCON Thread.

Ergebnis

`deactivate_all_compute_devices` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`activate_compute_device`, `deactivate_compute_device`

Modul

Foundation

deactivate_compute_device (: : DeviceHandle :)

Deaktivieren eines Compute Devices.

deactivate_compute_device deaktiviert das Compute Device DeviceHandle für den aktuellen HALCON Thread. Das Compute Device wird mit activate_compute_device reaktiviert.

Parameter

▷ **DeviceHandle** (input_control) . . . . . compute_device ⇝ *handle*
Compute Device Handle.

Ergebnis

Ist DeviceHandle gültig, gibt deactivate_compute_device den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

activate_compute_device

Siehe auch

deactivate_all_compute_devices

Modul

Foundation

get_compute_device_info (: : DeviceIdentifier, InfoName : Info)

Abfragen von Informationen über ein Compute Device.

Der Operator get_compute_device_info liefert Informationen über ein Compute Device. Im Gegensatz zu get_compute_device_param werden nur statische Informationen abgefragt, so dass das Device weder geöffnet (open_compute_device) noch aktiviert (activate_compute_device) sein muss.

Folgende Informationen können abgefragt werden:

'vendor' Hersteller des Compute Devices.

'name' Name des Compute Devices.

'platform_version' Version der Compute Device Platform. Für OpenCL-Geräte ist dies die OpenCL Version.

'driver_version' Version des Treibers des Compute Devices.

'extensions' Unterstützte OpenCL Extensions.

'image_support' *'true'* wenn das Device Bild Objekte unterstützt.

'image2d_max_width' Maximale Breite von OpenCL Image Objekten.

'image2d_max_height' Maximale Höhe von OpenCL Image Objekten.

Parameter

▷ **DeviceIdentifier** (input_control) . . . . . integer ⇝ *integer*
Compute Device Handle.

▷ **InfoName** (input_control) . . . . . string ⇝ *string*
Name der abzufragenden Information.
Defaultwert: 'name'
Werteliste: InfoName ∈ {'driver_version', 'extensions', 'image_support', 'image2d_max_width', 'image2d_max_height', 'name', 'platform_version', 'vendor'}

▷ **Info** (output_control) . . . . . string(-array) ⇝ *string* / integer / real
Zurückgegebene Information.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_compute_device_info` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`query_available_compute_devices`

Nachfolger

`activate_compute_device`

Modul

Foundation

`get_compute_device_param` `( : : DeviceHandle, GenParamName : GenParamValue )`

Abfragen von Compute Device Parametern.

`get_compute_device_param` gibt die Werte von Parametern des Compute Devices `DeviceHandle` für den aktuellen HALCON Thread zurück in `GenParamValue` zurück.

Die folgenden Parameter können in `GenParamName` gesetzt und damit abgefragt werden (für genauere Beschreibungen siehe `set_compute_device_param`):

'alloc_pinned' Wenn *'alloc_pinned'* auf *'true'* gesetzt ist, werden Ausgabebildmatrizen in page locked Speicher erzeugt.

'asynchronous_execution' Wenn *'asynchronous_exection'* auf *'true'* gesetzt ist, warten Operatoren, die auf dem Compute Device ausgeführt werden, nicht auf Beendigung der Berechnungen sondern kehren nach dem Anstoßen der Berechnung zurück.

'buffer_cache_capacity' Maximale Größe (in Byte) des Compute Device Buffer Caches.

'buffer_cache_used' Aktuelle Größe (in Byte) des Compute Device Buffer Caches.

'image_cache_capacity' Maximale Größe (in Byte) des Compute Device Image Caches.

'image_cache_used' Aktuelle Größe (in Byte) des Compute Device Image Caches.

'pinned_mem_cache_capacity' Maximale Größe (in Bytes) des page locked Speicher Caches.

'pinned_mem_cache_used' Aktuelle Größe (in Bytes) des page locked Speicher Caches.

Parameter

▷ **`DeviceHandle`** (input_control) . . . . . compute_device ⇝ *handle*
Compute Device Handle.

▷ **`GenParamName`** (input_control) . . . . . string ⇝ *string*
Name des abzufragenden Parameters.
Defaultwert: 'buffer_cache_capacity'
Werteliste: `GenParamName` ∈ {'alloc_pinned', 'asynchronous_execution', 'buffer_cache_capacity', 'buffer_cache_used', 'image_cache_capacity', 'image_cache_used', 'pinned_mem_cache_capacity', 'pinned_mem_cache_used'}

▷ **`GenParamValue`** (output_control) . . . . . string(-array) ⇝ *string* / integer / real
Wert des Parameters.

Ergebnis

Sind alle Parameterwerte korrekt, gibt `get_compute_device_param` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`activate_compute_device`

Siehe auch

`set_compute_device_param`, `get_compute_device_info`

Modul

Foundation

init_compute_device (: : DeviceHandle, Operators :)

Initialisieren eines Compute Devices.

`init_compute_device` initialisiert ein Compute Device und bereitet eine Liste von Operatoren (`Operators`) auf die Ausführung auf dem Device vor. Außerdem wird das Device implizit für den aktuellen Thread aktiviert. Wird `init_compute_device` nicht aufgerufen oder werden andere Operatoren als die in `Operators` auf dem Compute Device verwendet, wird die Initialisierung beim ersten Aufruf des entsprechenden Operators durchgeführt.

`get_operator_info` gibt Auskunft darüber ob ein Operator auf dem Compute Device ausgeführt werden kann.

Achtung

Die Laufzeit von `init_compute_device` hängt von der Anzahl der Operatoren in `Operators` ab und kann mehrere Sekunden betragen.

Parameter

▷ **DeviceHandle** (input_control) ..... compute_device ⇝ *handle*
Compute Device Handle.

▷ **Operators** (input_control) ..... string-array ⇝ *string*
Liste der vorzubereitenden Operatoren
Defaultwert: 'all'
Werteliste: `Operators` ∈ {'all', 'derivate_gauss', 'sobel_amp'}

Ergebnis

Ist die Initialisierung erfolgreich, gibt `init_compute_device` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`query_available_compute_devices`, `open_compute_device`

Nachfolger

`activate_compute_device`

Modul

Foundation

open_compute_device (: : DeviceIdentifier : DeviceHandle)

Öffnen eines Compute Device.

open_compute_device öffnet das durch DeviceIdentifier referenzierte Compute Device und gibt das DeviceHandle zurück. Die Berechnung auf dem Device DeviceHandle wird mittels activate_compute_device aktiviert.

Das Device DeviceIdentifier muss mit dem Operator query_available_compute_devices ermittelt werden.

Parameter

▷ **DeviceIdentifier** (input_control) . . . . . integer ⇝ *integer*
Compute Device Identifier.

▷ **DeviceHandle** (output_control) . . . . . compute_device ⇝ *handle*
Compute Device Handle.

Ergebnis

Ist DeviceIdentifier gültig, liefert open_compute_device den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

query_available_compute_devices

Nachfolger

activate_compute_device, init_compute_device

Modul

Foundation

query_available_compute_devices (: : : DeviceIdentifier)

Abfragen der Liste verfügbarer Compute Devices.

Gibt die Liste aller verfügbaren Compute Devices zurück. Mittels get_compute_device_info können Eigenschaften eines Devices abgefragt werden. open_compute_device öffnet ein Device in HALCON.

Gegenwärtig unterstützt HALCON nur OpenCL-kompatible GPUs, welche die OpenCL-Erweiterung cl_khr_byte_addressable_store und Image Objekte unterstützen. Falls Sie sich nicht sicher sind, ob ein bestimmtes Device unterstützt wird, wenden Sie sich bitte an den Hersteller.

Momentan ist es nicht möglich OpenCL mittels Windows Remote Desktop zu verwenden, da dieser den Zugriff auf den Grafiktreiber unterbindet. Mittels einer VNC Lösung ist es aber möglich, OpenCL über eine remote Verbindung zu verwenden.

Es wird empfohlen die jeweils aktuellsten Grafiktreiber zu installieren. Um auf Linuxsystemen auf die Grafikkarte zugreifen zu können sollte der Benutzer Mitglied der Gruppe 'video' sein.

Parameter

▷ **DeviceIdentifier** (output_control) . . . . . integer-array ⇝ *integer*
Liste von verfügbaren Compute Devices.

Beispiel

```
query_available_compute_devices (DeviceIdentifiers)
for Index := 0 to |DeviceIdentifiers|-1 by 1
```

```
    get_compute_device_info (DeviceIdentifiers[Index], 'name', DeviceName)
    get_compute_device_info (DeviceIdentifiers[Index], 'vendor', DeviceVendor)
    if (DeviceVendor == 'NVIDIA Corporation' and \
        DeviceName == 'GeForce 8800 Ultra')
        open_compute_device (DeviceIdentifiers[Index], DeviceHandle)
        break
    endif
endfor
*
init_compute_device (DeviceHandle, 'derivate_gauss')
read_image (Image, 'rings_and_nuts')
*
* Gaussian convolution on a compute device
activate_compute_device (DeviceHandle)
derivate_gauss (Image, DerivGauss, 5, 'none')
*
* Gaussian convolution on the CPU
deactivate_compute_device (DeviceHandle)
derivate_gauss (Image, DerivGauss, 5, 'none')
```

Ergebnis

`query_available_compute_devices` gibt TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: mutually exclusive (läuft parallel zu anderen nicht-exklusiven Operatoren mit Ausnahme von sich selbst).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`get_compute_device_info`, `open_compute_device`

Modul

Foundation

release_all_compute_devices (: : :)

Schließen aller Compute Devices.

`release_all_compute_devices` darf nicht in HDevelop verwendet werden!

`release_all_compute_devices` schließt alle Compute Devices für den aktuellen HALCON Thread und gibt alle zugehörigen Ressourcen frei. Der Zweck von `release_all_compute_devices` ist einzig diese Ressourcen vor dem Entladen der HALCON Bibliothek freizugeben, wenn diese mit einem Mechanismus wie LoadLibrary oder dlopen geladen wurde. In allen anderen Fällen darf `release_all_compute_devices` nicht verwendet werden!

Ergebnis

Wurden alle Compute Devices erfolgreich geschlossen, gibt `release_all_compute_devices` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

release_compute_device, deactivate_compute_device, deactivate_all_compute_devices

Modul

Foundation

release_compute_device (: : DeviceHandle :)

Schließen eines Compute Devices.

release_compute_device darf nicht in HDevelop verwendet werden!

release_compute_device schließt ein Compute Device für den aktuellen HALCON Thread und gibt alle zu DeviceHandle gehörigen Ressourcen frei. Der Zweck von release_compute_device ist einzig diese Ressourcen vor dem Entladen der HALCON Bibliothek freizugeben, wenn diese mit einem Mechanismus wie LoadLibrary oder dlopen geladen wurde. In allen anderen Fällen darf release_compute_device nicht verwendet werden!

Parameter

▷ **DeviceHandle** (input_control) . . . . . compute_device ⇝ *handle*
Compute Device Handle.

Ergebnis

Wurde DeviceHandle erfolgreich geschlossen, gibt release_compute_device den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

release_all_compute_devices, deactivate_compute_device, deactivate_all_compute_devices

Modul

Foundation

set_compute_device_param (: : DeviceHandle, GenParamName, GenParamValue :)

Setzen von Parametern eines Compute Devices.

set_compute_device_param setzt die Parameter des Compute Devices DeviceHandle für den aktuellen HALCON Thread.

Die folgenden Parameter können gesetzt werden:

'alloc_pinned' Wenn *'alloc_pinned'* auf *'true'* gesetzt ist werden die Ausgabebilder aller folgenden Operatoraufrufe (egal ob auf dem Device ausgeführt oder nicht) dieses Threads in page locked (oder 'pinned') Speicher erzeugt. Das beschleunigt den Transfer zwischen dem Hauptspeicher und dem Speicher des Devices. Zu exzessive Verwendung von page locked Speicher kann aber negativen Einfluss auf die Geschwindigkeit des Systems nehmen. *'alloc_pinned'* sollte aktiviert werden, wenn das Ausgabebild des nächsten Operatoraufrufs auf das Device transferiert werden soll. Für Operatoraufrufe, deren Ausgabebilder nicht auf dem Device verwendet werden, sollte *'alloc_pinned'* deaktiviert werden (siehe Beispiel).

Werteliste: 'true', 'false'.

Standardwert: 'true'.

'asynchronous_execution' Ist 'asynchronous_execution' auf *'true'* gesetzt, warten Operatoren, die auf dem Device ausgeführt werden, nicht auf das Ende der Berechnungen, sondern enden, sobald die Berechnungen angestoßen wurden. Alle Berechnungen auf dem Device werden synchronisiert sobald die Ausgabe einer Device Berechnung auf der CPU benötigt wird (z.B. von `disp_image`).

Werteliste: 'true', 'false'.

Standardwert: 'true'.

'buffer_cache_capacity' Maximale Größe (in Byte) des Compute Device Buffer Caches.

Standardwert: 1/3 des verfügbaren Device Speichers

'image_cache_capacity' Maximale Größe (in Byte) des Compute Device Bild Caches.

Standardwert: 1/3 des verfügbaren Device Speichers

'pinned_mem_cache_capacity' Maximale Größe (in Byte) des page locked Speicher Caches.

Standardwert: 32000000 (32MByte)

Parameter

▷ **`DeviceHandle`** (input_control) . . . . . compute_device ⇝ *handle*
Compute Device Handle.

▷ **`GenParamName`** (input_control) . . . . . string ⇝ *string*
Name des zu setzenden Parameters.
Defaultwert: 'buffer_cache_capacity'
Werteliste: `GenParamName` ∈ {'alloc_pinned', 'asynchronous_execution', 'buffer_cache_capacity', 'image_cache_capacity', 'pinned_mem_cache_capacity'}

▷ **`GenParamValue`** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
Neuer Wert des Parameters.

Beispiel

```
activate_compute_device (DeviceHandle)
read_image (Image, 'fuse')
set_compute_device_param (DeviceHandle, 'alloc_pinned', 'true')
* filter on compute device, output image is page locked
derivate_gauss (Image, DerivGauss, 3, 'gradient')
* filter result on the CPU, output image should not be page locked
set_compute_device_param (DeviceHandle, 'alloc_pinned', 'false')
median_image (DerivGauss, ImageMedian, 'circle', 1, 'mirrored')
```

Ergebnis

Sind alle Parameterwerte korrekt, liefert `set_compute_device_param` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`activate_compute_device`

Siehe auch

`get_compute_device_param`

Modul

Foundation

25.3 Datenbank

count_relation (: : RelationName : NumOfTuples)

Ermitteln der Anzahl der Einträge in der HALCON-Datenbank.

count_relation liefert die Anzahl NumOfTuples der Einträge in der Relation RelationName der HALCON-Datenbank ab dem Zeitpunkt, an dem die Datenbank mit dem Parameter *'database'* von set_system *'database'* eingeschaltet wurde. Ist die Datenbank ausgeschaltet, ist NumOfTuples 0. Zusätzlich wird in diesem Fall eine Warnung ausgegeben, falls *'do_low_error'* über set_system gesetzt ist. Der Zustand der Datenbank kann über den Parameter *'database'* mit get_system erfragt werden.

Die HALCON-Datenbank ist aus fünf Tabellen ikonischer Datentypen aufgebaut, genannt Relationen. Die Einträge in diese Relationen werden allgemein Tupel genannt (nicht zu verwechseln mit einem HALCON-Tupel, was ein Objekt-Array bezeichnet). Ist die HALCON-Datenbank eingeschaltet, beinhaltet sie die Basisrelationen für Regionendaten, Bildmatrizen und XLDs sowie die Containerrelationen für HALCON-Objekte und HALCON-Tupel (Objekt-Array). Die HALCON-Objekte Region und Bild werden aus den Elementen der Regionendaten und Bildmatrix-Relationen aufgebaut: Eine Region besteht aus einem Zeiger auf ein Tupel in der Regionendatenrelation. Ein Bild besteht aus einem Zeiger auf ein Tupel in der Regionendatenrelation (wie eine Region) und zusätzlich aus einem oder mehreren Zeigern auf Tupel der Bildmatrixrelation. Mehrkanalbilder haben mehrere Zeiger auf Bildmatrix-Tupel.

Regionen und Bilder werden als Objekte bezeichnet. Man kann eine Region als den Spezialfall eines Bildes auffassen, das keine Bildmatrizen hat. Regionendaten und Bildmatrizen werden in HALCON aus Speicherplatzgründen üblicherweise von mehreren Objekten gemeinsam verwendet. So kann es beispielsweise mehr Bilder als Bildmatrizen geben. Relevant für den Speicherbedarf sind nur die drei Basisrelationen *'image'*, *'region'* und *'XLD'*. Die HALCON-Objekte (Regionen und Bilder) und HALCON-Tupel bestehen nur aus Verweisen auf Regionen- und Matrixdaten und benötigen deshalb nur wenige Byte Speicher.

Mögliche Werte für RelationName:

'image': Bildermatrizen. Dabei kann die selbe Matrix auch Bestandteil mehrerer Bilder sein (keine redundante Speicherung).

'region': Regionendaten. Dabei kann eine Region auch Bestandteil mehrerer Bildobjekte sein (keine redundante Speicherung).

'XLD': eXtended Line Description: Konturen, Polygone, Parallele, Linien etc. XLD Daten haben keine Grauwerte und sind subpixelgenau.

'object': HALCON-Bild-Objekte. Aufgebaut aus einer Region und optional einer oder mehrerer Bildmatrizen. Ein HALCON-Objekt bestehend aus einer Region wird Regionen-Objekt, mit zusätzlichen Bildmatrizen Bildobjekt bezeichnet.

'tuple': Im Kompaktmodus werden Tupel von Bildobjekten unter einem Surrogat in dieser Relation gespeichert. Statt mit den einzelnen Objektschlüsseln wird dann nur mehr mit diesem Schlüssel gearbeitet. Es hängt von der Wirtssprache ab, ob die Objekte einzeln (z.B. C++) oder als Tupel (z.B. C) übergeben werden.

Gewisse Datenbank-Objekte werden bereits von reset_obj_db angelegt und müssen daher immer vorhanden sein (die undefinierte Grauwertkomponente, die Objekte 'full' (FULL_REGION in HALCON/C) und 'empty' (EMPTY_REGION in HALCON/C) sowie die darin enthaltene leere und volle Region). reset_obj_db erscheint dementsprechend bei einem Aufruf von get_channel_info auch als 'creator' der vollen und leeren Region. Der Operator count_relation dient beispielsweise dazu, den die Vollständigkeit von clear_obj Operation zu überprüfen.

Achtung

Das Sammeln von Informationen für die Datenbank ist nicht thread-safe, wenn ikonisch Objekte zwischen den Threads ausgetauscht werden, d.h. wenn Objekte in einem anderen Thread zerstört werden als in dem sie erzeugt wurden.

Parameter

▷ **RelationName** (input_control) ..... string ⇝ *string*
Interessierende Relation der HALCON-Datenbank.
Defaultwert: 'object'
Werteliste: RelationName ∈ {'image', 'region', 'XLD', 'object', 'tuple'}

▷ **NumOfTuples** (output_control) . . . . . integer ⇝ *integer*
Anzahl der Tupel in der Relation.

Beispiel

```
* Close the graphics window in order to close the graphics stack,
* which would influence the measurement.
dev_close_window ()
* Enable the measurement.
set_system ('database','true')
*
count_relation ('image', I1)
count_relation ('region', R1)
count_relation ('XLD', X1)
count_relation ('object', O1)
count_relation ('tuple', T1)
*
* Result:
* I1 = 0
* R1 = 0
* X1 = 0
* O1 = 0
* T1 = 0
*
read_image (Patras, 'patras')
*
count_relation ('image', I2)
count_relation ('region', R2)
count_relation ('XLD', X2)
count_relation ('object', O2)
count_relation ('tuple', T2)
*
* I2 = 3  (three channels of the rgb image 'patras')
* R2 = 1  (the image domain of the image 'patras')
* X2 = 0  (no XLD data)
* O2 = 1  (the iconic object holding the channels and the domain)
* T2 = 0  (no empty object or object array )
```

Ergebnis

Ist der Parameter korrekt, dann liefert `count_relation` den Wert TRUE. Ansonsten wird eine Fehlerbehandlung ausgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`reset_obj_db`

Nachfolger

`set_system`

Siehe auch

`clear_obj`

Modul

Foundation

get_modules (: : : UsedModules, ModuleKey)

Abfragen der verwendeten Module und des Modulschlüssels.

get_modules gibt in UsedModules alle Module zurück, die bis zu diesem Zeitpunkt in der HALCON-Session genutzt wurden. Aus den verwendeten Modulen wird der Modulschlüssel erzeugt, der in ModuleKey übergeben wird. Jedem Operator ist genau ein Modul zugeordnet.

Parameter

▷ **UsedModules** (output_control) ..... string-array ⇝ *string*
Namen der verwendeten Module.

▷ **ModuleKey** (output_control) ..... integer ⇝ *integer*
Schlüssel für Lizenz-Manager.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

reset_obj_db (: : DefaultImageWidth, DefaultImageHeight, DefaultChannels :)

Initialisieren des HALCON-Systems.

reset_obj_db initialisiert das HALCON-System. Dabei legt der Operator die fünf Relationen (Grauwertdaten, Regionendaten, XLDs, Bildobjekte und ObjektTupel) an (siehe auch count_relation), die für die Bildverarbeitung mit HALCON nötig sind. Wurde die HALCON-Datenbank angeschaltet (set_system ('database','true')) und sind die Relationen bereits vorhanden, dann werden alle in den Relationen gelisteten Daten gelöscht!

Die DefaultImageWidth und DefaultImageHeight geben die Startwerte für das globale maximale Bildformat an. Wenn das erste erzeugte Objekt ein Bild ist werden die Werte jedoch nur dann mit der neuen Bildgröße überschrieben, falls diese die initialen Werte übersteigt. Wird statt einem Bild zuerst eine Region erzeugt, dann werden die Werte nur verändert, wenn das neue Bild größer als der Vorgabewert ist. Dies gilt aber nicht nur bei dem ersten Bild, das erzeugt, bzw. eingelesen wird: Das globale Bildformat wird immer vergrößert wenn größere Bilder erzeugt werden.

Das globale Bildformat hat Bedeutung beim Öffnen von Fenstern (open_window) und bei Speicherabschätzungen sowie Clipping von Regionen. Regionen werden, sofern der Clip-Modus eingeschaltet ist (set_system ('clip_region','true')) an dem globalen Bildformat geclippt. Dies kann dann zu Problemen führen, wenn Bilder unterschiedlicher Größe verwendet werden. In diesem Fall ist nur garantiert, dass eine Region kleiner oder gleich dem Bildformat der größten Bildes ist.

Der Parameter DefaultChannels gibt die häufigste Anzahl von Kanälen eines Bildobjektes an. Dieser Wert kann auch auf 0 gesetzt werden, wenn viele Regionen verwendet werden. Wenn mehr Kanäle für ein Bild nötig sind, als bei der Initialisierung angegeben wurde, dann wird für dieses Bild die Anzahl dynamisch erweitert. Wenn weniger Kanäle benötigt werden als bei der Initialisierung angegeben, dann werden die überzähligen Kanäle auf undefiniert gesetzt. Dies wirkt für den Anwender, als wären sie nicht vorhanden; es wird jedoch unnötiger Speicher allokiert.

Die Parameterwerte können mit get_system abgefragt werden.

Achtung

Wird reset_obj_db am Anfang einer HALCON-Sitzung nicht aufgerufen, dann wird HALCON automatisch mit reset_obj_db(128,128,0) initialisiert - mit entsprechendem Nebeneffekt beim Clipping von Regionen. Wurde die Datenbank mit Hilfe des Parameterwertes *'database'* des Operators set_system angeschaltet, werden durch den Aufruf von reset_obj_db alle ikonischen Objekte in der Datenbank gelöscht

Parameter

▷ **DefaultImageWidth** (input_control) . . . . . integer ⇝ *integer*
Default-Bildbreite (in Pixeln).
Defaultwert: 128
Wertevorschläge: `DefaultImageWidth` ∈ {64, 128, 320, 640, 800, 1280}

▷ **DefaultImageHeight** (input_control) . . . . . integer ⇝ *integer*
Default-Bildhöhe (in Pixeln).
Defaultwert: 128
Wertevorschläge: `DefaultImageHeight` ∈ {64, 128, 240, 480, 600, 1024}

▷ **DefaultChannels** (input_control) . . . . . integer ⇝ *integer*
Übliche Anzahl von Kanälen.
Defaultwert: 0
Wertevorschläge: `DefaultChannels` ∈ {0, 1, 2, 3, 4, 5, 6, 7}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `reset_obj_db` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`get_channel_info`, `count_relation`

Modul

Foundation

25.4 Fehlerbehandlung

get_check (: : : Check)

Abfragen des Status der HALCON-Kontrollmodi.

Mit `get_check` kann abgefragt werden, welche Überprüfungen gerade eingeschaltet und welche ausgeschaltet sind. `Check` liefert ein Tupel mit den Namen der Kontrollmodi (vgl. `set_check`), denen jeweils eine Tilde (~, z.B. '~data') vorangestellt ist, falls die betreffende Kontrolle ausgeschaltet ist.

Parameter

▷ **Check** (output_control) . . . . . string-array ⇝ *string*
Tupel der aktuellen Kontrollmodi.

Ergebnis

`get_check` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`set_check`

Siehe auch

`set_check`

Modul

Foundation

get_error_text (: : ErrorCode : ErrorMessage)

Abfragen der Fehlerbeschreibung zu einem HALCON-Fehlercode.

get_error_text gibt zu einem HALCON-Fehlercode den zugehörigen Fehlerbeschreibung aus. Es handelt sich hierbei um denselben Text, der bei einer Fehlerbehandlung ausgegeben wird. get_error_text ist z.B. dann nützlich, wenn die Fehlerbehandlung selbst programmiert wird (siehe: set_check(::'~give_error':)).

Achtung

Bei unbekanntem Fehlercode wird eine Standardmeldung ausgegeben.

Parameter

- ▷ **ErrorCode** (input_control) ..... integer ⇝ *integer*
  Fehlercode.
  Restriktion: 1 <= ErrorCode && ErrorCode <= 36000
- ▷ **ErrorMessage** (output_control) ..... string ⇝ *string*
  Zugehörige Fehlerbeschreibung.

Beispiel

```
Herror   err;
char       message[MAX_STRING];

set_check("~give_error");
err = send_region(region,socket);
set_check("give_error");
if (err != H_MSG_TRUE) {
get_error_text((Hlong)err,message);
  fprintf(stderr,"my error message: %s\n",message);
  exit(1);
}
```

Ergebnis

get_error_text liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Ausnahme ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_check

Siehe auch

set_check

Modul

none

get_extended_error_info (: : : OperatorName, ErrorCode, ErrorMessage)

Abfragen der erweiterten Fehlerinformation für den letzten aufgetretenen HALCON Fehler des aufrufenden Threads.

get_extended_error_info gibt eine erweiterte Beschreibung des letzten aufgetretenen HALCON Fehlers innerhalb des aufrufenden Threads zurück. OperatorName enthält den Operator Namen des HALCON

Operators, der diesen Fehlercode gesetzt hat. ErrorCode enthält einen erweiterten Fehlercode, sofern er vom HALCON Operator gesetzt wurde. Die Beschreibung des Fehlers wird in ErrorMessage zurückgegeben. Wenn keine erweiterte Fehlerbeschreibung von einem HALCON Operator gesetzt wurde, wird ein leerer Fehlercode zurückgegeben und die Parameter OperatorName und ErrorMessage enthalten leere Strings und *'ErrorCode'* den Wert 0. Die entsprechende Dokumentation eines Operators kann man entnehmen, ob und unter welchen Umständen der Operator die erweiterte Fehlerbeschreibung setzt.

get_extended_error_info ist thread-lokal, d.h. eine erweiterte Beschreibung des letzten HALCON Fehlers wird für jeden Thread separat zur Verfügung gestellt. Das Setzen eines erweiterten Fehlercodes in einem Thread beeinflusst somit nicht den erweiterten Fehlercode in einem anderen Thread. Das bedeutet auch, dass ein erweiterter Fehlercode nur in dem Thread ausgelesen werden kann, der auch den Operator aufgerufen hat, welcher den Fehler gesetzt hat. Eine thread-lokale Fehlerbeschreibung wird gelöscht (oder überschrieben), wenn innerhalb desselben Threads ein weiterer HALCON Fehler auftritt oder der Thread beendet wird.

Parameter

- ▷ **OperatorName** (output_control) ..... string ⇝ *string*
  Operator, in welchem der Fehler aufgetreten ist.
- ▷ **ErrorCode** (output_control) ..... integer ⇝ *integer*
  Erweiterter Fehlercode.
- ▷ **ErrorMessage** (output_control) ..... string ⇝ *string*
  Erweiterte Fehlerbeschreibung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

get_error_text

Modul

none

get_spy (: : Class : Value)

Aktuelle Konfiguration des HALCON Debugging-Tool.

get_spy liefert die aktuelle Konfiguration von spy, dem HALCON Debugging Tool. Die vorhandenen Überwachungsmodi (Wahlmöglichkeiten für Class) sowie die zugehörigen Einstellmöglichkeiten (mögliche Werte für Value) können mittels query_spy abgefragt werden. Eine detaillierte Beschreibung findet sich bei set_spy.

Parameter

- ▷ **Class** (input_control) ..... string ⇝ *string*
  Überwachungsmodus
  Defaultwert: 'mode'
  Werteliste: Class ∈ {'db', 'error', 'input_control', 'internal', 'log_file', 'mode', 'operator', 'output_control', 'parameter_values', 'time'}
- ▷ **Value** (output_control) ..... string ⇝ *string* / integer / real
  Status des Überwachungsmodus.

Ergebnis

get_spy liefert den Wert TRUE, falls der Parameter Class korrekt ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`reset_obj_db`

Siehe auch

`set_spy`, `query_spy`

Modul

Foundation

query_spy (: : : Classes, Values)

Einstellmöglichkeiten des HALCON Debugging-Tool abfragen.

`query_spy` liefert alle Einstellmöglichkeiten von spy, dem HALCON Debugging Tool, also die vorhandenen Überwachungsmodi (`Classes`) und die zugehörigen Einstellmöglichkeiten (`Values`). Eine detaillierte Beschreibung von spy findet sich bei `set_spy`.

Achtung

Die Werte von `Values` können nicht direkt als Eingabe für `set_spy` dienen, da sie als symbolische Konstante übergeben werden.

Parameter

▷ **Classes** (output_control) . . . . . string-array ⇝ *string*
Vorhandene Überwachungsmodi (siehe set_spy).

▷ **Values** (output_control) . . . . . string-array ⇝ *string*
Zugehöriger Status der Überwachungsmodi.

Ergebnis

query_spy liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`reset_obj_db`

Siehe auch

`set_spy`, `get_spy`

Modul

Foundation

set_check (: : Check :)

Ein- und Ausschalten von HALCON-Kontrollen.

Mit `set_check` kann man verschiedene automatische Kontrollen des HALCON-Systems ein- bzw. ausschalten. Ist ein bestimmter Kontrollmodus eingeschaltet, werden zur Laufzeit Überprüfungen von Parametern etc. vorgenommen. Wird dabei eine Inkonsistenz festgestellt, so wird der Programmlauf mit einer Fehlerbehandlung abgebrochen. Typischerweise sollte man bei der Programmentwicklung die Kontrollen eingeschaltet lassen und sie erst nach abgeschlossener Testphase ausschalten. Ist der Kontrollmodus ausgeschaltet und tritt währenddessen ein Fehler auf, kann es zu unvorhersehbarem Verhalten des Systems kommen. Das HALCON-System verfügt über verschiedene Kontrollmöglichkeiten, die jeweils getrennt gesetzt oder ausgeschaltet werden können. Das Einschalten erfolgt durch den Aufruf von `set_check` mit dem Namen (`Check`) der gewünschten Kontrolle; ausgeschaltet wird sie durch Angabe ihres Namens mit vorangestellter Tilde (~, z.B. '~data'). Mit Ausnahme des Kontrollmodus' *'memory'*, welche exclusiv für das ganze System gesetzt wird, werden alle anderen Kontrollmodi Thread spezifisch

gesetzt. Wird dem Namen dieser Modi der Prefix *'ref_'* vorangestellt, so wird nur der Referenzwert des jeweiligen Modus gesetzt, mit dem der Thread spezifische Modus eines vom Benutzer neu gestarteter Thread initialisiert wird.

Kontrollmodi:

'color': Ist diese Kontrolle eingeschaltet, dürfen nur Farben verwendet werden, die das Display für das aktuelle Fenster unterstützt. Anderenfalls wird eine Fehlermeldung ausgegeben.

Ist diese Kontrolle ausgeschaltet, wird für nicht existente Farben eine *Näherungsfarbe* verwendet (siehe: `set_color`, `set_gray`, `set_rgb`).

'text': Ist diese Kontrolle eingeschaltet, werden die Koordinaten beim Setzen des Textcursors sowie bei der Ausgabe von Zeichenreihen (`write_string`) dahingehend überprüft, ob ein Teil eines Zeichens außerhalb des Fensters zu liegen kommt (was vom System nicht grundsätzlich verboten wird).

Ist der Modus ausgeschaltet, dann wird der Text an den Fensterrändern geclippt.

'data': (Für Programmentwicklung) Überprüfung der Konsistenz von Bildobjekten (Regionen und Grauwertkomponenten).

'interface': Ist diese Kontrolle eingeschaltet, wird die Schnittstelle zwischen Wirtssprache und den HALCON-Prozeduren zur Laufzeit überwacht (z.B. Typisierung und Anzahl der Werte).

'database': Konsistenzprüfungen von Objekten (z.B. existiert ein Objekt, das gelöscht werden soll, überhaupt?)

'give_error': Legt fest, ob Fehler „Exceptions“ auslösen oder nicht. Ist diese Kontrolle ausgeschaltet, muss sich das Anwenderprogramm selbst um eine geeignete Fehlerbehandlung kümmern. Es sei insbesondere darauf hingewiesen, dass nicht gemeldete Fehler meist zu nicht besetzten Ausgabeparametern führen, die ein unvorhersehbares Verhalten des Programms verursachen können. Einzelheiten über die Fehlerbehandlung in den jeweiligen HALCON-Sprachschnittstellen können im HALCON Programmer's Guide bzw. im HDevelop User's Guide nachgelesen werden.

'father': Ist diese Kontrolle eingeschaltet, erlaubt HALCON als Vaterfenster eines neuen Fensters bei `open_window` nur die Nummer eines anderen HALCON-Fensters, andernfalls auch die IDs von Betriebssystem-Fenstern. Diese Kontrolle ist nur für die Fenstertypen 'WIN32-Window', 'X-Window' und 'Cocoa' relevant.

'region': (für Programmentwicklung) Überprüfung der Konsistenz von Sehnendaten (bedeutet mitunter starke Verlangsamung der Routinen).

'clear': Soll ein Tupel von Objekten mittels `clear_obj` gelöscht werden, wird normalerweise eine Exception ausgelöst, wenn einzelne Objekte gar nicht (mehr) existieren. Ist *'clear'* eingeschaltet, werden solche Objekte ignoriert.

'memory': (für Programmentwicklung) überprüft die von HALCON freigegebenen Speicher auf Konsistenz und Überschreiben der Speicherbereiche.

'all': Schaltet alle Kontrollmodi ein.

'none': Schaltet alle Kontrollmodi aus.

'default': Voreinstellung: ['give_error','database']

Parameter

▷ **`Check`** (input_control) ..... string(-array) ⇝ *string*
Gewünschter Kontrollmodus.
Defaultwert: 'default'
Werteliste: `Check` ∈ {'color', 'text', 'database', 'data', 'interface', 'give_error', 'father', 'region', 'clear', 'memory', 'all', 'none', 'default'}

Ergebnis

`set_check` liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

get_check, set_color, set_rgb, set_hsi, write_string

Modul

none

set_spy (: : Class, Value :)

Steuern des HALCON Debugging Tools.

set_spy ist das HALCON Debugging Tool. Es erlaubt die flexible Überwachung der Ein- und Ausgabedaten von HALCON-Operatoren - sowohl graphisch als auch textuell. Aktiviert wird die Datenkontrolle mittels

```
set_spy(::'mode','on':),
```

ausgeschaltet mittels

```
set_spy(::'mode','off':).
```

Außerdem kann das Debugging Tool mit der Environmentvariable HALCONSPY aktiviert werden: Wenn diese Variable definiert ist, entspricht das dem Aufruf mit *'mode'* und *'on'*.

Die folgenden Überwachungsmodi (Class) können über Value gesteuert werden:

Class = *'operator'*
Bei Aufruf einer Routine werden deren Name und die Namen ihrer Parameter (in TRIAS Notation) ausgegeben.
Value: *'on'* oder *'off'*.
Standardwert: 'off'.

Class = *'input_control'*
Bei Aufruf einer Routine werden die Namen und Werte der Eingabesteuerparameter ausgegeben.
Value: *'on'* oder *'off'*.
Standardwert: 'off'.

Class = *'output_control'*
Bei Aufruf einer Routine werden die Namen und Werte der Ausgabesteuerparameter ausgegeben.
Value: *'on'* oder *'off'*.
Standardwert: 'off'.

Class = *'db'*
Informationen über die 4 Relationen in der HALCON-Datenbank. Dies ist insbesondere bei der Suche nach vergessenen clear_obj zu suchen.
Value: *'on'* oder *'off'*.
Standardwert: 'off'.

Class = *'parameter_values'*
Ergänzung zu *'input_control'* und *'output_control'*: Gibt an, wie viele Werte pro Parameter maximal ausgegeben werden sollen (maximale Tupellänge für Ausgabe).
Value: Tupellänge (integer)
Standardwert: 4

Class = *'time'*
Rechenzeit für die Ausführung des Operators
Value: *'on'* oder *'off'*.
Standardwert: 'off'.

Class = *'log_file'*
Umleitung der Textausgabe von spy in ein File, das mit open_file geöffnet wurde.
Value: ein File Handle (siehe open_file).

Class = *'error'*
Ist *'error'* eingeschaltet, gibt spy beim Auftreten eines internen Fehlers die Prozeduren (Datei/Zeilennummer) aus, bei denen der Fehler auftritt.
Value: *'on'* oder *'off'*.
Standardwert: 'off'.

Class = *'internal'*

Ist *'internal'* eingeschaltet, gibt spy vor dem Aufruf eines internen HALCON-Operators den Prozedurname mit ihren Parametern (Datei/Zeilennummer) aus.

Value: *'on'* oder *'off'*.

Standardwert: 'off'.

Jede Ausgabe beginnt mit dem thread handle, gefolgt von einem globalen Zähler, welcher die Reihenfolge der Ausgaben markiert. In multi-threaded Anwendungen kann mit dieser Information die Ausgabe einzelnen User Threads zugeordnet und der chronologische Ablauf rekonstruiert werden.

Achtung

Unter Windows funktioniert die Ausgabe auf stdout nur in Konsolenapplikationen. Dies bedeutet, dass man diesen Mechanismus zum Beispiel nicht in Kombination mit HDevelop verwenden kann.

Parameter

▷ **Class** (input_control) . . . . . string ⇝ *string*
Überwachungsmodus
Defaultwert: 'mode'
Werteliste: Class ∈ {'db', 'error', 'input_control', 'internal', 'log_file', 'mode', 'operator', 'output_control', 'parameter_values', 'time'}

▷ **Value** (input_control) . . . . . string ⇝ *string* / integer / real
Einzustellender Status des Überwachungsmodus.
Defaultwert: 'on'
Wertevorschläge: Value ∈ {'on', 'off', 1, 2, 3, 4, 5, 10, 50, 0.0, 1.0, 2.0, 5.0, 10.0}

Beispiel

```
/* init spy: Setting of the wished control modi */
set_spy("mode","on");
set_spy("operator","on");
set_spy("input_control","on");
set_spy("output_control","on");
/* calling of program section, that will be examined */
set_spy("mode","off");
```

Ergebnis

set_spy liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

reset_obj_db

Siehe auch

get_spy, query_spy

Modul

Foundation

25.5 I/O-Geräte

close_io_channel (: : IOChannelHandle :)

Schließen von I/O-Übertragungskanälen.

Der Operator close_io_channel schließt mittels open_io_channel geöffnete Übertragungskanäle. Die Referenzen zu den Instanzen der Kanäle werden in IOChannelHandle übergeben. Es werden etwaiger Speicherplatz und reservierte Ressourcen des Gerätes freigegeben.

Parameter

▷ **IOChannelHandle** (input_control) . . . . . io_channel(-array) ⇝ *handle*
Referenzen der geöffneten I/O-Übertragungskanäle.

Ergebnis

Für gültige Handles liefert der Operator close_io_channel den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- IOChannelHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

open_io_channel

Nachfolger

close_io_device

Modul

Foundation

close_io_device (: : IODeviceHandle :)

Schließen des angegebenen I/O-Gerätes.

Der Operator close_io_device schließt die Instanz zu dem durch IODeviceHandle spezifizierten I/O-Gerät. Dabei werden alle am Gerät geöffneten Kanäle geschlossen (d.h., close_io_channel wird implizit für alle geöffneten Kanäle des Gerätes aufgerufen) und etwaiger Speicherplatz für Daten freigegeben.

Parameter

▷ **IODeviceHandle** (input_control) . . . . . io_device ⇝ *handle*
Referenz zum geöffneten I/O-Gerät.

Ergebnis

Für gültige Handles liefert der Operator close_io_device den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- IODeviceHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_io_device`

Modul

Foundation

control_io_channel (: : IOChannelHandle, ParamAction, ParamArgument : GenParamValue)

Ausführen einer Aktion auf I/O-Übertragungskanälen.

`control_io_channel` führt eine Aktion auf den angegebenen I/O-Übertragungskanälen aus. Eine ausführliche Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis `doc/html/reference/io`.

Parameter

- ▷ **IOChannelHandle** (input_control) ..... io_channel(-array) ⇝ *handle*
  Referenzen auf die geöffneten I/O-Übertragungskanäle.
- ▷ **ParamAction** (input_control) ..... string ⇝ *string*
  Name der auszuführenden Aktion.
- ▷ **ParamArgument** (input_control) ..... string-array ⇝ *string* / integer / real
  Argumente der auszuführenden Aktion.
  Defaultwert: []
- ▷ **GenParamValue** (output_control) ..... string-array ⇝ *string* / integer / real
  Ergebniswerte der ausgeführten Aktion.

Ergebnis

Im Falle gültiger Parameterwerte liefert der Operator `control_io_channel` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit `get_extended_error_info` abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_io_channel`

Modul

Foundation

control_io_device (: : IODeviceHandle, Action, Argument : Result)

Ausführen einer Aktion auf einer I/O-Geräteinstanz.

`control_io_device` führt eine Aktion auf einer I/O-Geräteinstanz aus. Eine ausführliche Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis *'doc/html/reference/io'*.

Parameter

- ▷ **IODeviceHandle** (input_control) ..... io_device ⇝ *handle*
  Referenz auf das geöffnete I/O-Gerät.
- ▷ **Action** (input_control) ..... string ⇝ *string*
  Name der auszuführenden Aktion.

▷ **Argument** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
Argumente der auszuführenden Aktion.
Defaultwert: []

▷ **Result** (output_control) . . . . . string-array ⇝ *string* / integer / real
Ergebniswerte der ausgeführten Aktion.

Ergebnis

Im Falle gültiger Parameterwerte liefert der Operator `control_io_device` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit `get_extended_error_info` abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_io_device`

Modul

Foundation

control_io_interface (: : IOInterfaceName, Action,
Argument : Result)

Ausführen einer Aktion auf der I/O-Schnittstelle.

`control_io_interface` führt eine Aktion auf der I/O-Schnittstelle aus, wozu noch kein I/O-Gerät mit `open_io_device` geöffnet worden sein muss. Die unterstützten Parameterwerte für die Aktion, welche in `Action` übergeben wird, und ihr optionales Argument in `Argument` hängen von der entsprechenden I/O-Schnittstelle ab. Die Beschreibung der gerätespezifischen Parameterwerte sind in der entsprechenden Dokumentation der Schnittstelle gelistet, die sich im Verzeichnis `doc/html/reference/io` findet.

Der besondere Wert *'keep_open'* für `Action` steuert, wann die I/O-Schnittstelle entladen wird. Wird `Argument` auf *'true'* gesetzt, wird die I/O-Schnittstelle erst dann entladen, wenn der HALCON-Prozess beendet wird. Ist er auf *'false'* gesetzt, wird die I/O-Schnittstelle entladen, sobald wie an dieser Schnittstelle keine Geräte mehr offen sind. Diese ist das Standardverhalten.

Achtung

Auf Windows Systemen können Fehler-Dialogboxen vom Betriebssystems erscheinen, wenn abhängende Module des Interfaces nicht gefunden werden können. Dies ist beispielsweise der Fall, wenn das entsprechende SDK nicht installiert wurde. Das Erscheinen der Dialogboxen kann über Windows Error Mode kontrolliert werden. Informationen dazu finden sich in der Beschreibung von SetErrorMode in der MSDN Dokumentation.

Parameter

▷ **IOInterfaceName** (input_control) . . . . . string ⇝ *string*
Name der HALCON I/O-Schnittstelle.
Defaultwert: []
Wertevorschläge: `IOInterfaceName` ∈ {'ADLINK-DAQPilot', 'ADLINK-EOS', 'Advantech', 'Contec', 'Hilscher-cifX', 'Interface', 'Linux-GPIO', 'NIDAQmx', 'OPC_Classic', 'OPC_UA'}

▷ **Action** (input_control) . . . . . string ⇝ *string*
Name der auszuführenden Aktion.

▷ **Argument** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
Argumente der auszuführenden Aktion.
Defaultwert: []

▷ **Result** (output_control) . . . . . string-array ⇝ *string* / integer / real
Ergebniswerte der ausgeführten Aktion.

Ergebnis

Der Operator `control_io_interface` liefert den Wert TRUE, wenn die angeforderte Aktion korrekt ausgeführt werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit `get_extended_error_info` abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`query_io_interface`

Modul

Foundation

`get_io_channel_param` (: : `IOChannelHandle`, `GenParamName` : `GenParamValue`)

Abfragen der Parameter von I/O-Übertragungskanälen.

`get_io_channel_param` liefert Parameterwerte zur Konfiguration bestimmter Übertragungskanäle. Die Abfrageparameter werden in `GenParamName` übergeben, die erfragten Konfigurationswerte in `GenParamValue` zurückgegeben. `IOChannelHandle` referenziert die Übertragungskanäle, die zuvor von `open_io_channel` erzeugt worden sein müssen.

Eine ausführliche Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis `doc/html/reference/io`.

Parameter

▷ **`IOChannelHandle`** (input_control) . . . . . io_channel(-array) ⇝ *handle*
Referenzen auf die geöffneten I/O-Übertragungskanäle.

▷ **`GenParamName`** (input_control) . . . . . string(-array) ⇝ *string*
Parameternamen.
Defaultwert: 'param_name'
Wertevorschläge: `GenParamName` ∈ {'io_channel_name', 'param_name'}

▷ **`GenParamValue`** (output_control) . . . . . string-array ⇝ *string* / integer / real / handle
Parameterwerte.

Ergebnis

Im Falle gültiger Parameterwerte liefert der Operator `get_io_channel_param` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit `get_extended_error_info` abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_io_channel`

Nachfolger

`read_io_channel`, `write_io_channel`

Siehe auch

`open_io_channel`, `query_io_device`, `set_io_channel_param`

Modul

Foundation

get_io_device_param (: : IODeviceHandle, GenParamName : GenParamValue)

Abfragen der Parameter einer I/O-Geräteinstanz.

get_io_device_param liefert Parameterwerte zur Konfiguration einer bestimmten Geräteinstanz. Die Abfrageparameter werden in GenParamName übergeben, die korrespondierenden Konfigurationswerte in GenParamValue zurückgegeben. IODeviceHandle adressiert die Geräteinstanz, die von open_io_device zurückgegeben wurde.

Eine ausführliche Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis doc/html/reference/io.

Parameter

- ▷ **IODeviceHandle** (input_control) ..... io_device ⇝ *handle*
  Referenz auf das geöffnete I/O-Gerät.
- ▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
  Parameternamen.
  Defaultwert: 'param_name'
  Wertevorschläge: GenParamName ∈ {'io_device_name', 'param_name'}
- ▷ **GenParamValue** (output_control) ..... attribute.value(-array) ⇝ *string* / integer / real
  Parameterwerte.

Ergebnis

Im Falle gültiger Parameterwerte liefert der Operator get_io_device_param den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_io_device, set_io_device_param

Nachfolger

open_io_channel

Siehe auch

open_io_device, query_io_interface, set_io_device_param

Modul

Foundation

open_io_channel (: : IODeviceHandle, IOChannelName, GenParamName, GenParamValue : IOChannelHandle)

Öffnen und Konfigurieren von I/O-Übertragungskanälen.

Der Operator open_io_channel öffnet und konfiguriert Übertragungskanäle einer geöffneten I/O-Geräteinstanz. Die Geräteinstanz muss zuvor von open_io_device erzeugt worden sein und wird dem Parameter IODeviceHandle übergeben. Die Übertragungskanäle selbst werden durch ihre Namen in IOChannelName adressiert. Die vom Gerät unterstützten Namen können mit Hilfe des Operators query_io_device über den Parameter *'io_channel_name'* abgefragt werden. Die Kanäle können über die Parameter GenParamName und GenParamValue konfiguriert werden. Es wird eine Instanz jedes Übertragungskanals erzeugt, deren Referenzen in IOChannelHandle zurückgegeben werden. Falls die Instanz eines Kanals nicht mehr benötigt wird, sollte sie über den Operator close_io_channel freigegeben und geschlossen werden. Sie wird automatisch geschlossen, wenn die mit dem Kanal zusammenhängende Geräteinstanz über close_io_device geschlossen wird.

Mit einem geöffneten Übertragungskanal können schließlich die Werte vom Gerät mit Hilfe der Operatoren `read_io_channel` und `write_io_channel` gelesen und geschrieben werden.

Ein bereits geöffneter Kanal kann über die Operatoren `set_io_channel_param` und `get_io_channel_param` rekonfiguriert werden.

Eine ausführliche Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis `doc/html/reference/io`.

Parameter

- ▷ **`IODeviceHandle`** (input_control) ..... io_device ⇝ *handle*
  Referenz auf das geöffnete I/O-Gerät.
- ▷ **`IOChannelName`** (input_control) ..... string(-array) ⇝ *string*
  I/O-Kanalnamen.
- ▷ **`GenParamName`** (input_control) ..... string-array ⇝ *string*
  Generische Parameternamen.
  Defaultwert: []
- ▷ **`GenParamValue`** (input_control) ..... string-array ⇝ *string* / integer / real
  Generische Parameterwerte.
  Defaultwert: []
- ▷ **`IOChannelHandle`** (output_control) ..... io_channel(-array) ⇝ *handle*
  Referenzen auf die geöffneten I/O-Übertragungskanäle.

Beispiel

```
query_io_device (IODeviceHandle, [], 'io_channel_names.digital_output', ChannelOutputNam
open_io_channel (IODeviceHandle, ChannelOutputNames[0], [], [], IOChannelHandle)
write_io_channel (IOChannelHandle, 1, Status)
```

Ergebnis

Der Operator `open_io_channel` liefert den Wert TRUE, wenn die Parameterwerte gültig waren und die Kanäle für das Gerät geöffnet werden konnten. Ansonsten wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit `get_extended_error_info` abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_io_device`, `query_io_device`

Nachfolger

`set_io_channel_param`, `read_io_channel`, `write_io_channel`

Modul

Foundation

`open_io_device` `( : : IOInterfaceName, IODeviceName, GenParamName, GenParamValue : IODeviceHandle )`

Öffnen und Konfigurieren eines I/O-Gerätes.

Der Operator `open_io_device` öffnet und konfiguriert die gewählte I/O-Schnittstelle für ein bestimmtes Gerät. Der Name der I/O-Schnittstelle ist durch den Basisnamen der darunterliegenden Bibliothek bestimmt und

wird dem Parameter IOInterfaceName übergeben. Bei der Konfiguration des ersten Gerätes für diese Schnittstelle wird automatisch die Bibliothek der I/O-Schnittstelle geladen. Das Gerät selbst wird durch den Namen adressiert, der dem Parameter IODeviceName übergeben wird. Die Namen der unterstützten Geräte können über den Operator query_io_interface abgefragt werden. Mit Hilfe der Parameter GenParamName und GenParamValue kann das Gerät konfiguriert werden. In IODeviceHandle wird schließlich eine Referenz zu der Geräteinstanz zurückgegeben. Falls diese Geräteinstanz nicht mehr benötigt wird, muss sie über den Operator close_io_device wieder freigegeben werden.

Mit einer geöffneten Instanz zu einem Gerät kann ein Übertragungskanal mit Hilfe des Operators open_io_channel geöffnet werden. Auf einem solchen Kanal können dann Werte über die Operatoren read_io_channel und write_io_channel gelesen und geschrieben werden.

Eine geöffnete Instanz eines Gerätes kann über die Operatoren set_io_device_param und get_io_device_param rekonfiguriert werden.

Die ausführliche Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis doc/html/reference/io.

Achtung

Auf Windows Systemen können Fehler-Dialogboxen vom Betriebssystems erscheinen, wenn abhängende Module des Interfaces nicht gefunden werden können. Dies ist beispielsweise der Fall, wenn das entsprechende SDK nicht installiert wurde. Das Erscheinen der Dialogboxen kann über Windows Error Mode kontrolliert werden. Informationen dazu finden sich in der Beschreibung von SetErrorMode in der MSDN Dokumentation.

Parameter

- ▷ **IOInterfaceName** (input_control) ..... string ⇝ *string*
  Name der HALCON I/O-Schnittstelle.
  Defaultwert: []
  Wertevorschläge: IOInterfaceName ∈ {'ADLINK-DAQPilot', 'ADLINK-EOS', 'Advantech', 'Contec', 'Hilscher-cifX', 'Interface', 'Linux-GPIO', 'NIDAQmx', 'OPC_Classic', 'OPC_UA'}
- ▷ **IODeviceName** (input_control) ..... tuple ⇝ *string*
  I/O-Gerätename.
  Defaultwert: []
- ▷ **GenParamName** (input_control) ..... string-array ⇝ *string*
  Generische Parameternamen.
  Defaultwert: []
- ▷ **GenParamValue** (input_control) ..... string-array ⇝ *string* / integer / real
  Generische Parameterwerte.
  Defaultwert: []
- ▷ **IODeviceHandle** (output_control) ..... io_device ⇝ *handle*
  Referenz auf das geöffnete I/O-Gerät.

Beispiel

```
* Select a suitable i/o device interface of name IOInterfaceName
query_io_interface (IOInterfaceName, 'io_device_names', DeviceNames)
open_io_device (IOInterfaceName, DeviceNames[0], [], [], IODeviceHandle)
query_io_device (IODeviceHandle, [], 'io_channel_names.digital_input', ChannelInputNames
open_io_channel (IODeviceHandle, ChannelInputNames[0], [], [], IOChannelHandle)
read_io_channel (IOChannelHandle, Value, Status)
```

Ergebnis

Der Operator open_io_device liefert den Wert TRUE, wenn eine Instanz für das Gerät geöffnet werden konnte. Ansonsten wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

query_io_interface

Nachfolger

set_io_device_param, get_io_device_param, open_io_channel

Modul

Foundation

query_io_device (: : IODeviceHandle, IOChannelName, Query : Result)

Abfragen von Informationen über Kanäle eines I/O-Gerätes.

Der Operator query_io_device liefert Informationen zu den Übertragungskanälen, welche von einem spezifischen Gerät unterstützt werden. Die Referenz zu einem I/O-Gerät wird in IODeviceHandle übergeben und die Kanäle über ihre Namen in IOChannelName adressiert. Die gewünschte Information über die Kanäle wird mit Query spezifiziert. Eine Liste aller unterstützten Kanalnamen erhält man, wenn Query der Parametername *'io_channel_name'* übergeben wird. Der Parameter IOChannelName wird dabei ignoriert.

Die Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis doc/html/reference/io.

Parameter

- ▷ **IODeviceHandle** (input_control) ..... io_device ⇝ *handle*
  Referenz auf das geöffnete I/O-Gerät.
- ▷ **IOChannelName** (input_control) ..... string(-array) ⇝ *string*
  I/O-Kanalnamen.
- ▷ **Query** (input_control) ..... string-array ⇝ *string*
  Abzufragende Information.
  Defaultwert: 'param_name'
  Werteliste: Query ∈ {'io_channel_names', 'param_name'}
- ▷ **Result** (output_control) ..... string-array ⇝ *string* / integer / real
  Ergebniswerte (gemäß Query).

Beispiel

```
* Select a suitable i/o device interface of name IOInterfaceName
query_io_interface (IOInterfaceName, 'io_device_names', DeviceNames)
open_io_device (IOInterfaceName, DeviceNames[0], [], [], IODeviceHandle)
query_io_device (IODeviceHandle, [], 'io_channel_names.digital_input', ChannelInputNames
open_io_channel (IODeviceHandle, ChannelInputNames[0], [], [], IOChannelHandle)
read_io_channel (IOChannelHandle, Value, Status)
```

Ergebnis

Sind die Parameterwerte korrekt und ist die angegebene I/O-Schnittstelle verfügbar, liefert query_io_interface den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

open_io_device

Modul

Foundation

query_io_interface (: : IOInterfaceName, Query : Result)

Abfragen von Informationen zu einer I/O-Geräteschnittstelle.

Der Operator query_io_interface liefert Informationen zur angegebenen I/O-Geräteschnittstelle IOInterfaceName. Die gewünschte Art der Information wird über den Parameter Query ausgewählt. In Result wird die angeforderte Information zurückgegeben.

Mit dem Parameterwert *'interface_name'* für Query können alle verfügbaren HALCON I/O-Schnittstellen abgefragt werden. In diesem Fall wird der Wert für IOInterfaceName ignoriert.

Die Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis doc/html/reference/io.

Achtung

Auf Windows Systemen können Fehler-Dialogboxen vom Betriebssystems erscheinen, wenn abhängende Module des Interfaces nicht gefunden werden können. Dies ist beispielsweise der Fall, wenn das entsprechende SDK nicht installiert wurde. Das Erscheinen der Dialogboxen kann über Windows Error Mode kontrolliert werden. Informationen dazu finden sich in der Beschreibung von SetErrorMode in der MSDN Dokumentation.

Parameter

- ▷ **IOInterfaceName** (input_control) ..... string ⇝ *string*
  Name der HALCON I/O-Schnittstelle.
  Defaultwert: []
  Wertevorschläge: IOInterfaceName ∈ {'ADLINK-DAQPilot', 'ADLINK-EOS', 'Advantech', 'Contec', 'Hilscher-cifX', 'Interface', 'Linux-GPIO', 'NIDAQmx', 'OPC_Classic', 'OPC_UA'}
- ▷ **Query** (input_control) ..... string(-array) ⇝ *string*
  Parametername der Anfrage.
  Defaultwert: 'io_device_names'
  Wertevorschläge: Query ∈ {'interface_name', 'io_device_names', 'param_name', 'revision'}
- ▷ **Result** (output_control) ..... string-array ⇝ *string* / integer / real
  Liste der Ergebniswerte (abhängig von Query).

Beispiel

```
* Select a suitable i/o device interface of name IOInterfaceName
query_io_interface (IOInterfaceName, 'io_device_names', DeviceNames)
open_io_device (IOInterfaceName, DeviceNames[0], [], [], IODeviceHandle)
query_io_device (IODeviceHandle, [], 'io_channel_names.digital_input', ChannelInputNames
open_io_channel (IODeviceHandle, ChannelInputNames[0], [], [], IOChannelHandle)
read_io_channel (IOChannelHandle, Value, Status)
```

Ergebnis

Sind die Parameterwerte korrekt und ist die angegebene I/O-Schnittstelle verfügbar, liefert query_io_interface den Wert TRUE. Ansonsten wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`open_io_device`

Modul

Foundation

read_io_channel (: : IOChannelHandle : Value, Status)

Lesen von Daten auf I/O-Übertragungskanälen.

`read_io_channel` liest Eingabedaten von den angegebenen Übertragungskanälen. Die Kanäle werden dem Parameter `IOChannelHandle` übergeben, die Daten in `Value` zurückgegeben. Der Parameter `Status` gibt für jeden gelesenen Wert in `Value` eine Statusmeldung zurück. Der Wert *0* zeigt an, dass das korrespondierende Datum in `Value` auf dem Kanal korrekt gelesen werden konnte. Die Bedeutung anderer Statuswerte hängt vom jeweiligen Interface ab. Ihre Beschreibung findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis `doc/html/reference/io`.

Parameter

- ▷ **IOChannelHandle** (input_control) . . . . . io_channel(-array) ⇝ *handle*
  Referenzen auf die geöffneten I/O-Übertragungskanäle.
- ▷ **Value** (output_control) . . . . . tuple-array ⇝ *integer* / real / string
  Gelesene Daten.
- ▷ **Status** (output_control) . . . . . integer-array ⇝ *integer*
  Statusrückmeldung für gelesene Daten.

Ergebnis

Im Falle gültiger Parameterwerte liefert der Operator `read_io_channel` den Wert TRUE, unabhängig vom Erfolg des Lesens. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit `get_extended_error_info` abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_io_channel`, `set_io_channel_param`

Nachfolger

`close_io_channel`

Siehe auch

`write_io_channel`, `query_io_device`, `set_io_channel_param`

Modul

Foundation

set_io_channel_param (: : IOChannelHandle, GenParamName, GenParamValue :)

Konfigurieren von I/O-Übertragungskanälen.

`set_io_channel_param` (re)konfiguriert die angegebenen Übertragungskanäle. Die Konfigurationsparameter werden in `GenParamName`, die korrespondierenden Werte in `GenParamValue` übergeben. `IOChannelHandle` referenziert die Übertragungskanäle, die mit `open_io_channel` geöffnet werden können.

Eine ausführliche Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis `doc/html/reference/io`.

Parameter

- ▷ **IOChannelHandle** (input_control) . . . . . io_channel(-array) ⇝ *handle*
  Referenzen auf die I/O-Übertragungskanäle.
- ▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
  Parameternamen.
  Defaultwert: []
- ▷ **GenParamValue** (input_control) . . . . . string-array ⇝ *string* / integer / real / handle
  Zu setzende Parameterwerte.
  Defaultwert: []

Ergebnis

Im Falle gültiger Parameterwerte liefert der Operator `get_io_device_param` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit `get_extended_error_info` abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_io_channel`

Nachfolger

`read_io_channel`, `write_io_channel`

Siehe auch

`open_io_channel`, `query_io_device`, `get_io_channel_param`, `read_io_channel`, `write_io_channel`

Modul

Foundation

set_io_device_param (: : IODeviceHandle, GenParamName, GenParamValue :)

Konfigurieren einer bestimmten I/O-Geräteinstanz.

`set_io_device_param` (re)konfiguriert eine bestimmte Geräteinstanz. Die Konfigurationsparameter werden in `GenParamName` übergeben, die korrespondierenden Konfigurationswerte in `GenParamValue`. `IODeviceHandle` adressiert die Geräteinstanz, die von `open_io_device` zurückgegeben wurde.

Eine ausführliche Beschreibung aller gerätespezifischen Parameter findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis `doc/html/reference/io`.

Parameter

- ▷ **IODeviceHandle** (input_control) . . . . . io_device ⇝ *handle*
  Referenz auf das geöffnete I/O-Gerät.
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Parameternamen.
  Defaultwert: []
- ▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
  Zu setzende Parameterwerte.
  Defaultwert: []

Ergebnis

Im Falle gültiger Parameterwerte liefert der Operator `get_io_device_param` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit `get_extended_error_info` abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_io_device

Nachfolger

open_io_channel

Siehe auch

open_io_device, query_io_interface, get_io_device_param

Modul

Foundation

write_io_channel (: : IOChannelHandle, Value : Status)

Schreiben von Daten auf I/O-Übertragungskanälen.

write_io_channel schreibt Daten auf die angegebenen Übertragungskanäle. Die Referenzen zu den Instanzen der Übertragungskanäle werden dem Parameter IOChannelHandle, die Daten dem Parameter Value übergeben. Der Parameter Status gibt für jedes übergebene Datum in Value eine Statusmeldung zurück. Der Wert *0* zeigt an, dass das korrespondierende Datum in Value auf dem Kanal korrekt geschrieben werden konnte. Die Bedeutung anderer Statuswerte hängt vom jeweiligen Interface ab. Ihre Beschreibung findet sich in der Dokumentation für die verwendete I/O-Schnittstelle im Verzeichnis doc/html/reference/io.

Parameter

- ▷ **IOChannelHandle** (input_control) . . . . . io_channel(-array) ⇝ *handle*
  Referenzen auf die geöffneten I/O-Übertragungskanäle.
- ▷ **Value** (input_control) . . . . . tuple-array ⇝ *integer* / real / string
  Zu schreibende Daten.
- ▷ **Status** (output_control) . . . . . integer-array ⇝ *integer*
  Statusrückmeldung für geschriebene Daten.

Ergebnis

Im Falle gültiger Parameterwerte liefert der Operator write_io_channel den Wert TRUE, unabhängig vom Erfolg des Schreibens. Andernfalls wird eine Fehlerbehandlung durchgeführt. In diesem Fall kann vom Interface eine zusätzliche Fehlerinformation gesetzt werden, die mit get_extended_error_info abgefragt werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_io_channel, set_io_channel_param

Nachfolger

close_io_channel

Siehe auch

read_io_channel, open_io_channel, query_io_device, set_io_channel_param

Modul

Foundation

25.6 Information

get_chapter_info (: : Chapter : Info)

Informationen zu den Operatorkapiteln.

get_chapter_info liefert Informationen zu den Operatorkapiteln. Wird für Chapter der leere String übergeben, liefert die Routine in Info die Namen aller Kapitel zurück. Spezifiziert man hingegen ein bestimmtes Kapitel bzw. Kapitel + Unterkapitel (durch ein Tupel von Namen), werden die zugehörigen Unterkapitel oder - falls es keine weiteren Unterkapitel mehr gibt - die Namen der zugehörigen Operatoren ausgegeben. Die Einteilung der Operatorkapitel entspricht dabei der Kapitel-/ Unterkapiteleinteilung im HALCON-Manual. Hinweis: Die Operatorkapitel bzw. -unterkapitel eines konkreten Operators kann mittels get_operator_info(::<Name>,'chapter',Info:) abgefragt werden. Die Online-Texte werden den Dateien operators_[LANG].ref, operators_[LANG].sta, operators_[LANG].key, operators_[LANG].num und operators_[LANG].idx entnommen, die HALCON im aktuellen Verzeichnis und im Verzeichnis 'help_dir' (siehe get_system und set_system) sucht.

Achtung

Das Encoding der Ergebnisse ist UTF-8.

Parameter

- ▷ **Chapter** (input_control) ..... string(-array) ⇝ *string*
  Interessierende Operatorklasse bzw. -unterklasse.
  Defaultwert: ''
- ▷ **Info** (output_control) ..... string-array ⇝ *string*
  Operatorklassen (Chapter = '') oder Operatorunterklassen bzw. Operatoren.

Ergebnis

Sind die Parameterwerte korrekt und ist die Hilfedateien verfügbar, dann liefert get_chapter_info den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_system, set_system

Siehe auch

get_operator_info, get_system, set_system

Modul

Foundation

get_keywords (: : OperatorName : Keywords)

Schlüsselwörter, die Operatoren zugeordnet sind, abfragen.

get_keywords liefert alle Schlüsselwörter in den Online-Texten zu den Operatoren, die den vorgegebenen Teilstring OperatorName in ihrem Namen haben. Wird für OperatorName der leere String übergeben, gibt get_keywords sämtliche Schlüsselwörter aus. Die Schlüsselwörter eines einzigen Operators können auch mit get_operator_info abgefragt werden. Die Online-Texte werden den Dateien operators_[LANG].ref, operators_[LANG].sta, operators_[LANG].key, operators_[LANG].num und operators_[LANG].idx entnommen, die HALCON im aktuellen Verzeichnis und im Verzeichnis 'help_dir' (siehe get_system und set_system) sucht.

Achtung

Das Encoding der Ergebnisse ist UTF-8.

Parameter

▷ **OperatorName** (input_control) ..... proc_name ⇝ *string*
Teilstring in den Namen der Operatoren, zu denen Schlüsselwörter benötigt werden.
Defaultwert: 'get_keywords'

▷ **Keywords** (output_control) ..... string-array ⇝ *string*
Schlüsselwörter zu den Operatoren.

Ergebnis

`get_keywords` liefert den Wert TRUE, wenn die Parameter korrekt sind und die Hilfedateien zur Verfügung stehen. Sonst wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_chapter_info`

Alternativen

`get_operator_info`

Siehe auch

`get_operator_name`, `search_operator`, `get_param_info`

Modul

Foundation

get_operator_info (: : OperatorName, Slot : Information)

Informationen zu einem HALCON-Operator.

`get_operator_info` dient zur Abfrage der Online-Texte zu einem vorgegebenen Operator (siehe auch `get_operator_name`). Die Art der Information (`Slot`), die für jeden Operator zur Verfügung stehen, lassen sich mittels `query_operator_info` abfragen. Derzeit werden folgende Slots angeboten:

'short': Kurzbeschreibung des Operators.

'abstract': Beschreibung des Operators.

'chapter': Kapitelname(n) in der Operatorhierarchie (Kapitel, Unterkapitel im HALCON-Manual).

'functionality': Funktionalität, ist gleich der Objektklasse, der der Operator zugeordnet werden kann.

'keywords': Schlüsselwörter zu dem Operator (optional).

'example': Beispiel für die Anwendung des Operators (optional). Mit 'example.SPRACHE' (SPRACHE ∈ {trias,c,c++,c#,vb.net}) können Beispiele für eine bestimmte Sprache abgefragt werden, insofern sie vorhanden sind. Wird die Sprache nicht mit angegeben oder ist in dieser Sprache kein Beispiel vorhandenen, wird das TRIAS-Beispiel ausgegeben.

'complexity': Komplexität des Operators (optional).

'effect': Momentan nicht genutzt.

'parallelization': Charakteristik des Operators bezüglich parallelen Verhaltens und Timeouts.

'execution information': Charakteristik des Operators bezüglich parallelen Verhaltens und Timeouts.

'parallel_method': Methoden der von einem Operator unterstützten automatischen Parallelisierung.

'interrupt_mode': Modi der Unterbrechbarkeit, die von einem spezifischen Operator unterstützt werden. In der Dokumentation von `set_operator_timeout` und `interrupt_operator` ist beschrieben, wie diese Modi genutzt und gesetzt werden können.

'alternatives': Alternativen zum Operator (optional).

'see_also': Operatoren mit weitergehenden Informationen (optional).

'predecessor': Mögliche und sinnvolle Vorgängerprozedur

'successor': Mögliche und sinnvolle Nachfolgerprozedur

'result_state': Rückgabewert des Operators (TRUE, FALSE, FAIL, VOID oder EXCEPTION).

'attention': Einschränkungen und Hinweise zur korrekten Anwendung des Operators (optional).

'parameter': Namen der Parameter des Operators (siehe `get_param_info`).

'references': Literaturverweise (optional).

'module': Das Modul, zu dem der Operator gehört.

'html_path': Das Verzeichnis, in dem die HTML-Dokumentation des Operators steht.

'warning': Eventuelle Warnungen zur Verwendung des Operators.

'compute_device': Liste von Compute Devices, die vom Operator unterstützt werden.

Die Texte werden den Dateien operators_[LANG].ref, operators_[LANG].sta, operators_[LANG].key, operators_[LANG].num und operators_[LANG].idx entnommen, die HALCON im aktuellen Verzeichnis und im Verzeichnis 'help_dir' (siehe `get_system` und `set_system`) sucht. Bei Slots mit textuellen Informationen kann mit '.latex' nach dem Slotnamen der Text in LaTeX Notation geholt werden.

Achtung

Das Encoding der Ergebnisse ist UTF-8.

Parameter

- ▷ **`OperatorName`** (input_control) ..... proc_name ⇝ *string*
  Name des Operators, zu dem Information benötigt werden.
  Defaultwert: 'get_operator_info'
- ▷ **`Slot`** (input_control) ..... string ⇝ *string*
  Gewünschte Information.
  Defaultwert: 'abstract'
  Werteliste: `Slot` ∈ {'short', 'abstract', 'chapter', 'chapter_id', 'functionality', 'parallelization', 'execution information', 'parallel_method', 'interrupt_mode', 'compute_device', 'complexity', 'predecessor', 'successor', 'alternatives', 'parameter', 'see_also', 'keywords', 'example', 'attention', 'result_state', 'references', 'module', 'html_path', 'warning'}
- ▷ **`Information`** (output_control) ..... string(-array) ⇝ *string*
  Information (leer, falls keine Information vorhanden ist)

Ergebnis

`get_operator_info` liefert den Wert TRUE, wenn die Parameter korrekt sind und die Hilfsdateien verfügbar sind. Sonst wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_keywords`, `search_operator`, `get_operator_name`, `query_operator_info`, `query_param_info`, `get_param_info`

Nachfolger

`get_param_names`, `get_param_num`, `get_param_types`

Alternativen

`get_param_names`

Siehe auch

`query_operator_info`, `get_param_info`, `get_operator_name`, `get_param_num`, `get_param_types`

Modul

Foundation

get_operator_name (: : Pattern : OperatorNames)

Operatoren, die eine Zeichenreihe als Teilstring im Namen haben.

get_operator_name nimmt eine Zeichenreihe (Pattern) als Eingabe und sucht alle HALCON-Operatoren, die diese Zeichenreihe als Teilstring im Namen enthalten. Bei Eingabe des leeren Strings werden die Namen aller verfügbaren Operatoren ausgegeben.

Parameter

▷ **Pattern** (input_control) . . . . . string ⇝ *string*
Teilstring des gesuchten Namens (leer <=> alle Namen).
Defaultwert: 'info'

▷ **OperatorNames** (output_control) . . . . . string-array ⇝ *string*
Gefundene Operatornamen.

Ergebnis

get_operator_name liefert den Wert TRUE falls die Hilfedateien verfügbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

get_operator_info, get_param_names, get_param_num, get_param_types

Alternativen

search_operator

Siehe auch

get_operator_info, get_param_names, get_param_num, get_param_types

Modul

Foundation

get_param_info (: : OperatorName, ParamName, Slot : Information)

Informationen zu Operatorparametern.

get_param_info dient zur Abfrage der Online-Texte zu einem Parameter eines vorgegebenen Operators. Die Art der Information (Slot), die für jeden Parameter zur Verfügung steht, lässt sich mittels query_param_info abfragen. Mögliche Slots sind unten gelistet. Für eine detailliertere Beschreibung der Einträge wird auf das Manual „Extension Package Programmer's Manual" verwiesen.

'description': Beschreibung des Parameters.

'description.latex': Beschreibung des Parameters in LaTeX Notation.

'parameter_class': Parameterklasse.

'input_object'
'output_object'
'input_control'
'output_control'

'type_list': Zulässige(r) Datentyp(en) für Parameterwerte.

'default_type': Default-Typ für Parameterwerte (nur bei Steuerparametern). Dies ist der Parametertyp der bei HALCON/C im „simple Mode" verwendet wird. Falls 'none' angegeben wird, dann muss der „Tupel Modus" verwendet werden.

'sem_type': Semantischer Typ des Parameters. Das ist wichtig für die Zuordnung der Parameter zu Objektklassen in objektorientierten Sprachen (C++, .NET, COM). Gehören mehrere Parameter semantisch zu einem Typ, so wird dies dabei ebenfalls bezeichnet.

'default_value': Default-Wert für den Parameter (nur bei Eingabe-Steuerparametern). Hierbei handelt es sich um eine reine Information (der Parameterwert muss in jedem Fall explizit übergeben werden, auch wenn der default-Wert verwendet werden soll). Dieser Eintrag dient nur als Hinweis, als Ausgangspunkt für eigene Experimente. Die Werte sind so gewählt, dass sie normalerweise keine Fehler verursachen und etwas sinnvolles bewirken.

'modified':
`'true'`, falls der Operator den Eingabeparameter modifiziert.

'multi_value':
`'true'`, falls ein Tupel an Werten übergeben werden muss,
`'false'`, falls genau ein Wert übergeben werden muss, und
`'optional'`, wenn beides erlaubt ist.

'multichannel':
`'true'`, falls ein Eingabebildobjekt mehrkanalig sein darf.

'multiinstance':
`'true'`, falls ein Array semantischer Tupel (z.B. Posen, Kameraparameter, etc.) übergeben werden muss,
`'false'`, falls genau ein semantisches Tupel übergeben werden muss, und
`'optional'`, wenn beides erlaubt ist.

'mixed_type': Nur bei Steuerparametern und auch nur dann, wenn Wertetupel ('multivalue'-`'true'`/`'optional'`) und verschiedene Datentypen für Parameterwerte ('type_list' mehr als ein Wert) zulässig sind. In diesem Fall gibt `Slot` an, ob in einem Tupel auch Werte verschiedener Typen gemischt auftreten dürfen (`'true'` oder `'false'`).

'values': Werteauswahl (optional).

'value_list': Falls ein Parameter nur eine begrenzte Anzahl von Werten annehmen kann, werden diese explizit aufgeführt (optional).

'valuemin': Minimalwert eines Werteintervalls.

'valuemax': Maximalwert eines Werteintervalls.

'valuefunction': Funktion des Werteverlaufs einer Testreihe (lin, log, quadr, ...).

'steprec': Empfohlene Schrittweite der Parameterwerte in einer Testreihe.

'stepmin': Minimale Schrittweite der Parameterwerte in einer Testreihe.

'valuenumber': Aussage über die Anzahl der Parameter absolut oder relativ zu anderen Parametern

'assertion': Aussage über die Parameterwerte absolut oder relativ zu anderen Parametern.

'cd_type_list.[compute_device]': Liste der Eingabebild-Typen, die von der angegebenen Compute-Device-Implementierung des Operators unterstützt werden (z.B. 'cd_type_list.opencl' für OpenCL-Geräte).

'cd_value_list.[compute_device]': Liste der Eingabe-Steuerparameter, die von der angegebenen Compute-Device-Implementierung des Operators unterstützt werden (z.B. 'cd_value_list.opencl' für OpenCL-Geräte).

Die Online-Texte werden den Dateien operators_[LANG].ref, operators_[LANG].sta, operators_[LANG].key, operators_[LANG].num und operators_[LANG].idx entnommen, die HALCON im aktuellen Verzeichnis und im Verzeichnis 'help_dir' (siehe `get_system` und `set_system`) sucht.

Achtung

Das Encoding der Ergebnisse ist UTF-8.

Parameter

▷ **`OperatorName`** (input_control) ..... proc_name ⇝ *string*
Name des Operators, zu dessen Parametern Information benötigt wird.
Defaultwert: 'get_param_info'

▷ **`ParamName`** (input_control) ..... string ⇝ *string*
Name des Parameters, zu dem Information benötigt wird.
Defaultwert: 'Slot'

▷ **Slot** (input_control) ..... string ⇝ *string*
Gewünschte Information.
Defaultwert: 'description'
Werteliste: Slot ∈ {'assertion', 'default_type', 'default_value', 'description', 'mixed_type', 'modified', 'multichannel', 'multivalue', 'sem_type', 'step_min', 'step_rec', 'type_list', 'value_function', 'value_list', 'value_max', 'value_min', 'value_number', 'values', 'cd_type_list.opencl', 'cd_value_list.opencl'}

▷ **Information** (output_control) ..... string(-array) ⇝ *string*
Information (leer, falls keine Informationen vorhanden sind).

Ergebnis

get_param_info liefert den Wert TRUE, wenn die Parameter korrekt sind und die Hilfsdateien verfügbar sind. Sonst wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_keywords, search_operator

Alternativen

get_param_names, get_param_num, get_param_types

Siehe auch

query_param_info, get_operator_info, get_operator_name

Modul

Foundation

get_param_names (: : OperatorName : InpObjPar, OutpObjPar, InpCtrlPar, OutpCtrlPar)

Zugriff auf die Parameternamen eines HALCON-Operators.

get_param_names liefert für den in OperatorName angegebenen HALCON-Operator die Namen der Eingabeobjekte, der Ausgabeobjekte, der Eingabesteuerparameter sowie der Ausgabesteuerparameter.

Parameter

▷ **OperatorName** (input_control) ..... proc_name ⇝ *string*
Name des Operators.
Defaultwert: 'get_param_names'

▷ **InpObjPar** (output_control) ..... string-array ⇝ *string*
Namen der Eingabeobjekte.

▷ **OutpObjPar** (output_control) ..... string-array ⇝ *string*
Namen der Ausgabeobjekte.

▷ **InpCtrlPar** (output_control) ..... string-array ⇝ *string*
Namen der Eingabesteuerparameter.

▷ **OutpCtrlPar** (output_control) ..... string-array ⇝ *string*
Namen der Ausgabesteuerparameter.

Ergebnis

get_param_names liefert den Wert TRUE, falls der Operatorname existiert und die Hilfedateien verfügbar sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_keywords, search_operator, get_operator_name, get_operator_info

Nachfolger

get_param_num, get_param_types

Alternativen

get_operator_info, get_param_info

Siehe auch

get_param_num, get_param_types, get_operator_name

Modul

Foundation

get_param_num (: : OperatorName : CName, InpObjPar, OutpObjPar, InpCtrlPar, OutpCtrlPar, Type)

Anzahl der Parameter eines HALCON-Operators.

get_param_num bestimmt für den angegebenen HALCON-Operator die Anzahl der Ein- und Ausgabeobjektparameter sowie der Ein- und Ausgabesteuerparameter. Außerdem erhält man den Namen der C-Funktion (CName), die von dem Operator aufgerufen wird. Der Ausgabeparameter Type gibt an, ob es sich um einen Operator oder eine Benutzer-Prozedur handelt.

Parameter

- ▷ **OperatorName** (input_control) ..... proc_name ⇝ *string*
  Name des Operators.
  Defaultwert: 'get_param_num'
- ▷ **CName** (output_control) ..... string ⇝ *string*
  Name der aufgerufenen C-Funktion.
- ▷ **InpObjPar** (output_control) ..... integer ⇝ *integer*
  Anzahl der Eingabeobjektparameter.
- ▷ **OutpObjPar** (output_control) ..... integer ⇝ *integer*
  Anzahl der Ausgabeobjektparameter.
- ▷ **InpCtrlPar** (output_control) ..... integer ⇝ *integer*
  Anzahl der Eingabesteuerparameter.
- ▷ **OutpCtrlPar** (output_control) ..... integer ⇝ *integer*
  Anzahl der Ausgabesteuerparameter.
- ▷ **Type** (output_control) ..... string ⇝ *string*
  Systemprozedur oder Benutzerprozedur.
  Wertevorschläge: Type ∈ {'system', 'user'}

Ergebnis

get_param_num liefert den Wert TRUE, falls der Operatorname existiert. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_keywords, search_operator, get_operator_name, get_operator_info

Nachfolger

get_param_types

Alternativen

get_operator_info, get_param_info

Siehe auch

get_param_names, get_param_types, get_operator_name

Modul

Foundation

get_param_types (: : OperatorName : InpCtrlParType, OutpCtrlParType)

Default-Datentyp für die Steuerparameter eines HALCON-Operators.

get_param_types gibt für jeden Eingabe- und jeden Ausgabesteuerparameter dessen Default-Datentyp an. Ein Default-Typ eines Parameters ist der Typ, der in HALCON/C in „simple Mode“ verwendet wird. Dies ist für Parameter relevant, die mehr als einen Typ zulassen wie z.B. write_string. Dabei werden die Typen der Eingabeparameter in der Variablen InpCtrlParType zusammengefasst, die Typen der Ausgabeparameter in der Variablen OutpCtrlParType. Mögliche Werte der Typen sind:

'integer': Eine ganze Zahl.

'integer tuple': Eine ganze Zahl oder Tupel ganzer Zahlen.

'real': Eine Gleitpunktzahl.

'real tuple': Eine Gleitpunktzahl oder Tupel von Gleitpunktzahlen.

'string': Eine Zeichenreihe.

'string tuple': Eine Zeichenreihe oder Tupel von Zeichenreihen.

'no_default': Einzelner Wert, dessen Typ nicht festgelegt werden kann.

'no_default tuple': Einzelner Wert oder Tupel von Werten, deren Typ nicht festgelegt werden kann.

'default': Einzelner Wert mit unbekanntem Typ, System nimmt 'integer' an.

Parameter

- ▷ **OperatorName** (input_control) ..... proc_name ⇝ *string*
  Name des Operators.
  Defaultwert: 'get_param_types'
- ▷ **InpCtrlParType** (output_control) ..... string-array ⇝ *string*
  Default-Typen der Eingabesteuerparameter.
- ▷ **OutpCtrlParType** (output_control) ..... string-array ⇝ *string*
  Default-Typen der Ausgabesteuerparameter.

Ergebnis

get_param_types liefert den Wert TRUE, falls der angegebene Operatorname existiert. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_keywords, search_operator, get_operator_name, get_operator_info

Alternativen

get_param_info

Siehe auch

get_param_names, get_param_num, get_operator_info, get_operator_name

Modul

Foundation

query_operator_info (: : : Slots)

Slots der Informationen für `get_operator_info`*.*

`query_operator_info` liefert die Namen der Online-Texte (`Slots`), die für jeden Operator online zur Verfügung stehen. Die Informationen selbst können mit

```
get_operator_info(::<OperatorName>,<Slot>:<Information>)
```

abgefragt werden.

Parameter

▷ **Slots** (output_control) . . . . . string-array ⇝ *string*
Namen der Slots von `get_operator_info`.

Ergebnis

`query_operator_info` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`get_operator_info`

Siehe auch

`get_operator_info`

Modul

Foundation

query_param_info (: : : Slots)

Slots der Online-Informationen für `get_param_info`*.*

`query_param_info` liefert die Namen der Informationen (`Slots`), die für jeden Parameter eines Operators online zur Verfügung stehen (Online-Texte). Die Online-Texte selbst können mit

```
get_param_info(::<Procedurname>,<Parametername>,<Slot>:<Information>)
```

abgefragt werden.

Parameter

▷ **Slots** (output_control) . . . . . string-array ⇝ *string*
Namen der Slots bei `get_param_info`.

Ergebnis

`query_param_info` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`get_param_info`

Siehe auch

`get_param_info`

Modul

Foundation

search_operator (: : Keyword : OperatorNames)

Namen aller Operatoren zu einem Schlüsselwort.

search_operator liefert die Namen aller Operatoren, deren Online-Texte das Schlüsselwort Keyword enthalten (siehe get_operator_info). Alle verfügbaren Schlüsselwörter liefert der Aufruf

get_keywords(::": <Schlüsselwörter>).

Die Online-Texte werden den Dateien operators_[LANG].ref, operators_[LANG].sta, operators_[LANG].key, operators_[LANG].num und operators_[LANG].idx entnommen, die HALCON im aktuellen Verzeichnis und im Verzeichnis 'help_dir' (siehe get_system und set_system) sucht.

Parameter

▷ **Keyword** (input_control) . . . . . string ⇝ *string*
Schlüsselwort, zu dem Operatoren gesucht werden sollen.
Defaultwert: 'Information'

▷ **OperatorNames** (output_control) . . . . . string-array ⇝ *string*
Operatoren, deren Slot 'keyword' das Schlüsselwort enthalten.

Ergebnis

search_operator liefert den Wert TRUE, wenn die Parameter korrekt sind und die Hilfsdateien verfügbar sind. Sonst wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_keywords

Siehe auch

get_keywords, get_operator_info, get_param_info

Modul

Foundation

25.7 Multithreading

broadcast_condition (: : ConditionHandle :)

Signalisieren eines Condition-Synchronisationsobjektes.

broadcast_condition weckt alle Threads auf, die auf die Condition ConditionHandle warten. broadcast_condition ist wirkungslos, falls keine Threads warten.

Parameter

▷ **ConditionHandle** (input_control) . . . . . condition ⇝ *handle*
Condition-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert broadcast_condition den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_condition, wait_condition

Siehe auch

wait_condition, timed_wait_condition

Modul

Foundation

clear_barrier (: : BarrierHandle :)

Zerstören eines Barrieren-Synchronisationsobjektes.

clear_barrier zerstört das Barriere-Objekt, welches in BarrierHandle übergeben wird. Es dürfen keine Threads an der Barriere warten. Nach seiner Zerstörung kann die Barriere nicht weiterverwendet werden und das Handle wird ungültig.

Parameter

▷ **BarrierHandle** (input_control) . . . . . barrier ⇝ *handle*
Barriere-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert clear_barrier den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BarrierHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_barrier

Siehe auch

create_barrier

Modul

Foundation

clear_condition (: : ConditionHandle :)

Zerstören eines Condition-Synchronisationsobjektes.

clear_condition zerstört die Condition-Variable, welche in ConditionHandle übergeben wird. Es dürfen keine Threads an der Condition warten. Nach seiner Zerstörung kann die Condition nicht weiterverwendet werden und das Handle wird ungültig.

Parameter

▷ **ConditionHandle** (input_control) . . . . . condition ⇝ *handle*
Condition-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert clear_condition den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ConditionHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_condition`

Siehe auch

`create_condition`

Modul

Foundation

`clear_event` (: : EventHandle :)

Löschen eines Event-Synchronisationsobjektes.

`clear_event` zerstört das Eventobjekt, welches in `EventHandle` übergeben wird. Das Event muss frei sein und darf keinem Thread gehören. Nach seiner Zerstörung kann das Event nicht weiterverwendet werden und das Handle wird ungültig.

Parameter

▷ **`EventHandle`** (input_control) . . . . . event ⇝ *handle*
Event-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert `clear_event` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- EventHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_event`

Siehe auch

`create_event`

Modul

Foundation

clear_message (: : MessageHandle :)

Zerstören eins Nachrichtenobjektes und Freigeben aller Ressourcen.

clear_message schließt die Nachrichtenobjekte MessageHandle, welche zuvor mit Hilfe des Operators create_message oder dequeue_message erzeugt wurden. Alle von den Nachrichten gehaltenen Ressourcen werden freigegeben, insbesondere die der gespeicherten Tupeldaten und ikonischen Objekte.

Achtung

Nachdem die Nachrichten durch clear_message zerstört wurden dürfen sie nicht mehr verwendet werden. Die Verwendung von ungültigen Nachrichtenobjekte führt zu undefiniertem Verhalten.

Parameter

▷ **MessageHandle** (input_control) . . . . . message(-array) ⇝ *handle*
Zu zerstörende Nachrichtenobjekte.
Parameteranzahl: MessageHandle >= 1
Restriktion: MessageHandle != 0

Beispiel

```
MessageHandles := []
for idx := 0 to 4 by 1
  create_message (MessageHandle)
  MessageHandles[idx]:= MessageHandle
endfor
* ...
```

Ergebnis

clear_message gibt TRUE zurück, wenn die übergebenen Nachrichtenobjekte gültig sind. Andernfalls wird eine Fehlerbehandlung durchgeführt. Falls ein Tupel von Nachrichtenobjekten übergeben wird welches ungültige Referenzen enthält, wird clear_message versuchen so viele Nachrichtenobjekte wie möglich frei zu geben, bevor ein Fehler zurückgegeben wird.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MessageHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_message, dequeue_message

Siehe auch

create_message, set_message_tuple, get_message_tuple, set_message_obj, get_message_obj, set_message_param, get_message_param, enqueue_message, dequeue_message

Modul

Foundation

clear_message_queue (: : QueueHandle :)

Schließen einer Warteschlange für Nachrichten.

clear_message_queue schließt ein Warteschlangenobjekt QueueHandle, dass zuvor durch den Operator create_message_queue erzeugt wurde. Es werden alle Ressourcen freigegeben, die von der Warteschlange gehalten werden, einschließlich der eingereihten Nachrichtenobjekte und der in ihnen enthaltenen Datensätze.

Achtung

Nachdem die Warteschlangenobjekte durch clear_message_queue zerstört wurden dürfen sie nicht mehr verwendet werden. Die Verwendung von ungültigen Referenzen führt zu undefiniertem Verhalten. Der Operator clear_message_queue darf nicht auf ein Warteschlangenobjekt aufgerufen werden, dass gleichzeitig noch in einem anderen Thread in Verwendung ist. Dies führt zu undefiniertem Verhalten.

Parameter

▷ **QueueHandle** (input_control) . . . . . message_queue(-array) ⇝ *handle*
Freizugebene Warteschlangenobjekt(e).
Parameteranzahl: QueueHandle >= 1
Restriktion: QueueHandle != 0

Beispiel

```
create_message_queue (ProducerQueue)
create_message_queue (ResultQueue)
* ...
```

Ergebnis

clear_message_queue gibt den Wert TRUE zurück, wenn die übergebenen Warteschlangenobjekte gültig sind. Andernfalls wird eine Fehlerbehandlung durchgeführt. Sollte eine der übergebenen Referenzen ungültig sein, wird clear_message_queue dennoch versuchen alle anderen Referenzen freizugeben bevor es den Fehler zurück gibt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- QueueHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_message_queue, set_message_queue_param

Siehe auch

create_message_queue, enqueue_message, dequeue_message, set_message_queue_param, get_message_queue_param, create_message, clear_message, set_message_tuple, get_message_tuple, set_message_obj, get_message_obj

Modul

Foundation

clear_mutex (: : MutexHandle :)

Löschen eines Mutex-Synchronisationsobjektes.

clear_mutex zerstört das Mutexobjekt, welches in MutexHandle übergeben wird. Der Mutex muss frei sein und darf keinem Thread gehören. Nach seiner Zerstörung kann der Mutex nicht weiterverwendet werden und das Handle wird ungültig.

Parameter

▷ **MutexHandle** (input_control) . . . . . mutex ⇝ *handle*
Mutex-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert `clear_mutex` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MutexHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_mutex`

Siehe auch

`create_mutex`

Modul

Foundation

create_barrier (: : AttribName, AttribValue,
TeamSize : BarrierHandle)

Erzeugen eines Barriere-Synchronisationsobjektes.

Eine Barriere ist ein Synchronisationsobjekt, welches einen Thread solange blockiert, bis eine zuvor definierte Anzahl von Threads diese Barriere erreicht haben.

`create_barrier` legt das Barriere-Objekt an, welches in `BarrierHandle` zurückgegeben wird, und initialisiert es entsprechend der Attribute in `AttribName` und `AttribValue`. `AttribName` definiert die Klasse und `AttribValue` den Typ der Barriere. Im Folgenden werden die möglichen Attributwerte für die gelisteten Attributnamen beschrieben:

'' der leere String setzt die Standardattribute.

'type' spezifiziert das Verhalten wenn ein Thread an einer Barriere warten muss:

'sleep' unterbricht den aufrufenden Thread und legt ihn schlafen (Standard).

'spin' ist die schnelle Implementierung einer Barriere, die in einer Schleife auf die Freigabe der Barriere wartet.

Das Argument `TeamSize` spezifiziert die Anzahl der Threads, die `wait_barrier` zuvor aufrufen müssen, bevor sie vom Warten zurückkehren können. Der Wert von `TeamSize` muss größer als Null sein.

Parameter

▷ **AttribName** (input_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Barriere-Attribut.
Defaultwert: []
Werteliste: `AttribName` ∈ {'type'}

▷ **AttribValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Barriere-Attributwert.
Parameteranzahl: AttribValue == AttribName
Defaultwert: []
Werteliste: `AttribValue` ∈ {'spin', 'sleep'}

▷ **TeamSize** (input_control) . . . . . number ⇝ *integer*
Größe des Barriere Teams.
Defaultwert: 1

▷ **BarrierHandle** (output_control) . . . . . barrier ⇝ *handle*
Barriere-Synchronisationsobjekt.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_barrier` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`wait_barrier`, `clear_barrier`

Modul

Foundation

create_condition (: : AttribName, AttribValue : ConditionHandle)

Erzeugen einer Condition-Variable.

Eine Condition-Variable (kurz.: „Condition“) ist ein Synchronisationsobjekt, das Threads erlaubt ihre Ausführung zu unterbrechen und den Prozessor freizugeben, bis eine Eigenschaft von gemeinsamen Daten erfüllt ist. Die grundlegende Vorgehensweise bei Conditions ist folgende: signalisiere die Condition, wenn die Bedingung erfüllt wird, und warte auf die Bedingung, indem die Threadausführung abgebrochen wird, bis ein anderer Thread die Bedingung signalisiert hat.

Eine Condition-Variable kann immer nur zusammen mit einem Mutexobjekt verwendet werden, um Race-Bedingungen zu vermeiden. Diese würden entstehen, wenn ein Thread auf eine Condition-Variable warten möchte, ein andere Thread jedoch die Condition signalisiert, gerade bevor der erste Thread tatsächlich auf die Bedingung wartet.

`create_condition` erzeugt und initialisiert die Condition-Variable `ConditionHandle` entsprechend der Attribute, die dem Parameterpaar `AttribName` und `AttribValue` übergeben werden. Die aktuelle Implementierung unterstützt jedoch keine Attribute, weshalb dem Parameter `AttribName` nur der leere String ” übergeben werden kann.

Parameter

▷ **AttribName** (input_control) . . . . . number(-array) ⇝ *string* / integer / real
Condition-Attribut.
Defaultwert: []

▷ **AttribValue** (input_control) . . . . . number(-array) ⇝ *string* / integer / real
Condition-Attributwert.
Parameteranzahl: AttribValue == AttribName
Defaultwert: []

▷ **ConditionHandle** (output_control) . . . . . condition ⇝ *handle*
Condition-Synchronisationsobjekt.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_condition` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`wait_condition`, `timed_wait_condition`, `signal_condition`

Siehe auch

`clear_condition`

Modul

Foundation

`create_event` `( : : AttribName, AttribValue : EventHandle )`

Erzeugen eines Event-Synchronisationsobjektes.

Ein Event ist ein Mutex, der keinem Thread gehört. D.h., er kann von einem anderen Thread freigegeben werden als von dem, der ihn gesperrt hat. Daher werden Events meist dann eingesetzt, wenn ein Thread einem anderem, wartenden Thread signalisieren soll, dass dieser weiterlaufen kann. Dieses Verständnis von Events leitet sich aus der Definition eines Events der WIN32-API oder der einer Semaphore (mit Zähler 0) der POSIX-Thread-Spezifikation ab.

`create_event` legt ein Eventobjekt an und initialisiert es entsprechend der Attribute, die über die Parameter `AttribName` und `AttribValue` gesetzt werden. `AttribName` definiert die Klasse und `AttribValue` den Typ des Events. Im Folgenden werden die möglichen Attributwerte für die gelisteten Attributnamen beschrieben:

'' der leere String setzt die Standardattribute.

'type' spezifiziert das Verhalten eines Events, wenn ein Thread auf ein Signal warten muss:

'sleep' unterbricht den aufrufenden Thread und legt ihn schlafen (Standard).

'spin' ist die schnelle Implementierung eines Events, die in einer Schleife auf die Signalisierung des Events wartet. Dieser Typ ist nicht rekursiv.

Nach der Initialisierung ist ein Event gesperrt.

Parameter

▷ **`AttribName`** (input_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Event-Attribut.
Defaultwert: []
Werteliste: `AttribName` ∈ {'type'}

▷ **`AttribValue`** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Event-Attributwert.
Parameteranzahl: AttribValue == AttribName
Defaultwert: []
Werteliste: `AttribValue` ∈ {'spin', 'sleep'}

▷ **`EventHandle`** (output_control) . . . . . event ⇝ *handle*
Event-Synchronisationsobjekt.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_event` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`wait_event`, `signal_event`, `clear_event`

Siehe auch

clear_event

Modul

Foundation

create_message (: : : MessageHandle)

Erzeugen einer neuen Nachricht.

create_message erzeugt eine neue leere Nachricht. Der Ausgabeparameter MessageHandle referenziert das erzeugte Nachrichtenobjekt und wird dazu verwendet, um die Nachricht in den nachfolgenden Operatoraufrufen identifizieren zu können.

Eine Nachricht ist ein Objekt, das ähnlich einem assoziativem Array (Wörterbuch) aufgebaut ist. Diese erzeugten Nachrichten können dann zwischen Threads einfach über eine Warteschlange für Nachrichten ausgetauscht werden. Alternativ dazu stehen in HALCON auch Dictionaries (create_dict) zur Verfügung, um Parameter innerhalb von Programmen und Prozeduren zu gruppieren.

Nachrichten erlauben es, eine beliebige Anzahl von Daten zu speichern, welche mit Hilfe eines eindeutigen Schlüssels (String oder Integer) identifiziert werden. Jeder Schlüssel kann entweder einen Tupeldaten-Parameter oder ein ikonisches Objekt referenzieren. Diese Daten werden in der Nachricht mit Hilfe der Operatoren set_message_obj und set_message_tuple gespeichert und können über die Operatoren get_message_obj und get_message_tuple wieder ausgelesen werden.

Die Daten, welche in den Nachrichten referenziert werden, sind stets Kopien ihrer Originale. Damit können die originalen Daten, nachdem sie in der Nachricht durch die Operatoren set_message_tuple oder set_message_obj gespeichert wurden, verändert oder gelöscht werden, ohne die Nachricht selbst zu verändern. Eine Ausnahme bilden Handles, welche von HALCON als üblicher Integerwert behandelt werden. Beim Speichern eines Handles in einer Nachricht wird daher nur der Handle-Wert, nicht aber das Handle-Objekt selbst kopiert.

Wie bereits oben erwähnt können die erzeugten Nachrichten zwischen Threads einfach über eine Warteschlange für Nachrichten ausgetauscht werden. Mit Hilfe des Operators enqueue_message können Nachrichten aus mehreren Erzeuger-Threads an eine Warteschlange angehängt und über den Operator dequeue_message aus mehreren Verbraucher-Threads wieder aus der Warteschlange entnommen werden. All diese Operationen werden intern synchronisiert, so dass der Mechanismus über die Warteschlange ohne externe Synchronisation auskommt. enqueue_message kopiert die Nachricht in die Warteschlange, weshalb das ursprüngliche Nachrichtenobjekt danach sofort weiterverwendet werden kann, ohne das kopierte Nachrichtenobjekt zu beeinflussen.

Parameter

▷ **MessageHandle** (output_control) ..... message ⇝ *handle*
Nachrichtenobjekt.
Parameteranzahl: MessageHandle == 1
Zusicherung: MessageHandle != 0

Beispiel

```
MessageHandles := []
for idx := 0 to 4 by 1
  create_message (MessageHandle)
  MessageHandles[idx]:= MessageHandle
endfor
* ...
```

Ergebnis

create_message liefert den Wert TRUE, sofern kein Allokierungsfehler auftritt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

set_message_tuple, set_message_obj

Alternativen

create_dict

Siehe auch

clear_message, set_message_tuple, get_message_tuple, set_message_obj, get_message_obj, set_message_param, get_message_param, enqueue_message, dequeue_message

Modul

Foundation

create_message_queue (: : : QueueHandle)

Erzeugen einer leeren Warteschlange für Nachrichten.

create_message_queue erzeugt eine leere Warteschlange für Nachrichtenobjekte. QueueHandle referenziert das erzeugte Warteschlangenobjekt und wird dazu verwendet, um die Warteschlange in den nachfolgenden Operatoraufrufen identifizieren zu können.

Warteschlangen sind FIFO-Puffer, mit welchen sich beliebige Datensätze zwischen unterschiedlichen Threads austauschen lassen. Der Zugriff auf eine Warteschlange ist intern synchronisiert, so dass vom Benutzer keine zusätzlichen Synchronisationsmaßnahmen notwendig sind. Die Datensätze wandern durch die Warteschlange in sogenannten Nachrichten (vergleiche create_message).

Mehrere Erzeugerthreads können Nachrichten gleichzeitig über enqueue_message anhängen, während mehrere Verbraucherthreads über dequeue_message gleichzeitig Nachrichten aus diesem Datenstrom empfangen können. Dabei können mehrere Nachrichten mit einem einzigen Aufruf von enqueue_message eingereiht werden. In diesem Fall wandern die Nachrichten gemeinsam durch die Warteschlange und werden mit einem einzigen Aufruf von dequeue_message empfangen.

enqueue_message kopiert die Nachricht in die Warteschlange, weshalb das ursprüngliche Nachrichtenobjekt danach sofort weiterverwendet werden kann, ohne das kopierte Nachrichtenobjekt zu beeinflussen.

Parameter

▷ **QueueHandle** (output_control) . . . . . message_queue ⇝ *handle*
Warteschlangenobjekt.
Parameteranzahl: QueueHandle == 1
Zusicherung: QueueHandle != 0

Beispiel

```
create_message_queue (ProducerQueue)
create_message_queue (ResultQueue)
* ...
```

Ergebnis

create_message_queue liefert den Wert TRUE, sofern kein Allokierungsfehler auftritt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`enqueue_message`, `dequeue_message`

Siehe auch

`clear_message_queue`, `enqueue_message`, `dequeue_message`, `set_message_queue_param`, `get_message_queue_param`, `create_message`, `clear_message`, `set_message_tuple`, `get_message_tuple`, `set_message_obj`, `get_message_obj`

Modul

Foundation

`create_mutex` (: : `AttribName`, `AttribValue` : `MutexHandle`)

Erzeugen eines Mutex-Synchronisationsobjektes.

Ein Mutex ist ein Synchronisationsobjekt, welches den wechselseitigen Ausschluss (MUTual EXclusion) zwischen den Threads eines Prozesses regelt. Damit lassen sich gleichzeitige Zugriffe auf gemeinsame Ressourcen wie beispielsweise globale Datenstrukturen blockieren um inkonsistente Zustände zu verhindern. Ein Mutex hat zwei mögliche Zustände: er ist frei, d.h. der Mutex wird von keinem Thread gehalten, oder gesperrt, d.h. er gehört einem bestimmten Thread. Threads, die versuchen einen Mutex zu sperren, der bereits von einem anderen Thread gehalten wird, müssen warten, bis dieser Thread den Mutex wieder frei gibt. Anders als bei Events kann nur der Thread einen gesperrten Mutex wieder freigeben, der ihn auch hält, d.h. der ihn gesperrt hat.

`create_mutex` legt ein Mutexobjekt an und initialisiert es entsprechend der Attribute, die über die Parameter `AttribName` und `AttribValue` spezifiziert werden. `AttribName` definiert den Parameter und `AttribValue` den Wert des Parameters. Die folgende Beschreibung beschreibt für die möglichen Parametern alle unterstützten Werte:

'' der leere String setzt die Standardattribute.

'type' spezifiziert das Verhalten eines Mutexes, wenn ein Thread versucht einen bereits gesperrten Mutex zu besitzen:

- *'sleep'* unterbricht den aufrufenden Thread und legt ihn schlafen (Standard). Das Verhalten ist allerdings undefiniert, wenn der gesperrte Mutex bereits dem Thread gehört.
- *'spin'* ist die schnelle Implementierung eines Mutexes, die in einer Schleife auf die Freigabe des Mutexes wartet. Das Verhalten ist allerdings undefiniert, wenn der gesperrte Mutex bereits dem Thread gehört.
- *'recursive'* unterbricht den aufrufenden Thread und legt ihn schlafen, wenn dem aufrufenden Thread der Mutex nicht gehört. Allerdings kann ein rekursiver Mutex erneut von dem Thread gesperrt werden, dem er gehört. Ein rekursiver Mutex wird nicht freigegeben, bis die Anzahl der Freigaben die Anzahl der Sperraufrufe erreicht.

Nach der Initialisierung ist der Mutex frei.

Parameter

▷ **`AttribName`** (input_control) . . . . . number(-array) ⇝ *string* / integer / real
Mutex-Attribut
Defaultwert: []
Werteliste: `AttribName` ∈ {'type'}

▷ **`AttribValue`** (input_control) . . . . . number(-array) ⇝ *string* / integer / real
Mutex-Attributwert
Parameteranzahl: AttribValue == AttribName
Defaultwert: []
Werteliste: `AttribValue` ∈ {'spin', 'sleep', 'recursive'}

▷ **`MutexHandle`** (output_control) . . . . . mutex ⇝ *handle*
Mutex-Synchronisationsobjekt

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_mutex` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`lock_mutex`, `clear_mutex`

Siehe auch

`clear_mutex`

Modul

Foundation

dequeue_message (: : QueueHandle, GenParamName, GenParamValue : MessageHandle)

Empfangen einer oder mehrerer Nachrichten aus einer Warteschlange.

`dequeue_message` empfängt eine Nachricht aus einer Warteschlange die in `QueueHandle` referenziert ist. Die Nachricht muss zuvor von einem beliebigen Thread mit Hilfe des Operators `enqueue_message` in die Warteschlange eingereiht worden sein.

Die Nachrichten werden in FIFO (first-in first-out) Reihenfolge empfangen, wobei die Nachrichten unabhängig von der Anzahl der wartenden Threads jeweils nur einmal empfangen werden können. Falls die Warteschlange Nachrichten enthält, entnimmt `dequeue_message` sofort die älteste Nachricht und gibt eine Referenz auf das Nachrichtenobjekt im Parameter `MessageHandle` zurück. Der Besitz auf die Nachrichtendaten geht damit von der Warteschlange auf die zurückgegebene Nachrichtenreferenz über.

Falls die Warteschlange leer ist, wartet `dequeue_message` bis ein Nachrichtenobjekt für die Entnahme aus der Warteschlange verfügbar ist (nachdem sie zuvor von einem anderen Thread in die Warteschlange mit Hilfe von `enqueue_message` eingereiht wurde).

Die Daten, die ein Nachrichtenobjekt enthält können anschließend über die Operatoren `get_message_param`, `get_message_tuple` oder `get_message_obj` abgefragt werden.

Die Nachrichtenobjekte, die man von `dequeue_message` erhält, können anschließend wiederverwendet, d.h. modifiziert und/oder in eine weitere Warteschlange eingereiht werden.

Falls mehrere Nachrichtenobjekte über einen einzelnen Aufruf von `enqueue_message` in die Warteschlange eingereiht wurden, werden all diese Nachrichtenobjekte auch wieder zusammen durch einen einzelnen Aufruf von `dequeue_message` der Warteschlange entnommen und als Tupel im Parameter `MessageHandle` zurückgegeben.

Der Zugriff auf die Warteschlange ist intern synchronisiert. Es müssen daher keine weiteren Synchronisationsmaßnahmen getroffen werden.

Es kann z.B. während einer Rekonfiguration oder dem Aufräumen für eine Applikation notwendig werden, Threads die im Operator `dequeue_message` auf Nachrichten warten aufzuwecken und die Operation abzubrechen. Dies erreicht man indem man den Operator `set_message_queue_param` mit dem Parameter *'abort_dequeuing'* verwendet. In diesem Fall wird der blockierte Operator `dequeue_message` sofort mit dem Fehler H_ERR_MQCNCL zurückkehren.

Es ist möglich das Verhalten von `dequeue_message` durch die generischen Parameter `GenParamName` und `GenParamValue` zu beeinflussen. Aktuell wird nur ein einziger Parameter unterstützt:

'timeout': Spezifiziert eine Spanne für eine Zeitüberschreitung, welche bestimmt, wie lange der Operator auf eine Nachricht maximal warten soll, wenn die Warteschlange leer sein sollte. Wenn die Zeitspanne überschritten wurde, kehrt der Operator mit dem Fehler H_ERR_TIMEOUT zurück. Die Spanne für die Zeitüberschreitung kann als Integer- oder Doublewert in Sekunden angegeben werden oder durch den String *'infinite'*. Wenn keine Zeitüberschreitung angegeben wurde wird implizit eine unendliche Zeitspanne (infinite) angenommen und der Operator blockiert bis er ein Nachrichtenobjekt aus der Warteschlange entnehmen kann oder die Operation von außen abgebrochen wird.

Parameter

▷ **QueueHandle** (input_control) . . . . . message_queue ⇝ *handle*
Warteschlangenobjekt.
Parameteranzahl: QueueHandle == 1
Restriktion: QueueHandle != 0

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Name der generischen Parameter
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 'timeout'
Werteliste: `GenParamName` ∈ {'timeout'}

▷ **GenParamValue** (input_control) . . . . . tuple(-array) ⇝ *string* / integer / real
Werte der generischen Parameter
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 'infinite'
Werteliste: `GenParamValue` ∈ {'infinite'}

▷ **MessageHandle** (output_control) . . . . . message(-array) ⇝ *handle*
Entnommene Nachrichtenobjekte.
Parameteranzahl: MessageHandle > 0
Zusicherung: MessageHandle != 0

Beispiel

```
create_message_queue (Queue)
* ...
dequeue_message (Queue, [], [], Message)
get_message_obj (Image, Message, 'my_image')
```

Ergebnis

Im Erfolgsfall gibt `dequeue_message` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler können durch ungültige Parameter, Zeitüberschreitung (H_ERR_TIMEOUT) oder externes Abbrechen (H_ERR_MQCNCL) verursacht werden. Sollte in seltenen Fällen intern ein Fehler auftreten nachdem die Nachricht bereits aus der Queue genommen wurde, wird der Operator die Nachricht wieder an den Anfang der Warteschlange einreihen. Dies kann zu einer veränderten Reihenfolge der entnommenen Nachrichten führen, Wenn zwischenzeitlich ein anderer Thread eine Nachricht aus der Warteschlange entnommen hat.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`get_message_tuple`, `get_message_obj`, `get_message_param`

Siehe auch

`create_message_queue`, `clear_message_queue`, `enqueue_message`, `set_message_queue_param`, `get_message_queue_param`, `create_message`, `clear_message`, `set_message_tuple`, `get_message_tuple`, `set_message_obj`, `get_message_obj`

Modul

Foundation

enqueue_message (: : QueueHandle, MessageHandle, GenParamName, GenParamValue :)

Einreihen von Nachrichtenobjekten in eine Warteschlange.

enqueue_message reiht eine oder mehrere Nachrichtenobjekte in eine Warteschlange ein, die durch den Parameter QueueHandle referenziert wird. Die eingereihten Nachrichten können wieder durch den Operator dequeue_message der Reihe nach ausgelesen werden.

Die einzureihenden Nachrichtenobjekte (vgl. create_message) werden im Parameter MessageHandle übergeben welcher sowohl einzelne als auch Tupel von Nachrichtenobjekten akzeptiert. Wenn mehrere Nachrichtenobjekte zusammen der Warteschlange angehängt werden, werden diese auch wieder zusammen durch den Operator dequeue_message ausgegeben.

Es können gleichzeitig sowohl mehrere Erzeuger (enqueuing) Threads zu einer Warteschlange Nachrichten schicken als auch mehrere Verbraucher (dequeuing) Threads aus der gleichen Warteschlange Nachrichten holen. Die Warteschlange ist als FIFO (first-in first-out) Puffer organisiert, wobei jede Nachricht unabhängig vom Thread nur einmal aus der Warteschlange ausgelesen werden kann.

Der Zugriff auf die Warteschlange ist intern synchronisiert, so dass vom Benutzer keine zusätzlichen externen Synchronisationsmaßnahmen notwendig sind. Wenn die Warteschlange leer ist und mindestens ein Verbraucherthread auf eine Nachricht in dequeue_message wartet, erhält einer der wartenden Threads eine Nachricht und wird aufgeweckt, sobald ein Erzeugerthread mit Hilfe von enqueue_message ebendiese Nachricht erfolgreich in die Warteschlange gestellt hat. Andernfalls wird die Nachricht an die Warteschlange asynchron angehängt, um an einen Verbraucherthread weitergereicht zu werden, sobald dieser bereit ist die Nachricht zu entnehmen.

enqueue_message kopiert die Nachricht in die Warteschlange, weshalb das ursprüngliche Nachrichtenobjekt danach sofort weiterverwendet werden kann, ohne das kopierte Nachrichtenobjekt zu beeinflussen.

Die Parameter GenParamName und GenParamValue sind für zukünftige Erweiterungen des Operators reserviert. Es werden aktuell keine generischen Parameterwerte unterstützt.

Falls die Warteschlange nach der Operation eine größere Anzahl von Nachrichten halten würde als die spezifizierte Maximalanzahl, wird der Fehler H_ERR_MQOVL zurückgegeben. Die Maximalanzahl von gehaltenen Nachrichten wird über den Parameterwert *'max_message_num'* im Operator set_message_queue_param festgelegt.

Parameter

- ▷ **QueueHandle** (input_control) . . . . . message_queue ⇝ *handle*
  Warteschlangenobjekt.
  Parameteranzahl: QueueHandle == 1
  Restriktion: QueueHandle != 0
- ▷ **MessageHandle** (input_control) . . . . . message(-array) ⇝ *handle*
  Einzureihende Nachrichtenobjekte.
  Parameteranzahl: MessageHandle > 0
  Restriktion: MessageHandle != 0
- ▷ **GenParamName** (input_control) . . . . . string-array ⇝ *string*
  Namen generischer Parameter.
  Parameteranzahl: GenParamName == GenParamValue
- ▷ **GenParamValue** (input_control) . . . . . tuple-array ⇝ *string* / integer / real
  Werte generischer Parameter.
  Parameteranzahl: GenParamName == GenParamValue

Beispiel

```
create_message_queue (Queue)
* ...
create_message (Message)
set_message_tuple (Message, 'mixed_tuple', ['The answer', 42])
enqueue_message (Queue, Message, [], [])
```

Ergebnis

Im Erfolgsfall gibt enqueue_message den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler können ungültige Parameter oder Allokierungsfehler sein.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_message_tuple, set_message_obj

Siehe auch

create_message_queue, clear_message_queue, dequeue_message, set_message_queue_param, get_message_queue_param, create_message, clear_message, set_message_tuple, get_message_tuple, set_message_obj, get_message_obj

Modul

Foundation

get_current_hthread_id (: : : HThreadID)

Gibt eine ID für den derzeit ausgeführten Thread zurück.

get_current_hthread_id gibt eine ID in HThreadID zurück, welche den derzeit ausgeführten Thread repräsentiert. Diese ID ist identisch zu den jeweiligen Thread IDs der Betriebssysteme (Windows Thread32, POSIX Threads unter UNIX Systemen) als auch zu der Thread ID welche die par_start Operation der HDevEngine zurückgibt. Sie kann verwendet werden, um mit interrupt_operator Operatoren abzubrechen, die in diesem Thread laufen.

Parameter

▷ **HThreadID** (output_control) ..... integer ⇝ *integer*
ID, welche den derzeitigen Thread repräsentiert.

Beispiel

```
global tuple HThreadID
get_current_hthread_id (HThreadID)
* call some slow operator...

* In a different thread
wait_seconds(2)
* Interrupt the long-running operator in the other thread
interrupt_operator (HThreadID, 'break')
```

Ergebnis

Wenn der derzeitige Thread ermittelt werden kann, liefert get_current_hthread_id TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

interrupt_operator

Siehe auch

interrupt_operator, set_operator_timeout

Modul

Foundation

get_message_obj (: ObjectData : MessageHandle, Key :)

Zurückgeben der in einer Nachricht von einem Schlüssel referenzierten ikonischen Objekte.

get_message_obj fragt aus einer Nachricht MessageHandle das mit einem Schlüssel Key assoziierte ikonische Objekt ObjectData ab. Das ikonische Objekt muss zuvor mit Hilfe des Operators set_message_obj in dem Nachrichtenobjekt abgespeichert worden sein.

Falls der spezifizierte Schlüssel Key nicht in dem Nachrichtenobjekt hinterlegt ist oder die mit ihm verbundenen Daten kein ikonisches Objekt sondern ein Datentupel ist, schlägt get_message_obj fehl. Die gespeicherten Schlüssel und die Information über die von ihnen referenzierten Daten kann über den Operator get_message_param abgefragt werden.

Parameter

- ▷ **ObjectData** (output_object) ..... object(-array) ⇝ *object*
  Vom Schlüssel referenziertes Objekt.
- ▷ **MessageHandle** (input_control) ..... message ⇝ *handle*
  Nachrichtenobjekt.
  Parameteranzahl: MessageHandle == 1
  Restriktion: MessageHandle != 0
- ▷ **Key** (input_control) ..... string ⇝ *string* / integer
  Schlüsselwort oder Integer.
  Parameteranzahl: Key == 1

Beispiel

```
* ...
get_message_param (Message,'key_exists',['simple_string','foo','my_image'],\
                   KeysPresence)
get_message_param (Message,'key_data_type',['simple_string','my_image'],\
                   KeysType)
get_message_obj (Image, Message, 'my_image')
```

Ergebnis

Im Erfolgsfall gibt get_message_obj den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler sind ungültige Parameter (Nachrichtenobjekt oder Schlüssel), Abwesenheit des Schlüssels in der Nachricht oder wenn der Schüssel andere Datentypen als ein ikonisches Objekt referenziert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

dequeue_message

Alternativen

get_message_tuple

Siehe auch

create_message, clear_message, set_message_tuple, get_message_tuple, set_message_obj, set_message_param, get_message_param, enqueue_message, dequeue_message

Modul

Foundation

get_message_param (: : MessageHandle, GenParamName, Key : GenParamValue)

Abfragen von Parametern oder Information zu einer Nachricht.

get_message_param fragt die aktuellen Werte der Parameter einer Nachricht oder anderweitige Informationen zum Status einer Nachricht ab.

Jeder Aufruf von `get_message_param` darf nur einen einzelnen Parameternamen in `GenParamName` enthalten. Es gibt zwei unterschiedliche Arten von Abfragen:

- Parameter, die auf die gesamte Nachricht angewendet werden. In diesem Fall dürfen keine Schlüssel übergeben werden, d.h. dem Parameter `Key` muss ein leeres Tupel übergeben werden.
- Parameter, die auf einen einzelnen Schlüssel angewendet werden. In diesem Fall müssen die Schlüssel in dem Parameter `Key` übergeben werden, auf die der Parameter angewendet werden soll. Die Schlüssel werden dabei in der gleichen Reihenfolge abgearbeitet wie sie dem Parameter `Key` übergeben werden.

Parameternamen, die unabhängig von Schlüsseln angewendet werden:

'message_keys': Fragt alle Schlüssel ab, die in einer Nachricht gespeichert sind, unabhängig von dem Typ der mit ihnen verbundenen Daten. Die Liste der Schlüssel wird über den Parameter `GenParamValue` als ein Tupel von Strings zurückgegeben. Für diese Abfrage muss dem Parameter `Key` ein leeres Tupel übergeben werden.

Parametername, die Schlüssel-spezifisch angewendet werden:

'key_exists': Gibt *1* zurück, wenn der übergebenen Schlüssel in der spezifizierten Nachricht gespeichert ist, sonst *0*. Die Ergebnisse werden im Parameter `GenParamValue` zurückgegeben, jeweils ein Wert pro übergebenem Schlüssel.

'key_data_type': Gibt den Wert *'tuple'* für Schlüssel zurück, die in der Nachricht Tupeldaten referenzieren. Die Tupeldaten selbst erhält man mit dem Operator `get_message_tuple`. Der Wert *'object'* wird für Schlüssel zurückgegeben, welche Objektdaten referenzieren. Die Objektdaten selbst werden vom Operator `get_message_obj` zurückgegeben. Die Ergebnisse werden im Parameter `GenParamValue` zurückgegeben, jeweils ein Wert pro Schlüssel. Dieser Parameter kann verwendet werden um dynamisch zu entscheiden, ob `get_message_tuple` oder `get_message_obj` verwendet werden muss um die Daten eines speziellen Schlüssels zu erhalten.

Parameter

▷ **`MessageHandle`** (input_control) . . . . . message ⇝ *handle*
Nachrichtenobjekt.
Parameteranzahl: MessageHandle == 1
Restriktion: MessageHandle != 0

▷ **`GenParamName`** (input_control) . . . . . string ⇝ *string*
Name des Parameters.
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 'message_keys'
Werteliste: `GenParamName` ∈ {'message_keys', 'key_exists', 'key_data_type'}

▷ **`Key`** (input_control) . . . . . string(-array) ⇝ *string* / integer
Schlüssel auf das der Parameter angewendet werden soll.

▷ **`GenParamValue`** (output_control) . . . . . tuple(-array) ⇝ *string* / integer / real
Wert der Parameterabfrage.

Beispiel

```
get_message_param (Message, 'message_keys', [], AllKeys)
get_message_param (Message, 'key_data_type', AllKeys, KeysType)
```

Ergebnis

Wenn alle Parameter und übergebenen Schlüssel gültig sind, gibt `get_message_param` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

dequeue_message

Siehe auch

create_message, clear_message, set_message_tuple, get_message_tuple, set_message_obj, get_message_obj, set_message_param, enqueue_message, dequeue_message

Modul

Foundation

get_message_queue_param (: : QueueHandle, GenParamName : GenParamValue)

Abfragen der Parameter einer Warteschlange für Nachrichten.

get_message_queue_param fragt die aktuellen Werte der Parameter einer Warteschlange oder Informationen zu ihrem Status ab.

Es können mit einem einzelnen Aufruf von get_message_queue_param mehrere Werte abgefragt werden, indem GenParamName mehrere Parameternamen übergeben werden. Die Parameterwerte werden in GenParamValue in korrespondierender Reihenfolge zu den Parameternamen zurückgegeben.

Es werden folgende Parameternamen unterstützt:

'is_empty': Gibt den Wert *1* zurück, wenn die Warteschlange leer ist, sonst *0*.

'message_num': Gibt die Anzahl der Nachrichten zurück, die aktuell in der Warteschlange eingereiht sind.

'max_message_num': Gibt den aktuellen Wert für die maximale Anzahl von Nachrichten zurück, die in einer Warteschlange eingereiht werden können. Dieser Wert kann mit set_message_queue_param eingestellt werden. Die Voreinstellung *-1* bedeutet, dass eine unbegrenzte Anzahl von Nachrichten eingereiht werden kann.

Parameter

▷ **QueueHandle** (input_control) ..... message_queue ⇝ *handle*
Warteschlangenobjekt.
Parameteranzahl: QueueHandle == 1
Restriktion: QueueHandle != 0

▷ **GenParamName** (input_control) ..... string(-array) ⇝ *string*
Parameternamen.
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 'max_message_num'
Werteliste: GenParamName ∈ {'is_empty', 'message_num', 'max_message_num'}

▷ **GenParamValue** (output_control) ..... tuple(-array) ⇝ *string* / integer / real
Abgefragte Parameterwerte.
Parameteranzahl: GenParamName == GenParamValue

Beispiel

```
create_message_queue (QueueHandle)
set_message_queue_param ( QueueHandle, 'max_message_num', 10)
* ...
get_message_queue_param( QueueHandle, 'message_num', Num)
```

Ergebnis

Wenn alle Parameter gültig sind liefert get_message_queue_param den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_message_queue`, `enqueue_message`, `dequeue_message`

Siehe auch

`create_message_queue`, `clear_message_queue`, `enqueue_message`, `dequeue_message`, `set_message_queue_param`, `create_message`, `clear_message`, `set_message_tuple`, `get_message_tuple`, `set_message_obj`, `get_message_obj`

Modul

Foundation

`get_message_tuple` (: : MessageHandle, Key : TupleData)

Zurückgeben der in einer Nachricht über einen Schüssel referenzierten Tupeldaten.

`get_message_tuple` fragt aus einer Nachricht `MessageHandle` die mit einem Schlüssel `Key` assoziierten Tupeldaten `TupleData` ab. Die Tupeldaten müssen zuvor mit Hilfe des Operators `set_message_tuple` in dem Nachrichtenobjekt abgespeichert worden sein.

Falls der spezifizierte Schlüssel `Key` nicht in dem Nachrichtenobjekt hinterlegt ist oder der mit ihm verbundene Eintrag kein Datentupel sondern ein ikonisches Objekt ist, schlägt `get_message_tuple` fehl. Die gespeicherten Schlüssel und die Information über die von ihnen referenzierten Daten kann über den Operator `get_message_param` abgefragt werden.

Parameter

▷ **`MessageHandle`** (input_control) . . . . . message ⇝ *handle*
Nachrichtenobjekt.
Parameteranzahl: MessageHandle == 1
Restriktion: MessageHandle != 0

▷ **`Key`** (input_control) . . . . . string ⇝ *string* / integer
Schlüsselwort oder Integer.
Parameteranzahl: Key == 1

▷ **`TupleData`** (output_control) . . . . . tuple(-array) ⇝ *string* / integer / real / handle
Vom Schlüssel referenzierte Tupeldaten.

Beispiel

```
* ...
get_message_param (Message,'key_exists',['simple_string','foo','my_image'],\
                   KeysPresence)
get_message_param (Message,'key_data_type',['simple_string','my_image'],\
                   KeysType)
get_message_tuple (Message,'simple_string',TupleString)
```

Ergebnis

Im Erfolgsfall gibt `get_message_tuple` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler sind ungültige Parameter (Nachrichtenobjekt oder Schlüssel), Abwesenheit des Schlüssels in der Nachricht oder wenn der Schüssel andere Datentypen als ein Datentupel referenziert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

dequeue_message

Alternativen

get_message_obj

Siehe auch

create_message, clear_message, set_message_tuple, set_message_obj, get_message_obj, set_message_param, get_message_param, enqueue_message, dequeue_message

Modul

Foundation

get_threading_attrib (: : ThreadingHandle : ThreadingClass, AttribName, AttribValue)

Abfragen der Attribute eines Thread-/Synchronisationsobjekts.

get_threading_attrib bestimmt den Typ und die Attribute des Threading-Objektes welches dem Parameter ThreadingHandle übergeben wurde.

Parameter

- ▷ **ThreadingHandle** (input_control) ..... number ⇝ *integer*
  Threading Objekt.
- ▷ **ThreadingClass** (output_control) ..... string-array ⇝ *string*
  Klassenbezeichnung des Threading-Objektes.
- ▷ **AttribName** (output_control) ..... string-array ⇝ *string*
  Name des Attributs.
- ▷ **AttribValue** (output_control) ..... number-array ⇝ *string* / integer / real
  Wert des Attributs.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_threading_attrib den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

create_mutex, create_event, create_condition, create_barrier

Siehe auch

create_mutex, create_event, create_condition, create_barrier

Modul

Foundation

interrupt_operator (: : HThreadID, Mode :)

Abbrechen eines in einem anderen Thread laufenden Operators.

interrupt_operator versucht den derzeit im Thread HThreadID laufenden Operator zu unterbrechen. Wenn der derzeit laufende Operator unterbrechbar ist, wird er die Ausführung auf die in Mode gesetzte Art abbrechen (siehe Unten). Wenn der derzeit laufende Operator nicht unterbrechbar ist, oder wenn derzeit kein Operator in HThreadID läuft, hat interrupt_operator keine Auswirkung. Insbesondere wird die Ausführung späterer Operatoren nicht beeinflusst.

Das einen Thread repräsentierende Handle kann mittels get_current_hthread_id ermittelt werden.

interrupt_operator unterstützt momentan nur eine Art von Unterbrechung, die in Mode gesetzt werden kann. Es ist zu beachten, dass der im anderen Thread laufende, abzubrechende Operator die entsprechende Unterbrechbarkeit ebenfalls unterstützen muss. Ob ein Operator unterbrechbar ist, und welche Arten der Unterbrechbarkeit er unterstützt, kann im Abschnitt Äusführungsinformationen"in der Referenzdokumentation des Operators eingesehen oder mit Hilfe des Parameters *'interrupt_mode'* des Operators get_operator_info ausgelesen werden.

Einige Operatoren, für die in der Referenzdokumentation keine Unterbrechbarkeit angegeben ist, können dennoch an einigen Stellen ihrer Ausführung unterbrochen werden. Derartige Unterbrechbarkeiten sind allerdings nicht garantiert und können sich zwischen verschiedenen HALCON-Versionen ohne weitere Ankündigung verändern.

'cancel': Bricht die Ausführung des Operators ab. Alle eventuell bisher berechneten Zwischenergebnisse werden verworfen und der Operator gibt den Fehlercode H_ERR_CANCEL (22) zurück.

Achtung

Nicht alle Operatoren sind unterbrechbar. Wenn ein Operator unterbrechbar ist, ist dies im Abschnitt über Ausführungsinformationen in dessen Referenzdokumentation vermerkt.

Es gibt keine Garantie über die Granularität der Unterbrechungen. Diese kann vom Operator, dessen Eingabedaten und dem ausführenden System abhängen. Üblicherweise ist die Granularität besser als 10 ms.

Parameter

▷ **HThreadID** (input_control) ..... integer ⇝ *integer*
Zu unterbrechender Thread.

▷ **Mode** (input_control) ..... string ⇝ *string*
Art der Unterbrechung.
Defaultwert: 'cancel'
Werteliste: Mode ∈ {'cancel'}

Beispiel

```
global tuple HThreadID
get_current_hthread_id (HThreadID)
* call some slow operator...

* In a different thread
wait_seconds(2)
* Interrupt the long-running operator in the other thread
interrupt_operator (HThreadID, 'cancel')
```

Ergebnis

Wenn alle Parameter gültig sind liefert interrupt_operator den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

get_current_hthread_id

Alternativen

set_operator_timeout

Siehe auch

get_current_hthread_id, set_operator_timeout

Modul

Foundation

lock_mutex (: : MutexHandle :)

Sperren eines Mutex-Synchronisationsobjektes.

lock_mutex sperrt den Mutex, gegeben durch MutexHandle. Falls der Mutex gerade frei ist, wird er unverzüglich gesperrt und gehört dem aufrufenden Thread. lock_mutex kehrt in diesem Fall sofort zurück. Falls der Mutex bereits durch einen anderen Thread gesperrt wurde, wartet der aufrufende Thread solange, bis der Mutex frei ist. Die Art, wie gewartet wird, bestimmen die Attribute des Mutex, wie sie während der Initialisierung in create_mutex gesetzt wurden.

Parameter

▷ **MutexHandle** (input_control) . . . . . mutex ⇝ *handle*
Mutex-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert lock_mutex den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_mutex

Nachfolger

unlock_mutex

Siehe auch

try_lock_mutex

Modul

Foundation

read_message (: : FileName, GenParamName,
GenParamValue : MessageHandle)

Lesen einer Nachricht aus einer Datei.

read_message liest aus der durch FileName gegebenen Datei eine mit write_message abgespeicherte Nachricht ein. Die Dateiendung für die Nachricht ist in HALCON 'hmsg'.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname der zu lesenden Datei.
Dateiendung: .hmsg

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: []
Werteliste: GenParamName ∈ {'name'}

▷ **GenParamValue** (input_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Wert des generischen Parameters.
Defaultwert: []

▷ **MessageHandle** (output_control) . . . . . message ⇝ *handle*
Nachrichtenobjekt.
Parameteranzahl: MessageHandle == 1
Zusicherung: MessageHandle != 0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_message` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`write_message`

Siehe auch

`write_message`, `serialize_handle`, `deserialize_handle`

Modul

Foundation

set_message_obj (ObjectData : : MessageHandle, Key :)

Hinzufügen eines Schlüssel-Objekt-Paares zu einem Nachrichtenobjekt.

`set_message_obj` speichert in einer Nachricht ähnlich zu einem assoziativem Array ein ikonisches Objekt in Verbindung mit einem Schlüssel ab. Das Nachrichtenobjekt wird im Parameter `MessageHandle` übergeben.

`ObjectData` wird durch die Operation kopiert (vgl. `copy_obj`) und kann daher anschließend ohne Seiteneffekte weiterverwendet oder gelöscht werden. Sowohl leere Objekte als auch ikonische Tupelobjekte sind zulässig und können von einem Schlüssel referenziert werden. Falls bereits Daten (Tupel oder Objekt) durch den gegebenen Schlüssel referenziert werden, werden die alten Daten freigegeben und durch `ObjectData` ersetzt.

`Key` ist ein String oder ein Integer, wobei bei Strings die Groß-/Kleinschreibung beachtet wird.

Die ikonischen Objekte für den übergebenen Schlüssel können mit Hilfe des Operators `get_message_obj` aus dem Nachrichtenobjekt wieder ausgelesen werden.

Parameter

▷ **ObjectData** (input_object) ..... object(-array) ⇝ *object*
Vom Schlüssel zu referenzierendes ikonisches Objekt.

▷ **MessageHandle** (input_control) ..... message ⇝ *handle*
Nachrichtenobjekt.
Parameteranzahl: MessageHandle == 1
Restriktion: MessageHandle != 0

▷ **Key** (input_control) ..... string ⇝ *string* / integer
Schlüsselwort oder Integer.
Parameteranzahl: Key == 1

Beispiel

```
create_message (Message)
read_image( Image, 'filename')
set_message_obj (Image, Message, 'my_image')
```

Ergebnis

Im Erfolgsfall gibt `set_message_tuple` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler sind ungültige Parameter (Nachrichtenobjekt oder Schlüssel) oder ein Allokierungsfehler.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MessageHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_message`

Nachfolger

`enqueue_message`, `set_message_tuple`

Alternativen

`set_message_tuple`

Siehe auch

`create_message`, `clear_message`, `set_message_tuple`, `get_message_tuple`, `get_message_obj`, `set_message_param`, `get_message_param`, `enqueue_message`, `dequeue_message`

Modul

Foundation

set_message_param (: : MessageHandle, GenParamName, Key, GenParamValue :)

Setzen von Parametern oder Ausführen von Kommandos auf einer Nachricht.

`set_message_param` kann Parameter einer Nachricht setzen oder Aktionen auf dieser Nachricht ausführen.

Jeder Aufruf von `set_message_param` darf nur einen einzelnen Parameternamen in `GenParamName` enthalten. Es gibt zwei unterschiedliche Arten von Parametern/Aktionen:

- Parameter/Aktionen, die auf die gesamte Nachricht angewendete werden können. In diesem Fall dürfen keine Schlüssel spezifiziert werden, d.h. es muss dem Parameter `Key` ein leeres Tupel übergeben werden.
- Parameter/Aktionen, die auf einzelne Schlüssel angewendet werden. In diesem Fall muss eine Tupel von Schlüsseln dem Parameter `Key` übergeben werden. Die Schlüssel werden dann in der selben Reihenfolge abgearbeitet in der sie im Parameter `Key` spezifiziert wurden.

Parameternamen, die unabhängig von Schlüsseln angewendet werden:

'remove_all_keys': Entfernt alle Schlüssel die aktuell in der Nachricht gespeichert sind und gibt die von den Schlüsseln referenzierten ikonischen Objekten oder Tupeldaten frei. Als Ergebnis erhält man eine leere Nachricht. Der Parameter `GenParamValue` wird in diesem Fall ignoriert, während für `Key` ein leeres Tupel übergeben werden muss.

Parametername, die Schlüssel-spezifisch angewendet werden:

'remove_key': Entfernt die Schlüssel welche im Parameter `Key` spezifiziert werden und gibt die vom Schlüssel referenzierten ikonischen Objekten oder Tupeldaten frei. Falls ein Fehler während der Verarbeitung eines oder mehrerer Schlüssel auftritt, wird versucht so weit wie möglich alle weiteren Schlüssel zu entfernen bevor der Operator mit dem Fehler zurückkehrt. Der Parameter `GenParamValue` wird in diesem Fall ignoriert.

Parameter

▷ **MessageHandle** (input_control) . . . . . message ⇝ *handle*
Nachrichtenobjekt.
Parameteranzahl: MessageHandle == 1
Restriktion: MessageHandle != 0

▷ **GenParamName** (input_control) . . . . . string ⇝ *string*
Name des Parameters oder der Aktion.
Parameteranzahl: GenParamName == 1
Defaultwert: 'remove_key'
Werteliste: GenParamName ∈ {'remove_key', 'remove_all_keys'}

▷ **Key** (input_control) . . . . . string(-array) ⇝ *string* / integer
Schlüssel auf den der Parameter/die Aktion angewendet wird.

▷ **GenParamValue** (input_control) . . . . . tuple(-array) ⇝ *string* / integer / real
Werte des Parameters oder der Aktion.

Beispiel

```
* Remove some keys
set_message_param (Message, 'remove_key', ['my_image', 'simple_string'], [])
```

Ergebnis

Im Erfolgsfall gibt set_message_param den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MessageHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

dequeue_message, create_message

Nachfolger

enqueue_message, clear_message

Siehe auch

create_message, clear_message, set_message_tuple, get_message_tuple, set_message_obj, get_message_obj, get_message_param, enqueue_message, dequeue_message

Modul

Foundation

set_message_queue_param (: : QueueHandle, GenParamName, GenParamValue :)

Setzen von Parametern für eine Warteschlange von Nachrichten.

set_message_queue_param setzt Parameter für eine Warteschlange von Nachrichten oder führt Aktionen auf einer solchen aus.

GenParamName können mehrere Parameter und Aktionen und in GenParamValue die korrespondierenden Werte übergeben werden. Dabei müssen stets die gleiche Anzahl von Parameternamen und Werten übergeben

werden und die Reihenfolge der Werte muss mit der der Namen übereinstimmen. Die Parameter und Aktionen werden in der Reihenfolge abgearbeitet, in der sie in einem Tupel dem Parameter GenParamName übergeben wurden. Dabei kann im Falle von Aktionsparameter der Effekt eines früheren Aktionsparameters nicht rückgängig gemacht werden, wenn eine spätere Aktion fehlschlagen sollte.

Folgende Parameternamen werden unterstützt:

'abort_dequeuing': Bricht alle dequeue_message Operator ab, die in beliebigen Threads auf Nachrichten warten. Die abgebrochenen Aufrufe von dequeue_message geben den Fehler H_ERR_MQCNCL zurück. Jeder weitere zukünftige Aufruf von dequeue_message kehrt sofort mit dem Fehler H_ERR_MQCNCL zurück, unabhängig vom Status der Warteschlange.

Diese Aktion wird typischerweise verwendet bevor die Warteschlange aufgeräumt wird. Denn es ist nicht erlaubt die Warteschlange über clear_message_queue frei zu geben, während sie noch von anderen Operatoren verwendet wird. Daher ist es in der Regel notwendig, nach der Aktion *'abort_dequeuing'* auf die Beendigung der Threads zu warten, um sicherzustellen, dass kein Operator das Warteschlangenobjekt mehr verwenden kann.

Die Aktion wirkt sich jedoch nicht auf die Nachrichtenobjekte aus, die in der Warteschlange gespeichert sind. Der Effekt, dass auch alle weiteren Aufruf von dequeue_message abgebrochen werden, kann mit Hilfe der Aktion *'restore_queue'* wieder aufgehoben werden.

Der korrespondierende Parameterwert der Aktion muss den Integerwert *1* haben.

'restore_queue': Hebt die Auswirkungen einer vorangegangenen *'abort_dequeuing'* Aktion wieder auf. Dies bedeutet, dass der nächste Aufruf von dequeue_message wieder Nachrichtenobjekte aus der Warteschlange empfangen wird.

Der korrespondierende Parameterwert der Aktion muss den Integerwert *1* haben.

'flush_queue': Leert die Warteschlange. Alle sich in der Warteschlange befindenden Nachrichten werden aus der Warteschlange entfernt und alle damit verbundenen Ressourcen freigegeben, insbesondere die im Nachrichtenobjekt gespeicherten Tupel und Objektdaten.

Der korrespondierende Parameterwert der Aktion muss den Integerwert *1* haben.

'max_message_num': Setzt die maximale Anzahl von Nachrichten, die eine Warteschlange zu einem Zeitpunkt enthalten kann. Der Aufruf des Operators enqueue_message gibt dann den Wert H_ERR_MQOVL zurück, wenn die Warteschlange nach der Operation mehr Nachrichten enthalten würde als das gesetzte Maximum.

Das Setzen des Parameters *'max_message_num'* hat jedoch keine Auswirkung auf bereits eingereihte Nachrichtenobjekte. D.h. es werden durch diese Aktion keine Nachrichten aus der Warteschlange entfernt.

Der Parameterwert in GenParamValue kann ein positiver Integerwert oder *-1* sein, falls die Größe der Warteschlange nicht limitiert sein soll (default). Der String *'infinite'* wird ebenfalls akzeptiert.

Parameter

▷ **QueueHandle** (input_control) ..... message_queue ⇝ *handle*
Warteschlangenobjekt.
Parameteranzahl: QueueHandle == 1
Restriktion: QueueHandle != 0

▷ **GenParamName** (input_control) ..... string(-array) ⇝ *string*
Namen der Parameter oder Aktionen.
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 'max_message_num'
Werteliste: GenParamName ∈ {'abort_dequeuing', 'restore_queue', 'flush_queue', 'max_message_num'}

▷ **GenParamValue** (input_control) ..... tuple(-array) ⇝ *string* / integer / real
Werte zu den Parameternamen oder Aktionen.
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 1
Wertevorschläge: GenParamValue ∈ {1, -1}

Beispiel

```
* abort waiting dequeue_message operators
set_message_queue_param (Queue, 'abort_dequeuing', 1)
```

Ergebnis

Im Erfolgsfall gibt `set_message_queue_param` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_message_queue`

Nachfolger

`clear_message_queue`

Siehe auch

`create_message_queue`, `clear_message_queue`, `enqueue_message`, `dequeue_message`, `get_message_queue_param`, `create_message`, `clear_message`, `set_message_tuple`, `get_message_tuple`, `set_message_obj`, `get_message_obj`

Modul

Foundation

`set_message_tuple` (: : MessageHandle, Key, TupleData :)

Hinzufügen eines Schlüssel-Tupeldaten-Paares zu einer Nachricht.

`set_message_tuple` speichert in einer Nachricht ähnlich zu einem assoziativem Array ein Datentupel in Verbindung mit einem Schlüssel ab. Das Nachrichtenobjekt wird durch den Parameter `MessageHandle` referenziert.

`TupleData` wird in das Nachrichtenobjekt kopiert und kann anschließend weiterverwendet oder gelöscht werden. Ein leeres Tupel ist ebenfalls ein gültiger Wert, der mit einem Schlüssel referenziert werden kann. Falls bereits Daten (Tupel oder Objekt) durch den gegebenen Schlüssel referenziert werden, werden die alten Daten freigegeben und durch `TupleData` ersetzt.

`Key` ist ein String oder ein Integer, wobei bei Strings die Groß-/Kleinschreibung beachtet wird.

Die Tupeldaten für den übergebenen Schlüssel können mit Hilfe des Operators `get_message_tuple` aus dem Nachrichtenobjekt wieder ausgelesen werden.

Achtung

Falls `TupleData` Handles enthält, werden nur die Handle-Werte kopiert, nicht aber die Objekte welche die Handles referenzieren.

Parameter

- ▷ **`MessageHandle`** (input_control) . . . . . message ⇝ *handle*
  Nachrichtenobjekt.
  Parameteranzahl: MessageHandle == 1
  Restriktion: MessageHandle != 0
- ▷ **`Key`** (input_control) . . . . . string ⇝ *string* / integer
  Schlüsselwort oder Integer.
  Parameteranzahl: Key == 1
- ▷ **`TupleData`** (input_control) . . . . . tuple-array ⇝ *string* / integer / real / handle
  Vom Schlüssel zu referenzierende Tupeldaten.

Beispiel

```
create_message (Message)
set_message_tuple (Message, 'simple_integer', 27)
set_message_tuple (Message, 'simple_string', 'Hello world')
set_message_tuple (Message, 'mixed_tuple', ['The answer', 42])
```

Ergebnis

Im Erfolgsfall gibt set_message_tuple den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler sind ungültige Parameter (Nachrichtenobjekt oder Schlüssel) oder ein Allokierungsfehler.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- MessageHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_message

Nachfolger

enqueue_message, set_message_obj

Alternativen

set_message_obj

Siehe auch

create_message, clear_message, get_message_tuple, set_message_obj, get_message_obj, set_message_param, get_message_param, enqueue_message, dequeue_message

Modul

Foundation

signal_condition (: : ConditionHandle :)

Signalisieren eines Condition Synchronisationsobjektes.

signal_condition weckt einen der Threads auf, die auf die Condition Variable ConditionHandle warten. Falls kein Thread wartet, ist signal_condition wirkungslos. Warten mehrere Threads auf das Signalisieren von ConditionHandle, wird genau ein Thread von ihnen aufgeweckt, wobei es nicht spezifiziert ist welcher.

Parameter

▷ **ConditionHandle** (input_control) . . . . . condition ⇝ *handle*
Condition-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert signal_condition den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_condition, wait_condition

Siehe auch

wait_condition, timed_wait_condition

Modul

Foundation

signal_event (: : EventHandle :)

Entsperren eines Event-Synchronisationsobjektes.

signal_event gibt das Event gegeben durch EventHandle frei. Das Signalisieren eines Events, das bereits signalisiert ist hat keinen Effekt.

Parameter

▷ **EventHandle** (input_control) ..... event ⇝ *handle*
Event-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert signal_event den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

wait_event

Nachfolger

wait_event, clear_event

Modul

Foundation

timed_wait_condition (: : ConditionHandle, MutexHandle, Timeout :)

Begrenztes Warten auf das Signal eines Condition-Synchronisationsobjektes.

timed_wait_condition gibt MutexHandle frei und wartet auf das Signalisieren von ConditionHandle. Wie bei wait_condition geschehen Freigabe und Warten atomar, jedoch kann die Dauer des Wartens begrenzt werden. Falls ConditionHandle nicht während der von Timeout vorgegebenen Zeit (in Mikrosekunden) signalisiert wurde, wird der Mutex MutexHandle wieder gesperrt und timed_wait_condition wirft den Fehler H_ERR_TIMEOUT.

Negative Werte für Timeout bedeuten ein unendlich langes Warten.

Parameter

▷ **ConditionHandle** (input_control) ..... condition ⇝ *handle*
Condition-Synchronisationsobjekt.

▷ **MutexHandle** (input_control) ..... mutex ⇝ *handle*
Mutex-Synchronisationsobjekt.

▷ **Timeout** (input_control) ..... number ⇝ *integer*
Timeout in Mikrosekunden.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert timed_wait_condition den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Kommt es zu einer Zeitüberschreitung, wird der Wert H_ERR_TIMEOUT zurück gegeben.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`signal_condition`, `clear_condition`

Modul

Foundation

try_lock_mutex (: : MutexHandle : Busy)

Sperren eines Mutex-Synchronisationsobjektes, wenn er frei ist.

`try_lock_mutex` verhält sich identisch zu `lock_mutex`, bis auf dass der aufrufende Thread nicht blockiert, wenn der Mutex bereits gesperrt ist. Der Aufruf kommt in jedem Fall sofort zurück und enthält im Parameter `Busy` den Zustand des Mutexes vor dem Versuch ihn zu sperren. *1* bedeutet, dass der Mutex bereits gesperrt ist, *0*, dass der Mutex frei war und nun durch den aufrufenden Thread gesperrt wurde.

Parameter

- ▷ **MutexHandle** (input_control) . . . . . mutex ⇝ *handle*
  Mutex-Synchronisationsobjekt.
- ▷ **Busy** (output_control) . . . . . number ⇝ *integer*
  Mutex bereits gesperrt?

Ergebnis

Ist der Parameterwert korrekt, dann liefert `try_lock_mutex` den Wert TRUE.Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_mutex`

Nachfolger

`unlock_mutex`

Siehe auch

`lock_mutex`

Modul

Foundation

try_wait_event (: : EventHandle : Busy)

Sperren eines Event-Synchronisationsobjektes wenn es frei ist.

`try_wait_event` verhält sich identisch zu `wait_event`, bis auf dass der aufrufende Thread nicht blockiert, wenn er warten muss, bis das Eventobjekt signalisiert wird. Der Aufruf kommt in jedem Fall sofort zurück und enthält im Parameter `Busy` den Zustand des Events vor dem Versuch ihn zu sperren. *1* bedeutet, dass das Event bereits gesperrt ist, *0*, dass das Event signalisiert war und nun durch den aufrufenden Thread gesperrt wurde.

Parameter

- ▷ **EventHandle** (input_control) . . . . . event ⇝ *handle*
  Event-Synchronisationsobjekt.
- ▷ **Busy** (output_control) . . . . . number ⇝ *integer*
  Objekt bereits gesperrt?

Ergebnis

Ist der Parameterwert korrekt, dann liefert `try_wait_event` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_event`

Nachfolger

`signal_event`

Siehe auch

`wait_event`

Modul

Foundation

unlock_mutex (: : MutexHandle :)

Entsperren eines Mutex-Synchronisationsobjektes.

`unlock_mutex` gibt das Mutexobjekt, gegeben in Parameter `MutexHandle`, frei. Der Mutex muss dem aufrufenden Thread gehören. Die Freigabe eines bereits freien Mutexes hat keine Auswirkungen.

Parameter

▷ **MutexHandle** (input_control) ..... mutex ⇝ *handle*
Mutex-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert `unlock_mutex` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`lock_mutex`, `try_lock_mutex`

Nachfolger

`lock_mutex`, `try_lock_mutex`, `clear_mutex`

Modul

Foundation

wait_barrier (: : BarrierHandle :)

Warten auf das öffnen eines Barrieren-Synchronisationsobjektes.

`wait_barrier` blockiert den aufrufenden Thread an einer Barriere, gegeben durch `BarrierHandle`, bis die festgelegte Anzahl von Threads die Barriere erreicht haben. Diese Anzahl wird durch den Parameter `TeamSize` während des Anlegens des Barriere-Objektes in `create_barrier` spezifiziert.

Parameter

▷ **BarrierHandle** (input_control) . . . . . barrier ⇝ *handle*
Barriere-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert `wait_barrier` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`clear_barrier`

Modul

Foundation

wait_condition (: : ConditionHandle, MutexHandle :)

Warten auf das Signal eines Condition-Synchronisationsobjektes.

`wait_condition` gibt den Mutex, gegeben durch `MutexHandle`, frei (vergleiche `unlock_mutex`) und wartet bis die Condition-Variable `ConditionHandle` signalisiert wird. Freigeben und Warten geschehen atomar. Die Threadausführung wird unterbrochen und verursacht keine Prozessorlast, bis die Condition signalisiert wird. Der Mutex muss bei der Übergabe zu `wait_condition` vom aufrufenden Thread gesperrt sein. Bevor der Operator wieder zum aufrufenden Thread zurückkehrt, wird der Mutex wieder gesperrt (vergleiche `lock_mutex`).

Wenn der Programmierer dafür sorgt, dass alle Threads stets den Mutex sperren bevor sie die Condition signalisieren, dann ist sichergestellt, dass eine Condition nicht zwischen dem Zeitpunkt des Sperren eines Mutexes und dem Warten auf eine Condition signalisiert (und somit ignoriert) werden kann.

Parameter

▷ **ConditionHandle** (input_control) . . . . . condition ⇝ *handle*
Condition-Synchronisationsobjekt.

▷ **MutexHandle** (input_control) . . . . . mutex ⇝ *handle*
Mutex-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert `wait_condition` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`signal_condition`, `clear_condition`

Modul

Foundation

wait_event (: : EventHandle :)

Sperren eines Event-Synchronisationsobjektes.

wait_event wartet, bis das Eventobjekt in EventHandle signalisiert wurde. Wenn das Event signalisiert wurde, setzt wait_event den Zustand automatisch wieder auf gesperrt und kehrt zum Aufrufer zurück. Die Art des Wartens hängt von den Attributen des Events ab, die während der Initialisierung in create_event gesetzt wurden.

Parameter

▷ **EventHandle** (input_control) . . . . . event ⇝ *handle*
Event-Synchronisationsobjekt.

Ergebnis

Ist der Parameterwert korrekt, dann liefert wait_event den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_event

Nachfolger

signal_event

Modul

Foundation

write_message (: : MessageHandle, FileName, GenParamName, GenParamValue :)

Abspeichern einer Nachricht in einer Datei.

write_message speichert die Nachricht MessageHandle in der Datei FileName ab. Die Dateiendung für die Nachricht ist in HALCON 'hmsg'.

Wenn MessageHandle ein Handle enthält, das nicht serialisiert werden kann oder das schon freigegeben wurde, bricht write_message standardmäßig mit einer Fehlermeldung ab. Dieses Verhalten kann durch Setzen von GenParamName auf *'raise_error_if_content_not_serializable'* verändert werden, der zugehörige GenParamValue kann folgende Werte annehmen:

'true': Der Default: Fehler werden geworfen und der Schreibprozess abgebrochen.

'low_level': Nur Low-Level-Fehler werden geworfen. Anstatt des betreffenden Handles wird ein leeres Handle in FileName geschrieben und der Schreibprozess weitergeführt. Der Umgang mit HALCON Low-Level-Fehlern wird durch *'do_low_error'* in set_system bestimmt.

'false': Es werden keine Fehler geworfen. Anstatt des betreffenden Handles wird ein leeres Handle in FileName geschrieben und der Schreibprozess weitergeführt.

Parameter

▷ **MessageHandle** (input_control) . . . . . message ⇝ *handle*
Nachrichtenobjekt.
Parameteranzahl: MessageHandle == 1
Restriktion: MessageHandle != 0

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Dateiname der zu lesenden Datei.
Dateiendung: .hmsg

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: []
Werteliste: GenParamName ∈ {'raise_error_if_content_not_serializable'}

▷ **GenParamValue** (input_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Wert des generischen Parameters.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'true', 'false', 'low_level'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_message den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_message, deserialize_handle, create_message

Nachfolger

read_message

Siehe auch

read_message, serialize_handle, deserialize_handle

Modul

Foundation

25.8 Parallelisierung

get_aop_info (: : OperatorName, IndexName, IndexValue,
InfoName : InfoValue)

Abfragen von Informationen über HALCONs automatische Operator-Parallelisierung.

HALCON unterstützt einen Mechanismus, um das Verhalten der automatischen Parallelisierung von Operatoren (AOP) an Mehrprozessorarchitekturen anzupassen und Operatoren zu beschleunigen. Dazu können für jeden Operator, der sich auf Tupel-, Kanal- oder Domain-Ebene parallelisieren lässt, mit Hilfe von optimize_aop Abhängigkeiten des Laufzeitverhaltens von der Hardware und Eingabeparametern ermittelt werden. Der Operator get_aop_info liest Elemente dieser Informationen, während diese durch den Operator set_aop_info verändert werden können.

Um die operatorspezifischen Informationen adressieren zu können, sind die Daten in einer 3-stufigen Hierarchie indizierbar, die sich mit query_aop_info abfragen lässt. Die drei Indizierungsstufen setzen sich durch den Operatornamen, den ikonischen Bildtyp und optional einen Parameterstring zusammen, der einen speziellen Modus oder Methode des Operators spezifiziert. Während der Operatorname in OperatorName übergeben wird, definieren IndexName und IndexValue die letzteren beiden Indizes. IndexName erhält dabei die Indexkennung, IndexValue den Indexwert, wie sie von query_aop_info zurückgegeben werden.

get_aop_info gibt die spezifizierten Elemente des AOP-Wissens eines Operators in InfoValue zurück. Die Art der Information kann mit dem Parameter InfoName aus folgenden Werten gewählt werden:

'max_threads' gibt die größte zulässige Threadanzahl zurück, mit der der Operator parallelisiert werden darf. Wenn keine Daten vorhanden sind enthält InfoValue den Wert *-1* (unbeschränkt). Falls der Operator keine automatische Parallelisierung unterstützt, wird der Wert *1* zurückgegeben.

'split_level' gibt alle zulässigen Parallelisierungsebenen für den spezifizierten Operator, ikonischen Typ und Methodenparameter zurück. Mögliche Ebenen sind *'split_tuple'*, *'split_channel'*, *'split_domain'* und *'split_partial'*. Falls keine automatische Parallelisierung unterstützt wird oder alle Ebenen ausgeschaltet wurden, wird ein leerer String zurückgegeben. Um alle Parallelisierungsebenen zu erfahren, die vom Operator prinzipiell unterstützt werden, sollte der Operator get_operator_info verwendet werden.

'model' gibt den gespeicherten Modelltyp zurück. Mögliche Werte sind *'threshold'*, *'linear'* und *'mlp'* (siehe hierzu auch die Beschreibung des Operators optimize_aop). Falls keine Optimierungsdaten vorliegen wird ein leerer String zurückgegeben.

Parameter

- ▷ **OperatorName** (input_control) . . . . . string ⇝ *string* / integer
  Operator, zu dem Information abgefragt werden soll.
  Wertevorschläge: OperatorName ∈ {'mean_image', 'opening_circle', 'find_shape_model'}
- ▷ **IndexName** (input_control) . . . . . string-array ⇝ *string*
  Ikonischer Objekttyp für Information.
  Defaultwert: ['iconic_type','parameter:0']
- ▷ **IndexValue** (input_control) . . . . . string-array ⇝ *string*
  Parameterwert für Information
  Parameteranzahl: IndexName == IndexValue
  Defaultwert: ['byte','']
- ▷ **InfoName** (input_control) . . . . . string ⇝ *string*
  Art der Information.
  Defaultwert: 'max_threads'
  Wertevorschläge: InfoName ∈ {'max_threads', 'split_level', 'model'}
- ▷ **InfoValue** (output_control) . . . . . string(-array) ⇝ *string*
  AOP-Information.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_aop_knowledge den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

optimize_aop (: : OperatorName, IconicType, FileName, GenParamName, GenParamValue :)

Überprüfen der Hardware bezüglich des Potentials zur automatischen Operator-Parallelisierung.

HALCON unterstützt die automatische Parallelisierung (AOP) für eine Reihe von Operatoren. Mit optimize_aop lässt sich eine effiziente Ausführung der Parallelisierung unabhängig von der Rechnerarchitektur gewährleisten. Daher können alle HALCON-Programme ohne Anpassungen an die Multiprozessor-Hardware durch den Benutzer verwendet werden, so dass der Benutzer leicht von dem Potential paralleler Hardware profitiert. Standardmäßig verwendet HALCON einen Thread je Prozessor für die AOP. Allerdings kann abhängig von der Datenmenge und den Parametern eines Operators die Parallelisierung auf allen verfügbaren Threads zu überzogen und ineffizient sein. optimize_aop testet die parallele Datenverarbeitung der Operatoren und optimiert die AOP hinsichtlich der Threadanzahl. Hierfür wird jeder Operator überprüft, wie er sich auf Tupel-, Kanal- oder Domain-Ebene parallelisieren lässt (die partielle Ebene wird nicht betrachtet). Jeder dieser Operatoren wird mehrere Male ausgeführt - sequentiell wie parallel - mit unterschiedlichen Eingabeparametern und wechselnden Bildgrößen. Dadurch können Abhängigkeiten der Laufzeit des Operators von Eingabeparametern und Parallelisierung ermittelt werden. Dies kann bis zu einigen Stunden dauern, abhängig von den Parameterwerten dieses Operators. Für eine korrekte Optimierung ist es wichtig, keine anderen rechenintensiven Anwendungen gleichzeitig auf dem Rechner laufen zu lassen, da dies die Zeitmessungen des Operators stark beeinflussen und zu verfälschten Ergebnissen führen würde.

optimize_aop sammelt eine Menge Informationen für die aktuelle Rechnerarchitektur, die es HALCON im laufenden Betrieb ermöglicht die automatische Parallelisierung zu optimieren. Dieses Wissen über die Hardware kann in einer Binärdatei, gegeben durch FileName, abgespeichert werden und in zukünftigen HALCON-Anwendungen wiederverwendet oder auf andere, ähnliche Rechner übertragen werden. Wenn ein leerer String " als

Dateiname übergeben wird, speichert optimize_aop die gewonnenen Daten in die spezielle HALCON-System-Datei *'.aop_info'* unter Linux/macOS im HALCON-Installationsverzeichnis (siehe Umgebungsvariable $HALCONROOT) oder unter Windows in das Datenverzeichnis für gemeinsame Anwendungen. Diese Datei wird während der Initialisierungsphase beim Aufruf des ersten HALCON-Operators automatisch ausgelesen. Es ist zu beachten, dass die entsprechenden Zugriffsrechte für Lesen und Schreiben gesetzt sind. optimize_aop überprüft die entsprechenden Zugriffsrechte vor dem Starten des Optimierungsprozesses und gibt im Falle des Scheiterns einen entsprechenden Fehler zurück. Die geschriebene Datei kann durch den Operator read_aop_knowledge wieder gelesen werden.

Es ist ausreichend, optimize_aop einmal auf jedem Rechner auszuführen, auf dem die automatische Operator-Parallelisierung ausgeführt werden soll. Jedoch sollte der Operator erneut gestartet werden, wenn die Hardware des Rechners geändert wird, beispielsweise durch einen neue CPU, einen neuen Speicher oder ein verändertes Betriebssystem. Es ist notwendig optimize_aop einmal für jede neue Laufzeitumgebung auszuführen, da sich das Zeitverhalten von Operatoren geändert haben könnte. Eine Veränderung der Laufzeitumgebung wird verursacht durch einen Wechsel des Betriebssystems, wie zum Beispiel Windows auf Linux, durch unterschiedliche HALCON-Architekturen, unterschiedliche HALCON-Varianten, d.h. HALCON und HALCON XL, oder wenn einen neue HALCON-Version oder -Revision aktualisiert wird. Diese Hardwareabhängigkeiten werden zusammen mit dem Computernamen gemeinsam mit den AOP-Optimierungsdaten in der Datei gespeichert. Diese Attribute ordnen einen gespeicherten AOP-Datensatz einem speziellen Rechner zu und ermöglichen somit, dass AOP-Datensätze von unterschiedlichen Rechnern in der selben Datei gespeichert werden.

optimize_aop unterstützt einen Reihe von Parametern, um das Verhalten der Optimierung zu beeinflussen. Eine Auswahl von Operatornamen kann dem Parameter OperatorName übergeben werden. Dadurch lässt sich der Optimierungsprozess beispielsweise auf Operatoren mit problematischen Laufzeitverhalten beschränken. Wird ein leerer String *''* übergeben, werden alle Operatoren getestet. Der Testumfang von ikonischen Typen kann mit Hilfe des Parameters IconicType eingeschränkt werden. Der leere String *''* bewirkt, dass alle unterstützten ikonischen Typen eines Operators getestet werden. Zusätzlich kann der Optimierungsprozess mit Hilfe des Parameterpaares GenParamName und GenParamValue beeinflusst werden. GenParamName definiert dabei den Parameternamen, während GenParamValue den jeweiligen Wert spezifiziert, so dass übergebene Tupel bei beiden Parametern die gleiche Länge haben müssen. Im Folgenden sind für alle Parameternamen von GenParamName alle unterstützten Parameterwerte von GenParamValue beschrieben:

'none' für GenParamName hat keine Funktion, ignoriert aber den Parameter GenParamValue.

'system_mode', *'file_mode'* setzen jeweils die Art und Weise, wie die Informationen im HALCON-System b.z.w. der Datei aktualisiert werden sollen.

- *'truncate'* für GenParamValue löscht alle bestehenden Informationen, bevor das neue Wissen hinzugefügt wird.
- *'renew'* überschreibt bestehendes Wissen und fügt neues hinzu (Standard).
- *'append'* belässt bestehendes Operatorwissen und fügt ausschließlich Wissen hinzu, das noch nicht enthalten ist.
- *'nil'* führt den Optimierungsprozess durch, unterdrückt jedoch die Aktualisierung des Systems b.z.w. der Datei.

'parameters' testet auch laufzeitrelevante Parameter des entsprechenden Operators, falls der korrespondierende Wert in GenParamValue auf *'true'* gesetzt ist. Es werden Parameter berücksichtigt, die bedeutenden Einfluss auf die Laufzeit haben, wie beispielsweise Parameter, die eine Methode oder einen Mode des Operators die Größe eines Filters setzen. *'false'* unterdrückt die Überprüfung der Parameter zu Gunsten eines schnelleren Optimierungsprozesses (Standard).

'model' Setzt das zugrunde liegende Modell vom Laufzeitverhalten des parallelisierten Operators auf der aktuellen Hardware. Die Modelle, die durch den Parameter GenParamValue ausgewählt werden, unterscheiden sich in ihrer Adaptionsfähigkeit an Hardwareeigenheiten und dem Rechenaufwand sie zu bestimmen:

- *'threshold'* ermittelt für einen Operator, ob es sich rentiert, ihn auf der maximalen Threadanzahl zu parallelisieren. Dieses Modell ist standardmäßig auf Dual-Prozessor-Systemen eingestellt.
- *'linear'* spezifiziert ein lineares Skalierungsmodell, um die effizienteste Threadanzahl für einen Operator mit einer gegebene Datenmenge und Parametersatz. Dies ist der Standard auf Multi-Prozessor-Systemen.
- *'mlp'* ist das komplexeste, aber auch das anpassungsfähigste Model. Der Optimierungsprozess kann allerdings abhängig von der verwendeten Hardware einige Stunden dauern.

'timeout' setzt die maximale Ausführungszeit für einen Testoperator. Falls die Laufzeitmessung eines Operators den Timeout überschreitet, wird der Test des Operators abgebrochen. In diesem Fall werden keine Informationen über diesen Operator gespeichert. Der Timeout wird durch `GenParamValue` in Sekunden definiert. Der Wert *'infinite'* verhindert den Abbruch des Optimierungsprozesses eines Operators (Standard).

'split_level' beschränkt den Optimierungsprozess auf eine bestimmte Parallelisierungsmethode. Der entsprechende Wert in `GenParamValue` kann einen der folgenden Werte annehmen:

'split_domain' führt die Optimierung auf allen bildverarbeitenden Operatoren durch, die eine Datenparallelisierung auf Domainebene unterstützen.

'split_channel' führt die Optimierung auf allen bildverarbeitenden Operatoren durch, die eine Datenparallelisierung auf Kanalebene unterstützen.

'split_tuple' führt die Optimierung auf allen bildverarbeitenden Operatoren durch, die eine Datenparallelisierung auf Tupelebene unterstützen.

Achtung

Während der Tests muss `optimize_aop` jeden zu optimierenden Operator mehrere Male aufrufen. Abhängig von den Parameterwerten kann daher die Ausführzeit von `optimize_aop` sehr lang sein. Es ist daher für eine korrekte Optimierung essentiell, keine weiteren rechenintensiven Anwendungen gleichzeitig auf dem Rechner laufen zu lassen, da diese die Zeitmessungen für die Optimierung deutlich beeinflussen und zu falschen Ergebnissen führen würden. `optimize_aop` muss durch entsprechend autorisierte Benutzer aufgerufen werden, damit die gesammelten Informationen dauerhaft gespeichert werden können (Details hierzu finden sich in der obigen Operatorbeschreibung).

Parameter

▷ **`OperatorName`** (input_control) ..... string(-array) ⇝ *string*
Zu testende Operatoren.
Defaultwert: ''

▷ **`IconicType`** (input_control) ..... string(-array) ⇝ *string*
Zu testende Objekttypen.
Defaultwert: ''
Wertevorschläge: `IconicType` ∈ {'', 'byte', 'int1', 'int2', 'uint2', 'int4', 'int8', 'direction', 'cyclic', 'vector_field', 'complex', 'region', 'xld', 'xld_cont', 'xld_poly'}

▷ **`FileName`** (input_control) ..... filename.write ⇝ *string* / integer
Dateiname.
Defaultwert: ''

▷ **`GenParamName`** (input_control) ..... string-array ⇝ *string*
Parametername.
Defaultwert: 'none'
Wertevorschläge: `GenParamName` ∈ {'parameters', 'model', 'timeout', 'file_mode', 'system_mode', 'split_level'}

▷ **`GenParamValue`** (input_control) ..... string-array ⇝ *string* / real / integer
Parameterwert.
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 'none'
Wertevorschläge: `GenParamValue` ∈ {'true', 'renew', 'truncate', 'threshold', 'linear', 'mlp', -1.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_aop_knowledge` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

query_aop_info (: : OperatorName, IndexName, IndexValue : Name, Value)

Abfragen von Indizierungs-Informationen über HALCONs automatische Operator-Parallelisierung.

HALCON unterstützt einen Mechanismus, um das Verhalten der automatischen Parallelisierung von Operatoren (AOP) an Mehrprozessorarchitekturen anzupassen und Operatoren zu beschleunigen. Dazu können für jeden Operator, der sich auf Tupel-, Kanal- oder Domain-Ebene parallelisieren lässt, mit Hilfe von optimize_aop Abhängigkeiten des Laufzeitverhaltens von der Hardware und Eingabeparametern ermittelt werden und durch die Operatoren get_aop_info und set_aop_info teilweise gelesen oder verändert werden. Um diese operatorspezifischen Informationen adressieren zu können, sind die Daten in einer 3-stufigen Hierarchie indizierbar, die sich mit query_aop_info abfragen lässt. Die drei Indizierungsstufen setzen sich durch den Operatornamen, den ikonischen Bildtyp und optional einen Parameterstring zusammen, der einen speziellen Modus oder Methode des Operators spezifiziert.

Übergibt man in OperatorName einen leeren String '', werden die Parameter IndexName und IndexValue ignoriert und alle Operatornamen in Value zurückgegeben, für die AOP-Optimierungsdaten vorhanden sind. Name enthält für jeden Operatornamen die entsprechende Indexkennung *'operator'*. Wird ein einzelner Operatorname mit OperatorName spezifiziert und ein leerer String '' den Parametern IndexName und IndexValue übergeben, werden alle ikonischen Typen erfragt, für die der Operator Informationen vorhält. Value enthält den ikonischen Typ in Form eines Strings, Name die Indexkennung *'iconic_type'*. Schließlich können die Parameterkennungen für einen ikonischen Typ eines Operators erfragt werden, wenn OperatorName den Operatornamen, IndexName die Kennung *'iconic_type'* und IndexValue den bestimmten ikonischen Typ enthält. Name enthält dann die Parameterkennung, Value den entsprechenden Parameterwert des Operators, der eine spezielle Methode beschreibt Das Format des Strings der Parameterkennung setzt sich aus dem String *'parameter'* zusammen, gefolgt von einem Doppelpunkt-Trennzeichen *':'* und einer Zahl, die die Parameterposition unter den Eingabe-Kontrollparameter des Operators beschreibt (beginnend bei 1). Falls der Operator keinen Methode-Parameter unterstützt oder keine parameterspezifischen Informationen für den Operator mit optimize_aop ermittelt wurden, wird der String *'parameter:0'* für Name zurückgegeben, während Value einen leeren String enthält. Falls jedoch keine AOP-Informationen für die spezifizierte Kennung vorliegen, enthält Name einen leeren String.

Parameter

▷ **OperatorName** (input_control) ..... string ⇝ *string* / integer
Operator für AOP-Information.
Defaultwert: ''

▷ **IndexName** (input_control) ..... string(-array) ⇝ *string* / integer
Indizierungsname.
Defaultwert: ''
Wertevorschläge: IndexName ∈ {'', 'iconic_type'}

▷ **IndexValue** (input_control) ..... string(-array) ⇝ *string* / integer
Adressierungswert.
Parameteranzahl: IndexName == IndexValue
Defaultwert: ''
Wertevorschläge: IndexValue ∈ {'', 'byte', 'int1', 'int2', 'uint2', 'int4', 'int8', 'direction', 'cyclic', 'vector_field', 'complex', 'region', 'xld', 'xld_cont', 'xld_poly'}

▷ **Name** (output_control) ..... string-array ⇝ *string*
Nächster Indizierungsname.

▷ **Value** (output_control) ..... string-array ⇝ *string*
Zugehöriger Adressierungswert.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_aop_knowledge den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

read_aop_knowledge (: : FileName, GenParamName, GenParamValue : Attributes, OperatorNames)

Laden von Informationen über das hardwareabhängige Verhalten der automatischen Operator-Parallelisierung.

read_aop_knowledge wird verwendet, um die automatische Operator-Parallelisierung (AOP) der HALCON-Operatoren zu unterstützen. HALCON verwendet intern einen Parallelisierungsmechanismus, um Mehrprozessorarchitekturen besser zu nutzen und die Bearbeitungsdauer von Operatoren zu verkürzen. Um diese Parallelisierung automatisch, optimiert durchführen zu können, benötigt HALCON spezielles Wissen über die verwendete Hardware bzw. über deren Potential, die verschiedenen HALCON-Operatoren parallel zu verarbeiten. Dieses Wissen ist rechnerspezifisch und kann mit dem Operator optimize_aop und write_aop_knowledge trainiert und abgespeichert werden.

Mit read_aop_knowledge ist es möglich, dieses Wissen explizit aus einer Binärdatei zu laden. Dabei definiert FileName den Namen der Datei mit Pfad und Dateierweiterung. Falls ein leerer String '' übergeben wird, liest HALCON die Informationen aus der spezifischen Datei *'.aop_info'* im HALCON Installationsverzeichnis (Linux/macOS) oder aus dem gemeinsamen Datenverzeichnis für Anwendungen (Windows). Diese Datei wird auch von HALCON automatisch während der Initialisierungsphase gelesen, kann aber mit read_aop_knowledge leicht wieder eingelesen werden und so das gespeicherte Wissen gegebenenfalls wieder restaurieren. Die Art und Weise wie das gelesene Wissen die vorhandene Systeminformation aktualisiert, kann mit Hilfe des Parameterpaares GenParamName und GenParamValue gesteuert werden. Dabei muss jeder Wert in GenParamName einen korrespondierenden Definitionswert in GenParamValue haben. D.h. die Länge der Parameterarrays muss gleich groß sein.

'ignore_attrib' für GenParamName ermöglicht es, das Überprüfen der Wissensattribute zu beeinflussen. Beispielsweise überprüft read_aop_knowledge während des Lesens, ob der Inhalt der Datei den aktuell verwendeten Computer, die HALCON-Version, -Revision und -Architektur betrifft. Die entsprechenden Attribute können mit Hilfe der folgenden Werte für GenParamValue unterdrückt werden:

- *'host'* der Computername des aktuellen Rechners
- *'cpu_info'* die Rechnertopologie
- *'variant'* die HALCON-Variante
- *'version'* die HALCON-Version
- *'revision'* die HALCON-Revision
- *'architecture'* die HALCON Architektur

Dabei können mehrere, unterschiedliche Attribute unterdrückt werden.

'mode' setzt die Art und Weise, wie die System-Informationen aktualisiert werden.

- *'truncate'* für GenParamValue löscht alle existierenden Informationen bevor das Wissen aus der Datei hinzugefügt wird.
- *'renew'* überschreibt existierendes Wissen und fügt neues hinzu (Standard).
- *'append'* behält das existierende Wissen und fügt nur das Wissen hinzu, welches noch nicht enthalten ist.
- *'nil'* verweigert jegliche Aktualisierung gibt aber die Wissensattribute in Attributes und alle in der Datei enthaltenen Operatoren in OperatorNames.

'operator' für GenParamName legt fest, dass der korrespondierende Index von GenParamValue einen Namen eines Operators enthält, dessen spezifisches AOP-Wissen aus der Datei gelesen und im System aktualisiert werden soll. Mehrfache Definitionen von Operatoren sind möglich. Standardmäßig werden die Informationen zu allen, in der Datei enthalten Operatoren gelesen.

'iconic_type' beschränkt sich auf AOP-Information eines bestimmten ikonischen Typs. Folgende Typen können mit GenParamValue spezifiziert werden: *'byte'*,*'uint2'*,*'real'*,*'int1'*, *'int2'*,*'int4'*,*'int8'*,*'direction'*, *'vector_field'*,*'cyclic'*, *'complex'*,*'region'*,*'xld'*. Mehrfache Definitionen von ikonischen Typen sind möglich.

Die Namen der aktualisierten Operatoren werden in OperatorNames zurückgegeben, die Attribute der enthaltenen Information in Attributes. Letzteres Tupel kodiert folgende Bedeutungen mit aufsteigendem Index, beginnend mit 0: Rechnername, HALCON-Variante, HALCON-Version, HALCON-Revision, HALCON-Architektur.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Name der Informationsdatei.
Defaultwert: ”

▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
Filtert die zu ladende Information.
Defaultwert: ’none’
Wertevorschläge: `GenParamName` ∈ {’none’, ’ignore_attrib’, ’operator’, ’iconic_type’, ’mode’}

▷ **GenParamValue** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
Spezifiziert den Filter.
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: ’none’
Wertevorschläge: `GenParamValue` ∈ {’none’, ’host_name’, ’variant’, ’architecture’, ’version’, ’revision’, ’byte’, ’int1’, ’int2’, ’uint2’, ’int4’, ’int8’, ’direction’, ’cyclic’, ’vector_field’, ’complex’, ’region’, ’xld’, ’xld_cont’, ’xld_poly’, ’nil’, ’truncate’, ’replace’, ’renew’, ’append’}

▷ **Attributes** (output_control) . . . . . string-array ⇝ *string*
Attribute der geladenen Information.

▷ **OperatorNames** (output_control) . . . . . string(-array) ⇝ *string*
Gelesene Operatoren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_aop_knowledge` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. HALCON verwendet intern einen Parallelisierungsmechanismus, um Mehrprozessorarchitekturen besser zu nutzen und die Bearbeitungsdauer von Operatoren zu verkürzen. Um diese Parallelisierung automatisch, optimiert durchführen zu können, benötigt HALCON spezielles Wissen über die verwendete Hardware bzw. über deren Potential, die verschiedenen HALCON-Operatoren parallel zu verarbeiten. Dieses Wissen ist rechnerspezifisch und kann man mit Hilfe der Operatoren `optimize_aop` und `read_aop_knowledge` erhalten.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

set_aop_info (: : OperatorName, IndexName, IndexValue, InfoName, InfoValue :)

Verändert Information über HALCONs automatische Operator-Parallelisierung.

HALCON unterstützt einen Mechanismus, um das Verhalten der automatischen Parallelisierung von Operatoren (AOP) an Mehrprozessorarchitekturen anzupassen und Operatoren zu beschleunigen. Dazu können für jeden Operator, der sich auf Tupel-, Kanal- oder Domain-Ebene parallelisieren lässt, mit Hilfe von `optimize_aop` Abhängigkeiten des Laufzeitverhaltens von der Hardware und Eingabeparametern ermittelt werden. Der Operator `get_aop_info` liest Elemente dieser Informationen, während diese durch den Operator `set_aop_info` verändert werden können.

Um die operatorspezifischen Informationen adressieren zu können, sind die Daten in einer 3-stufigen Hierarchie indizierbar, die sich mit `query_aop_info` abfragen lässt. Durch ein Tripel von Indexkennungen lässt sich so jeweils ein AOP-Set an Optimierungsdaten indizieren. Die drei Indizierungsstufen setzen sich durch den Operatornamen, den ikonischen Bildtyp und optional einen Parameterstring zusammen, der einen speziellen Modus oder Methode des Operators spezifiziert. Während der Operatorname `OperatorName` übergeben wird, definieren `IndexName` und `IndexValue` die letzteren beiden Indizes. `IndexName` erhält dabei die Indexkennung, `IndexValue` den Indexwert, wie sie von `query_aop_info` zurückgegeben werden. Es ist möglich mehrere Indexkennungen der gleichen Stufe anzugeben, um die spezifizierte Information bei mehreren AOP-Sets zu

speichern. Die gesamte Indexstufe kann adressiert werden, wenn eine Indexkennung ausgelassen oder für einen Indexwert ein leerer String übergeben wird. Beispielsweise werden durch eine leeren String für `OperatorName`, dem Tupel *['iconic_type','iconic_type']* für `IndexName` und dem Tupel *['byte','uint2']* für `IndexValue` alle Operatoren adressiert, die Byte oder UINT2 Bilder verarbeiten.

`set_aop_info` setzt die in `InfoValue` übergebenen Informationswerte in dem spezifizierten AOP-Sets. Die Art der Information kann mit dem Parameter `InfoName` mit einem der folgenden Werten spezifiziert werden:

'max_threads' setzt die größte zulässige Threadanzahl, mit der der Operator mit dem entsprechenden ikonischen Datentyp und Parameterwert parallelisiert werden darf.

'split_level' setzt die erlaubten Parallelisierungsebenen für die indizierten AOP-Sets. Mögliche Werte sind *'split_tuple'*, *'split_channel'*, *'split_domain'* und *'split_partial'*.

Achtung

`set_aop_info` verändert die AOP-Systemwissensbasis. Diese Änderungen werden ebenfalls durch `write_aop_knowledge` gespeichert. Es wird empfohlen eine Backup-Datei zu halten bevor das Systemwissen geändert wird.

Parameter

- ▷ **`OperatorName`** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Operatorname.
  Defaultwert: ''
- ▷ **`IndexName`** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Ikonischer Objekttyp.
  Defaultwert: ''
  Wertevorschläge: `IndexName` ∈ {'iconic_type', ['iconic_type','parameter:0']}
- ▷ **`IndexValue`** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Parameter-Typ.
  Parameteranzahl: IndexName == IndexValue
  Defaultwert: ''
  Wertevorschläge: `IndexValue` ∈ {'byte', ['uint2','']}
- ▷ **`InfoName`** (input_control) . . . . . string ⇝ *string*
  Art der Information.
  Defaultwert: 'max_threads'
  Wertevorschläge: `InfoName` ∈ {'max_threads', 'model'}
- ▷ **`InfoValue`** (input_control) . . . . . integer(-array) ⇝ *integer*
  AOP-Information.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_aop_knowledge` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

`write_aop_knowledge ( : : FileName, GenParamName, GenParamValue : )`

Speichern von Informationen über das hardwareabhängige Verhalten der automatischen Operator-Parallelisierung.

`write_aop_knowledge` speichert das operatorspezifische Wissen zur automatischen Operator-Parallelisierung (AOP) in einer Datei ab. Diese Wissen wird vom HALCON-System verwaltet und genutzt, um Mehrprozessorarchitekturen besser nutzen und die Operatoren zu beschleunigen. Es beschreibt für jeden Operator, der sich auf

Tupel-, Kanal- oder auf Domain-Ebene parallelisieren lässt, das Laufzeitverhalten in Abhängigkeit der Hardware. Dieses Hardware spezifische Wissen kann mit Hilfe des Operators `optimize_aop` ermittelt oder von einer Datei mit `read_aop_knowledge` gelesen werden.

`write_aop_knowledge` ermöglicht es, dieses Wissen explizit in einer Binär-Datei abzuspeichern. Hierfür wird in `FileName` der Name der Datei übergeben (inkl. des Pfades und der Dateierweiterung). Falls `FileName` einen leerer String '' enthält, schreibt HALCON die AOP-Informationen in die Datei *'.aop_info'* im HALCON Installationsverzeichnis (Linux/macOS) oder im gemeinsame Datenverzeichnis für Applikationen (Windows). Diese spezielle Datei wird automatisch während der Initialisierungsphase des ersten HALCON Operators geladen. `write_aop_knowledge` fügt dem gespeicherten Wissen Attribute wie den Computernamen und Informationen über die HALCON-Architektur, -Variante, -Version und -Revision hinzu. Das ermöglicht es HALCON, AOP-Informationen von verschiedenen Systemen in dieselbe Datei zu schreiben. Zu diesem Zweck kann `write_aop_knowledge` die Aktualisierung des Dateiinhalts mit dem Parameterpaar `GenParamName` und `GenParamValue` kontrollieren. `GenParamName` definiert dabei die Bedeutung, `GenParamValue` den Wert der gewünschten Aktion. Dabei muss jeder Eintrag in `GenParamName` jeweils einen korrespondierenden Eintrag in `GenParamValue` haben, d.h. dass die Länge der Parametertupel gleich lang sein muss. Im Folgenden sind die möglichen Werte für `GenParamName` gelistet und die jeweils möglichen Werte für `GenParamValue` beschrieben:

'mode' setzt die Art, wie das bestehende Wissen mit den gleichen Attributen in der Datei aktualisiert wird.

- *'truncate'* für `GenParamValue` löscht das bestehende Wissen mit passenden Attributen, bevor die neuen Informationen der Datei zugefügt werden.
- *'renew'* überschreibt bestehendes Wissen und fügt neues hinzu (Standard).
- *'append'* behält alle existierenden Operatorinformationen und fügt nur noch nicht enthaltenes Wissen hinzu.
- *'nil'* unterdrückt jegliche Aktualisierung der Datei, sondern überprüft nur die Datei hinsichtlich ausreichender Benutzerrechte für einen Schreib- und Lesezugriff.

'operator' weist darauf hin, dass der korrespondierende Wert von `GenParamValue` einen Operatornamen enthält. Damit lassen sich die Operatoren eingrenzen, für welche entsprechendes AOP-Wissen abgespeichert werden soll. Mehrere Operatoren können definiert. Standardmäßig wird jede verfügbare Information aller Operatoren geschrieben.

'iconic_type' schränkt die zu speichernde Information auf einen bestimmten ikonischen Typ ein. Folgende Werte werden für `GenParamValue` unterstützt: *'byte'*, *'uint2'*, *'real'*, *'int1'*, *'int2'*, *'int4'*, *'int8'*, *'direction'*, *'vector_field'*, *'cyclic'*, *'vector_field'*, *'complex'*, *'region'*, *'xld'*. Es können mehrere ikonische Typen festgelegt werden.

Parameter

▷ **`FileName`** (input_control) . . . . . filename.write ⇝ *string*
Name der Informationsdatei
Defaultwert: ''

▷ **`GenParamName`** (input_control) . . . . . string(-array) ⇝ *string*
Filtert die zu schreibende Information
Defaultwert: 'none'
Wertevorschläge: `GenParamName` ∈ {'mode', 'operator', 'iconic_type'}

▷ **`GenParamValue`** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
Spezifiziert den Filter
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 'none'
Wertevorschläge: `GenParamValue` ∈ {'truncate', 'renew', 'append'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_aop_knowledge` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: exclusive (läuft parallel nur zu unabhängigen Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

25.9 Parameter

get_system (: : Query : Information)

Abfragen von Informationen über die aktuellen HALCON-Systemparameter.

get_system liefert Informationen über die aktuellen HALCON-Systemparameter zurück.

Lesbare- und schreibbare Parameter

Der Operator get_system liefert Informationen über alle Parameter, die mit set_system gesetzt wurden zurück. Weitere Informationen über diese Parameter können in der Dokumentation zu set_system nachgelesen werden.

Nur lesbare Parameter

Folgende Systemparameter können abgefragt werden:

- **Versionen:**
  'halcon_xl': Verwendete Variante von HALCON: HALCON XL, das Bilder größer als 32768 × 32768 erlaubt (*'true'*) oder HALCON, bei dem die maximale Bildgröße auf 32768 × 32768 beschränkt ist (*'false'*).
  'version': Versionsnummer von HALCON, z.B.: 11.0.
  'edition': Verwendete HALCON Edition. Der Rückgabewert kann die Strings *'progress'* oder *'steady'* annehmen.
  'last_update': Datum der Erzeugung der HALCON-Bibliothek.
  'revision': Revisionsnummer von HALCON, z.B.: 1.
  'file_version': Dateiversionsnummer der HALCON Bibliothek. Sie setzt sich aus der HALCON Versionsnummer, der Revisionsnummer und der Buildnummer zusammen, (z.B.: 11.0.1.2). Der Versionsnummer anhängende Nullen entfallen (z.B.: 11.0.1 für Buildnummer 0 oder 11.0 für Revisionsnummer 0 und Buildnummer 0).
- **Obergrenzen:**
  'max_inp_obj_par': Maximale Anzahl von Eingabeparametern.
  'max_outp_obj_par': Maximale Anzahl von Ausgabeparametern.
  'max_inp_ctrl_par': Maximale Anzahl von Eingabe-Steuerparametern.
  'max_outp_ctrl_par': Maximale Anzahl von Ausgabe-Steuerparametern.
  'max_window': Maximale Anzahl von Fenstern.
- **Grafik:**
  'window_name': Liefert den Namen, der eventuell mit set_window_attr('window_title',...) gesetzt wurde zurück. Falls kein Name gesetzt wurde, wird *'default'* zurückgeliefert.
- **Parallelisierung:**
  'processor_num': Gibt die Anzahl der Prozessoren an, die in der verwendeten Hardware gefunden wurden.
  'tsp_used_thread_num': Gibt die Anzahl der Threads zurück, die der letzte Operator für die automatische Parallelisierung verwendet hat. Dabei werden nur Operatoren berücksichtigt, die von dem Thread aufgerufen wurden und die automatische Parallelisierung unterstützen. Es ist zu beachten, dass dieser Parameter in HDevelop nur im Ausführmodus, nicht im schrittweisen Modus funktioniert.
  'tsp_used_split_levels' Gibt alle Methoden der Parallelisierung zurück, die der letzter Operator für die automatische Parallelisierung verwendet hat. Dabei werden nur Operatoren berücksichtigt, die von dem Thread aufgerufen wurden und die automatische Parallelisierung unterstützen. Falls der letzte Operator nicht parallelisiert wurde, wird der String *'false'* zurückgegeben. Es ist zu beachten, dass dieser Parameter in HDevelop nur im Ausführmodus, nicht im schrittweisen Modus funktioniert.
- **Sonstiges:**
  'hostids': Die Host-IDs des Rechners, die zur Lizenzierung von HALCON verwendet werden können.
  'num_proc': Gesamtzahl der verfügbaren HALCON-Prozeduren (*'num_sys_proc'* + *'num_user_proc'*).
  'num_sys_proc': Anzahl der Systemprozeduren (unterstützte Prozeduren).
  'num_user_proc': Anzahl der benutzerdefinierten Prozeduren (siehe „Extension Package Programmer's Manual").

'byte_order': Byte-Order des Prozessors (*'msb_first'* oder *'lsb_first'*).

'operating_system': Name des Betriebssystems des Rechners, auf dem der HALCON-Prozess ausgeführt wird. Beachten Sie, dass für alle Windows-Versionen 'Windows NT' zurückgegeben wird. Der Parameter *'operating_system_version'* ermöglicht es, die genaue Version abzufragen.

'operating_system_version': Versionsnummer des Betriebssystems des Rechners, auf dem der HALCON-Prozess ausgeführt wird. Die Windows-Versionsnummern können mit Hilfe der Informationen zu Betriebssystemen und Versionsnummern auf der Microsoft-Webseite entschlüsselt werden.

'hostname': Name des Rechners, auf dem der HALCON-Prozess ausgeführt wird. Wenn der Rechner teil eines Windows Clusters ist, wird der Name des Clusters zurückgegeben, nicht der Name des lokalen Rechners. Kann der Rechnername nicht bestimmt werden, wird ein leerer String zurückgegeben.

'locale_raw': In der Laufzeitumgebung verwendete Locale-Einstellung, die für die Kodierung von Systemstrings verwendet wird (entsprechend der Locale-Kategorie LC_CTYPE). Das Ergebnis entspricht direkt dem Aufruf der System-C-Funktion setlocale, bzw. dem Aufruf von locale in der Linux-Shell. Es enthält i.d.R. Informationen zu Sprache, Land und Codepage bzw. Codeset - unter Windows beispielsweise *'German_Germany.1252'* oder unter Linux *'en_US.utf8'*. Dabei ist zu beachten, dass unter Linux und Mac der Name nicht eindeutig ist, da der Codeset-Name von System und Konfiguration abhängt und beispielsweise 'utf8' oder 'UTF-8' sein oder auch ganz fehlen kann. Auf Systemen ohne Locale-Unterstützung wird ein leerer String zurückgegeben.

'locale_codeset': Von der Laufzeitumgebung verwendetes Codeset, das für die Kodierung von Systemstrings verwendet wird (entsprechend der Locale-Kategorie LC_CTYPE). Im Gegensatz zu *'locale_raw'* enthält das Ergebnis nur die Codepage bzw. das Codeset und nicht die Sprache oder das Land. Dafür liegt das Ergebnis in normalisierter Form vor, so dass es leichter für Tests ausgewertet werden kann. Das Ergebnis enthält nur Kleinbuchstaben und Zahlen, wobei für Windows Codepages und Linux/Mac Codesets unterschiedliche Ergebnisse geliefert werden, da die verwendeten Codesets i.d.R. nicht identisch sind. Typische Ergebnisse unter Windows sind z.B. *'cp1252'* (für die in der westlichen Welt verwendete Codepage 1252 - weitestgehend Latin-1) oder *'cp932'* (für die in Japan verwendete Codepage 932 - ShiftJIS). Typische Ergebnisse unter Linux und Mac sind *'utf8'* (entsprechend dem Default-Locale unter Mac und den meisten aktuellen Linux-Distributionen), *'iso885915'* (Latin-9), *'shiftjis'* und *'ascii'* (für ANSI_X3.4-1968, US-ASCII, POSIX, C).

'halcon_64': Flag, das gesetzt ist, wenn es sich um eine 64-Bit-HALCON-Version handelt (*'true'*) oder nicht, wenn es sich um eine 32 Bit Version handelt (*'false'*).

'halcon_arch': Name der HALCON-Architektur des laufenden HALCON-Prozesses.

'library_fullname': Pfad zur aktuell geladenen HALCON-Bibliothek einschließlich des Dateinamens.

'temp_mem': Bedarf an temporärem Speicher des zuletzt ausgeführten Operators in Byte. Der Rückgabewert ist nur definiert, wenn vor dem zu vermessenden Operator `set_check('memory')` aufgerufen wurde.

'mmx_supported': Flag, das gesetzt ist, wenn der Prozessor MMX-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'sse_supported': Flag, das gesetzt ist, wenn der Prozessor SSE-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'sse2_supported': Flag, das gesetzt ist, wenn der Prozessor SSE2-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'sse3_supported': Flag, das gesetzt ist, wenn der Prozessor SSE3-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'ssse3_supported': Flag, das gesetzt ist, wenn der Prozessor SSSE3-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'sse41_supported': Flag, das gesetzt ist, wenn der Prozessor SSE41-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'sse42_supported': Flag, das gesetzt ist, wenn der Prozessor SSE42-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx_supported': Flag, das gesetzt ist, wenn der Prozessor AVX-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx2_supported': Flag, das gesetzt ist, wenn der Prozessor AVX2-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512f_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512F-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512dq_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512DQ-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512bw_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512BW-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512er_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512ER-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512cd_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512CD-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512vl_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512VL-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512vbmi_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512VBMI-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512ifma_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512IFMA-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512vbmi2_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512VBMI2-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512vpopcntdq_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512VPOPCNTDQ-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512bitalg_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512BITALG-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'avx512vnni_supported': Flag, das gesetzt ist, wenn der Prozessor AVX512VNNI-Operationen unterstützt (*'true'*). Andernfalls ist es nicht gesetzt (*'false'*).

'opengl_hidden_surface_removal_available': Flag, das gesetzt ist, wenn die Grafikkarte die Beschleunigung der Verdeckungsberechnung in den Operatoren `create_shape_model_3d`, `find_shape_model_3d`, `project_shape_model_3d` und `project_object_model_3d` unterstützt. Mindestvoraussetzungen sind OpenGL 2.0 und die Erweiterungen GL_EXT_framebuffer_object und GL_ARB_texture_float. Es ist nicht möglich diese Beschleunigung über den Windows Remote Desktop oder X11 Weiterleitung zu verwenden.

'alloctmp_max_used' Die maximal verwendete Menge temporären Speichers eines Threads in Bytes. Das erste Element des Tupels enthält den Wert für den aktuellen Thread, während die übrigen Elemente die Werte von derzeit ungenutzten Threads aus dem Threadpool enthalten.
Das Maximum wird zurückgesetzt, wenn der Cachemodus für temporären Speicher auf *'idle'* gesetzt wird.

'tsp_temporary_mem_cache_block_sizes' Tupel, das die Größen aller temporären Speicherblöcke in Bytes enthält, die momentan vom Thread gecached werden.

'temporary_mem_reservoir_block_sizes' Tupel, das die Größen aller temporären Speicherblöcke in Bytes enthält, die momentan im globalen temporären Speicher-Reservoir gecached sind.

'legacy_handle_mode', *'tsp_legacy_handle_mode'*: Gibt zurück, ob der Legacy Handle Modus aktiviert ist.

- **CUDA Unterstützung für Deep Learning:**

'cuda_loaded': Gibt *'true'* zurück, wenn die CUDA-Bibliothek geladen werden konnte.

'cuda_version': Gibt die Version der CUDA-Bibliothek zurück. Wenn die CUDA-Bibliothek nicht geladen werden konnte, wird *-1* zurückgegeben.

'cuda_devices': Gibt die Namen der verfügbaren Grafikkarten, die CUDA unterstützen, zurück.

'cudnn_loaded': Gibt *'true'* zurück, wenn die cuDNN-Bibliothek geladen werden konnte.

'cudnn_version': Gibt die Version der cuDNN-Bibliothek zurück. Wenn die cuDNN-Bibliothek nicht geladen werden konnte, wird *-1* zurückgegeben.

'cublas_loaded': Gibt *'true'* zurück, wenn die cuBLAS-Bibliothek geladen werden konnte.

'cublas_version': Gibt die Version der cuBLAS Bibliothek zurück. Wenn die cuBLAS-Bibliothek nicht geladen werden konnte, wird *-1* zurückgegeben.

- **Arm Compute Library:**

'arm_compute_loaded': Gibt *'true'* zurück, wenn die Arm Compute Library geladen werden konnte, *'false'* andernfalls.

'arm_compute_version': Gibt die Version der Arm Compute Library zurück. Wenn die Bibliothek nicht geladen werden konnte, wird ein leerer String zurückgegeben.

Parameter

▷ **Query** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Gewünschter Systemparameter.
Defaultwert: 'init_new_image'
Werteliste: `Query` ∈ {'3d_model_dir', 'alloctmp_max_blocksize', 'alloctmp_min_blocksize', 'alloctmp_max_used', 'available_license_files', 'available_parameters', 'avx_enable', 'avx_supported', 'avx2_enable', 'avx2_supported', 'avx512f_enable', 'avx512f_supported', 'avx512dq_enable', 'avx512dq_supported', 'avx512bw_enable', 'avx512bw_supported', 'avx512er_enable', 'avx512er_supported', 'avx512cd_enable', 'avx512cd_supported', 'avx512vl_enable', 'avx512vl_supported', 'avx512vbmi_enable', 'avx512vbmi_supported', 'avx512ifma_enable', 'avx512ifma_supported', 'avx512vbmi2_enable', 'avx512vbmi2_supported', 'avx512vpopcntdq_enable', 'avx512vpopcntdq_supported', 'avx512bitalg_enable', 'avx512bitalg_supported', 'avx512vnni_enable', 'avx512vnni_supported', 'backing_store', 'border_shape_models', 'bundle_version', 'byte_order', 'calib_dir', 'cancel_draw_result', 'clip_region', 'clock_mode', 'cuda_loaded', 'cuda_version', 'cuda_devices', 'cudnn_loaded', 'cudnn_version', 'cublas_loaded', 'cublas_version', 'current_license_info', 'current_runlength_number', 'database', 'default_font', 'disabled_operators', 'dl_dir', 'do_low_error', 'edition', 'empty_region_result', 'example_dir', 'expiration_date', 'extern_alloc_funct', 'extern_free_funct', 'failed_license_rechecks', 'filename_encoding', 'file_version', 'filter_dir', 'flush_file', 'flush_graphic', 'global_mem_cache', 'halcon_arch', 'halcon_dir', 'halcon_xl', 'halcon_64', 'height', 'help_dir', 'hthread_id', 'hostids', 'hostname', 'icon_name', 'image_cache_capacity', 'image_dir', 'image_dpi', 'init_new_image', 'int2_bits', 'int_zooming', 'is_license_valid', 'language', 'last_update', 'legacy_handle_mode', 'library_fullname', 'licensed_hostid', 'licensed_modules', 'licensed_version', 'locale_codeset', 'locale_raw', 'lut_dir', 'max_connection', 'max_inp_ctrl_par', 'max_inp_obj_par', 'max_outp_ctrl_par', 'max_outp_obj_par', 'max_window', 'mmx_enable', 'mmx_supported', 'neighborhood', 'no_object_result', 'num_proc', 'num_sys_proc', 'num_user_proc', 'ocr_dir', 'ocr_trainf_version', 'opengl_hidden_surface_removal_available', 'opengl_hidden_surface_removal_enable', 'opengl_compatibility_mode_enable', 'opengl_context_cache_enable', 'operating_system', 'operating_system_version', 'parallelize_operators', 'pregenerate_shape_models', 'processor_num', 'product_edition', 'read_halcon_files_encoding_fallback', 'reentrant', 'revision', 'seed_rand', 'sse_enable', 'sse_supported', 'sse2_enable', 'sse2_supported', 'sse3_enable', 'sse3_supported', 'ssse3_enable', 'ssse3_supported', 'sse41_enable', 'sse41_supported', 'sse42_enable', 'sse42_supported', 'store_empty_region', 'temp_mem', 'temporary_mem_cache', 'temporary_mem_reservoir', 'temporary_mem_reservoir_size', 'thread_num', 'thread_pool', 'timer_mode', 'tuple_string_operator_mode', 'tsp_cancel_draw_result', 'tsp_clip_region', 'tsp_current_runlength_number', 'tsp_empty_region_result', 'tsp_height', 'tsp_init_new_image', 'tsp_legacy_handle_mode', 'tsp_neighborhood', 'tsp_no_object_result', 'tsp_store_empty_region', 'tsp_temporary_mem_cache', 'tsp_temporary_mem_cache_block_sizes', 'tsp_temporary_mem_reservoir', 'tsp_thread_num', 'tsp_tuple_string_operator_mode', 'tsp_used_split_levels', 'tsp_used_thread_num', 'tsp_width', 'unlicensed_operators', 'update_lut', 'use_window_thread', 'version', 'width', 'window_name', 'write_halcon_files_encoding', 'x_package'}

▷ **Information** (output_control) . . . . . attribute.value(-array) ⇝ *integer* / real / string
Aktueller Wert des Systemparameters.

Ergebnis

`get_system` liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`reset_obj_db`

Nachfolger

`set_system`

Siehe auch

`set_system`

Modul

Foundation

get_system_info (: : Query : Information)

Abfragen von Informationen über die aktuellen HALCON-Systemparameter, ohne eine gültige HALCON-Lizenz zu benötigen.

get_system_info liefert Informationen über die aktuellen HALCON-Systemparameter zurück. Anders als bei dem sehr ähnlichen get_system Operator wird hierfür keine gültige HALCON-Lizenz benötigt.

get_system_info unterstützt eine Untermenge der von get_system unterstützen Parameter. Die verfügbaren Parameter werden durch den Aufruf von get_system_info mit dem Parameter *'available_parameters'* zurück gegeben. Weitere Informationen über diese Parameter können in der Dokumentation zu get_system und set_system nachgelesen werden.

Parameter

▷ **Query** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Gewünschter Systemparameter.
Defaultwert: 'available_parameters'
Werteliste: Query ∈ {'3d_model_dir', 'available_license_files', 'available_parameters', 'bundle_version', 'byte_order', 'current_license_info', 'disabled_operators', 'edition', 'example_dir', 'expiration_date', 'failed_license_rechecks', 'file_version', 'halcon_64', 'halcon_arch', 'halcon_dir', 'halcon_xl', 'help_dir', 'hostids', 'hostname', 'image_dir', 'is_license_valid', 'language', 'last_update', 'library_fullname', 'licensed_hostid', 'licensed_modules', 'licensed_version', 'operating_system', 'operating_system_version', 'parallel_halcon', 'product_edition', 'revision', 'unlicensed_operators', 'version'}

▷ **Information** (output_control) . . . . . attribute.value(-array) ⇝ *integer* / real / string
Aktueller Wert des Systemparameters.

Ergebnis

get_system_info liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

get_system

Modul

none

set_operator_timeout (: : OperatorName, Timeout, Mode :)

Setzt einen Timeout für einen Operator.

set_operator_timeout setzt den Timeout Timeout für alle zukünftigen Aufrufe des Operators OperatorName. Der Timeout wird in Sekunden angegeben. Ein Wert von *0* oder *'off'* löscht einen evtl. vorher gesetzten Timeout.

Der Timeout gilt nur für den aktuellen Thread und wird nicht an andere Threads weitergegeben.

Es werden zwei Arten von Timeout unterstützt, zwischen denen mittels Mode ausgewählt werden kann:

'cancel' Bricht die Ausführung des Operators ab, nachdem der Timeout abgelaufen ist. Der Operator liefert in diesem Fall den Fehlercode H_ERR_TIMEOUT (9400) zurück. Evlt. bereits vorhandene Ergebnisse des Operators werden verworfen.

'break' Hält die Ausführung des Operators an, nachdem der Timeout abgelaufen ist, und liefert, soweit möglich, alle bereits vorhandenen Ergebnisse zurück. Der Operator wird in diesem Fall normal beendet und liefert keinen Fehlercode zurück.

Achtung

Nicht alle Operatoren unterstützen Timeouts. Wenn ein Operator Timeouts unterstützt, ist dies im Abschnitt über Ausführungsinformationen in dessen Referenzdokumentation vermerkt.

Es gibt keine Garantie über die Granularität der Timeouts. Diese kann vom Operator, dessen Eingabedaten und dem ausführenden System abhängen. Üblicherweise ist die Granularität besser als 10 ms.

Parameter

- ▷ **`OperatorName`** (input_control) . . . . . string(-array) ⇝ *string*
  Operator für den der Timeout gesetzt werden soll.
- ▷ **`Timeout`** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Timeout in Sekunden.
  Defaultwert: 1
  Wertevorschläge: `Timeout` ∈ {1, 0.1, 0.5, 'off', 0}
- ▷ **`Mode`** (input_control) . . . . . string ⇝ *string*
  Art des zu setzenden Timeouts.
  Defaultwert: 'cancel'
  Wertevorschläge: `Mode` ∈ {'cancel', 'break'}

Ergebnis

Sind alle Parameterwerte korrekt, gibt `set_operator_timeout` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: local (kann nur von dem Thread aufgerufen werden, in dem die Fenster-, Modell- oder Tool-Instanz erzeugt wurde).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`interrupt_operator`

Siehe auch

`interrupt_operator`, `get_current_hthread_id`

Modul

Foundation

`set_system ( : : SystemParameter, Value : )`

Verändern von HALCON-Systemparametern.

`set_system` ermöglicht es, verschiedene Systemparameter zu verändern.

Parallelisierungsinformation: Einige Parameter werden exklusiv gesetzt, d.h., sie blockieren andere Threads bis kein weiterer Operator mehr auf die HALCON Bibliothek zugreift.

- Parameter, die mit *) markiert sind, gelten global und werden exklusiv ausgeführt.
- Parameter ohne *) und ohne *"tsp_"* gelten ebenfalls global, werden jedoch nicht exklusiv ausgeführt, d.h., der Operatoraufruf blockiert keine anderen Operatoraufrufe.
- Parameter mit "tsp_"-Variante sind Thread-spezifisch.
  1. Werden sie ohne *"tsp_"* aufgerufen, dann gelten sie für alle Threads, die nach dem Aufruf gestartet werden und im aktuellen Thread. Bereits gestartete Threads werden nicht beeinflusst.
  2. Parameter, die mit *"tsp_"* aufgerufen werden, gelten nur für den aktuellen Thread. Andere bereits laufende Threads werden nicht beeinflusst.

Mögliche Systemparameter:

- **Grafik:**

  'flush_graphic': Nach jeder HALCON-Operation, die eine Grafik-Ausgabe erzeugt, wird eine Flush-Operation ausgeführt, um die Daten sofort sichtbar zu machen. Bei einigen Programmen ist dies jedoch nicht nötig (z.B. wenn immer mit der Maus gearbeitet wird). In diesen Fällen kann *'flush_graphic'* auf *'false'* gesetzt werden, um die Laufzeit zu verbessern. Window-Manager von unixartigen Systemen führen eine automatische Flush-Operation durch, weshalb der Parameter bei diesen Betriebssystemen keine Auswirkungen hat.
  Seit HALCON 13 ist es nicht mehr empfehlenswert, *'flush_graphic'* zu verwenden, da dies zu unerwarteten Ergebnissen führen kann. Stattdessen sollte mit `set_window_param` der Parameter *'flush'* auf *'false'* gesetzt werden und zum Übertragen des Buffers auf das Grafikfenster `flush_buffer` aufgerufen werden.
  `Value`: *'true'* oder *'false'*
  Default: 'true'

  'int2_bits': Anzahl der signifikanten Bits bei int2 Bildern. Diese Anzahl wird für die Darstellung von int2 Bilder verwendet. Wenn der Wert -1 ist, werden die Grauwerte automatisch maximal skaliert (default).
  `Value`: *-1* oder *9..16*
  Default: -1

  'backing_store': Legt fest, ob der Inhalt von Fenstern, falls diese sich überlappen, automatisch neu geladen wird. Einige Implementierungen von X-Windows sind fehlerhaft; um diesen Fehler zu umgehen, kann das Sichern des Inhalts abgeschaltet werden. Es kann in einigen Fällen sinnvoll sein, den Sicherungsmechanismus auszuschalten, wenn es auf Laufzeit / Speicherplatz ankommt.
  `Value`: *'true'* oder *'false'*
  Default: 'true'

  'icon_name': Name von ikonifizierten Grafikfenstern unter X-Windows. Standardmäßig wird die Nummer des Grafikfensters angezeigt.
  `Value`: Name des ikonifizierten Grafikfensters.
  Default: 'default'

  'default_font' *): Default-Font für die Textausgabe in Text- oder Grafikfenstern. Verfügbare Fonts können mit `query_font` abgefragt werden. Die Syntax für Fontnamen in `Value` ist bei `set_font` beschrieben.
  `Value`: Name des Fonts.
  Default: 'fixed'

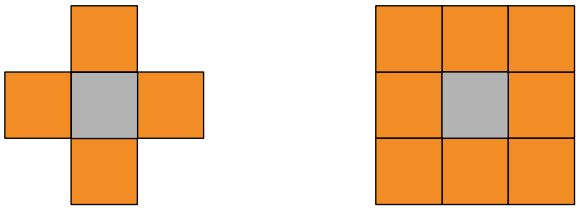
  'update_lut': Legt fest, ob die HALCON-Farbtabellen an die Umgebung angepasst werden. Solche Hardware-abhängigen Farbtabellen werden nicht mehr verwendet und damit ist der Parameter obsolet. Mit dem Operator `query_lut` können die möglichen Farbtabellen abgefragt werden, von denen eine spezifische mit dem Operator `set_lut` gesetzt werden kann.
  `Value`: *'true'* oder *'false'*
  Default: 'false'

  'use_window_thread': Wenn auf einem Microsoft Windows System einer der HALCON Grafikoperatoren aus einem anderen Thread aufgerufen wird, als derjenige, der das Ausgabefenster erzeugt hat, muss es eine aktive Schleife geben, innerhalb derer die Nachrichten vom Betriebssystem für dieses Fenster verarbeitet werden. Wird dieser Parameter auf *'true'* gesetzt, werden alle HALCON Grafik- und Textfenster, die kein Vaterfenster haben, automatisch in einem speziellen Thread geöffnet, der für die Verarbeitung der Nachrichten verantwortlich ist. Alle Grafikoperatoren, die sich an dieses Fenster richten, werden dann ebenfalls in diesem Thread abgearbeitet. Dieser Parameter wird auf anderen Betriebssystemen als Windows ignoriert.
  `Value`: *'true'* oder *'false'*
  Default: 'false'

  'x_package': Bei der Ausgabe von Bilddaten über das Netz kann es bei starker Auslastung des Rechners oder des Netzes zu Fehlübertragungen kommen. Um dies zu vermeiden, werden die Daten in kleinen Paketen transportiert. Wird lokal am Rechner gearbeitet, so können diese Einheiten beliebig vergrößert werden, was zu einer wesentlichen Verbesserung der Ausgabeleistung führt. Unter X-Windows wird beim ersten Öffnen eines Fensters der Maximalwert für die Größe eines Requests abgefragt und der Wert von *'x_package'* bei Bedarf verkleinert.
  `Value`: Paketgröße (in Bytes).
  Default: 131072

- **Bildverarbeitung:**

  'neighborhood', *'tsp_neighborhood'*: Dieser Parameter prüft entweder vier Pixel (in horizontaler und vertikaler Richtung) oder alle acht benachbarten Pixel. Verwendung bei allen Operatoren, bei denen Nachbarschaftsbeziehungen untersucht werden: `connection`, `get_region_contour`, `get_region_polygon`, `get_region_thickness`, `boundary`, `paint_region`, `disp_region`, `fill_up`, `contlength`, `shape_histo_all`.
  `Value`: *4* oder *8*
  Default: 8



(1) (2)

(1) Vier benachbarte Pixel, (2) acht benachbarte Pixel.

  'init_new_image', *'tsp_init_new_image'*: Legt fest, ob neue Bilder vor der Anwendung von Filtern auf 0 gesetzt werden sollen, um sicherzustellen, dass die Filterergebnisse bei wiederholter Programmausführung auf Systemen mit gleicher Konfiguration konsistent sind. Dies ist nicht nötig, falls immer das ganze Bild gefiltert wird oder die Daten der ungefilterten Bildbereichen nicht von Bedeutung sind.
  `Value`: *'true'* oder *'false'*
  Default: 'true'

  'no_object_result', *'tsp_no_object_result'*: Legt fest, wie sich Operationen, die Objekten verarbeiten, bei leeren Bildern (= keine Bilder) als Eingabe verhalten sollen.
  Mögliche Werte für `Value`:

  - *'true'*: Fehler wird ignoriert.
  - *'exception'*: Eine Fehlerbehandlung wird durchgeführt.

  Default: 'true'

  'empty_region_result', *'tsp_empty_region_result'*: steuert das Verhalten von Operatoren auf leere Eingaberegionen, die für solche Regionen eigentlich nicht sinnvoll sind (z.B. diverse Regionenmerkmale, Segmentierung, etc.).
  Mögliche Werte für `Value`:

  - *'true'*: Der Fehler wird so gut es geht ignoriert
  - *'exception'*: Eine Fehlerbehandlung wird durchgeführt.

  Default: 'true'

  'store_empty_region', *'tsp_store_empty_region'*: Bei einer Reihe von Operationen kann es vorkommen, dass Objekte mit leerer Region (= keine Bildpunkte) erzeugt werden (z.B. `intersection`, `threshold`, etc.). Dieser Parameter legt fest ob diese Operatoren ein Objekt mit leerer Region als Ergebnis ausgegeben (*'true'*) oder ob die Regionen unterdrückt also nicht ausgegeben werden sollen (*'false'*).
  `Value`: *'true'* oder *'false'*
  Default: 'true'

  'clip_region', *'tsp_clip_region'*: Legt fest, ob die Regionen in der HALCON-Datenbank auf das aktuelle Bildformat beschnitten („geclippt“) werden oder nicht. Dies ist bei Operatoren wie z.B. bei `gen_circle`, `gen_rectangle1` oder `dilation1` der Fall.
  Siehe auch `reset_obj_db`.
  `Value`: *'true'* oder *'false'*
  Default: 'true'

  'int_zooming': Bestimmt die Genauigkeit mit der gewisse Schritte der Bildverarbeitung ausgeführt werden. Für *'true'* wird Integer-Arithmetik oder schnelle Fließkommaarithmetik verwendet, für *'false'* wird Fließkommaarithmetik mit hoher Genauigkeit verwendet. Weitere Informationen wie ein spezifischer Operator durch den Parameter *'int_zooming'* beeinflusst wird, kann in der Dokumentation des direkt betroffenen Operators gefunden werden.
  `Value`: *'true'* oder *'false'*
  Default: 'true'

'pregenerate_shape_models': Dieser Parameter legt fest, ob die generierten Modelle vollständig vorab generiert werden (*'true'*) oder nicht (*'false'*), falls dies nicht explizit im formbasierten Matching-Operator festgelegt wird. Betroffene Operatoren:

- create_shape_model
- create_shape_model_xld
- create_scaled_shape_model
- create_scaled_shape_model_xld
- create_aniso_shape_model
- create_aniso_shape_model_xld

Value: *'true'* oder *'false'*
Default: 'false'

'border_shape_models': Dieser Parameter legt fest, ob die gesuchten Modelle teilweise außerhalb des Bildes liegen dürfen, also den Bildrand überschreiten dürfen, (*'true'*) oder nicht (*'false'*).
Betroffene Operatoren:

- find_component_model
- find_generic_shape_model
- find_shape_model
- find_shape_models
- find_scaled_shape_model
- find_scaled_shape_models
- find_aniso_shape_model
- find_aniso_shape_models
- find_planar_uncalib_deformable_model
- find_planar_calib_deformable_model
- find_local_deformable_model

Falls ein Wert für Formmodelle mit set_generic_shape_model_param gesetzt wurde, hat dieser spezielle Wert Vorrang vor den systemweiten Einstellungen.
Value: *'true'* oder *'false'*.
Default: 'false'.

'opengl_context_cache_enable': Dieser Parameter legt fest, ob der im Operator render_object_model_3d verwendete Grafikkartenkontext für weitere Aufrufe dieses Operators zwischengespeichert wird. Die Zwischenspeicherung beschleunigt spätere Aufrufe dieses Operators, benötigt aber zusätzliche Ressourcen. Um den Kontext explizit zu löschen muss *'opengl_context_cache_enable'* auf *'false'* gesetzt werden und render_object_model_3d einmalig aufgerufen werden. *'opengl_context_cache_enable'* kann nur auf *'true'* gesetzt werden, wenn die Grafikkarte das Rendern von 3D Objektmodellen unterstützt.
Value: *'true'* oder *'false'*
Default: 'true', sofern von der Grafikkarte unterstützt.

'opengl_hidden_surface_removal_enable': Dieser Parameter legt fest, ob die durch die Grafikkarte beschleunigte Verdeckungsberechnung in den Operatoren create_shape_model_3d, find_shape_model_3d, project_shape_model_3d und project_object_model_3d verwendet wird. *'opengl_hidden_surface_removal_enable'* kann nur auf *'true'* gesetzt werden, wenn die Grafikkarte die Beschleunigung des Algorithmus zur Verdeckungsberechnung unterstützt. Mindestvoraussetzungen sind OpenGL 2.0 und die Erweiterungen GL_EXT_framebuffer_object und GL_ARB_texture_float. Der get_system Parameter *'opengl_hidden_surface_removal_available'* kann verwendet werden, um diese Information abzufragen. Es ist nicht möglich diese Beschleunigung über den Windows Remote Desktop oder X11 Weiterleitung zu verwenden.
Value: *'true'* oder *'false'*
Default: 'true' sofern von der Grafikkarte unterstützt.

'opengl_compatibility_mode_enable': Dieser Parameter legt fest, ob die Visualisierung von 3D-Objektmodellen durch disp_object_model_3d mit hohen OpenGL-Anforderungen oder im Kompatibilitätsmodus mit OpenGL 1.1 durchgeführt wird. Wenn *'opengl_compatibility_mode_enable'* auf *'false'* gesetzt wird, dann entscheidet HALCON automatisch, welcher Visualisierungsmodus von der Grafikkarte unterstützt wird. Sollten bei der Visualisierung Probleme auftreten, so kann der Parameter auf *'true'* gesetzt werden. Value: *'true'* oder *'false'*
Default: 'false'.

'image_dpi': Dieser Parameter legt fest, welche DPI-Auflösung in Bilddateien, die mit `write_image` geschrieben werden, und die die DPI-Auflösung abspeichern, eingetragen wird.
`Value`: Auflösung in DPI.
Default: 300

'width', *'tsp_width'*: Legt die interne Bildbreite des Systems fest. Mit dieser Größe werden beispielsweise Regionen geclippt oder Speicherannahmen getroffen, wenn einem Operator keine weitere Bildinformationen zu Verfügung stehen. Diese Bildbreite wird als maximale Breite aller HALCON-Bildobjekte interpretiert, und somit beim Generieren neuer, größerer Bilder erhöht. Siehe auch `reset_obj_db`. Der Standardwert für die interne Bildbreite ist *512* in HDevelop (auch bei exportiertem Code), ansonsten *128*.
`Value`: Interne Bildbreite.
Default: 128 oder *512* (HDevelop)

'height', *'tsp_height'*: Legt die interne Bildhöhe des Systems fest. Mit dieser Größe werden beispielsweise Regionen geclippt oder Speicherannahmen getroffen, wenn einem Operator keine weitere Bildinformationen zu Verfügung stehen. Diese Bildhöhe wird als maximale Höhe aller HALCON-Bildobjekte interpretiert, und somit beim Generieren neuer, größerer Bilder erhöht. Siehe auch `reset_obj_db`. Der Standardwert für die interne Bildhöhe ist *512* in HDevelop (auch bei exportiertem Code), ansonsten *128*.
`Value`: Interne Bildhöhe.
Default: 128 oder *512* (HDevelop)

'current_runlength_number', *'tsp_current_runlength_number'*: Regionen werden intern in einer Lauflängenkodierung abgelegt. Mit diesem Parameter kann die maximale Anzahl der Sehnen festgelegt werden, die für die Darstellung einer Region verwendet werden können. Es ist zu beachten, dass einige Operatoren selbständig die Anzahl hochsetzen, falls dies notwendig ist. Der Wert kann sowohl vergrößert als auch verkleinert werden.
`Value`: Maximale Sehnenzahl.
Default: 50000

- **Parallelisierung:**

'parallelize_operators' *): Gibt an, ob HALCON eine automatische Parallelisierung einsetzt, um die Abarbeitung von Operatoren auf Mehrprozessormaschinen zu beschleunigen. Durch das Setzen auf *'false'* wird die automatische Parallelisierung abgeschaltet, wobei HALCON jedoch auch weiterhin nicht nur thread-safe bleibt, sondern auch die gleichzeitige Abarbeitung mehrerer HALCON-Operatoren erlaubt. Das Setzen auf *'false'* ist zum Beispiel dann sinnvoll, wenn HALCON-Operatoren innerhalb eines parallelen Programms aufgerufen werden, das auch das Scheduling bzw. ein geeignetes Load-Balancing der Operatoren und Daten selbst festlegt. In diesem Fall kann es wünschenswert sein, dass HALCON nicht auch noch zusätzliche Parallelisierungsschritte selbst durchführt, die dem Load-Balancing-Konzept der Anwendung in die Quere kommen könnten. Um eine feinere Kontrolle über die automatische Parallelisierung zu erreichen können auch einzelne Methoden der Datenparallelisierung an- und ausgeschaltet werden. *'split_tuple'* schaltet die Möglichkeit der Tupelparallelisierung, *'split_channel'* der Bildkanalparallelisierung, *'split_domain'* der Bilddomainparallelisierung und *'split_partial'* die der teilweisen internen Operatorparallelisierung wieder an. Mit einem vorangestelltem ' ' kann man die entsprechende Parallelisierungsmethode deaktivieren. Durch die Übergabe eines Tupels können auch mehrere Methoden gleichzeitig an- bzw. ausgeschaltet werden. So ist das Tupel [*'split_tuple'*,*'split_channel'*, *'split_domain'*,*'split_partial'*] gleichbedeutend mit dem Wert *'true'*.
Welche Methoden von einem Operator unterstützt werden, kann der Operator-Referenz entnommen werden oder über den Operator `get_operator_info` mit Hilfe des Parameterwertes *'parallel_method'* erfragt werden.
`Value`: *'true'*, *'false'*, *'split_tuple'*, *'split_channel'*, *'split_domain'*, *'split_partial'*, *' split_tuple'*, *' split_channel'*, *' split_domain'*, *' split_partial'*
Default: 'true'

**'reentrant'*: Gibt an, ob HALCON innerhalb einer parallel arbeitenden Programmumgebung eingesetzt wird, oder nicht. Ist der Parameter auf *'true'* gesetzt, werden interne Synchronisierungsmechanismen verwendet, um gemeinsam benutzte Datenobjekte vor parallelen Zugriffen zu schützen. So notwendig diese Maßnahme in einer echt parallel arbeitenden Anwendung sein mag, so unerwünscht ist der dadurch entstehende Overhead, falls die Anwendung alle HALCON-Operatoren rein sequentiell aufruft. Für den letzteren Fall kann man dem System die sequentielle Arbeitsweise durch das Setzen auf *'false'* signalisieren. Hierauf werden die internen Synchronisierungsmechanismen außer Kraft gesetzt, so dass sich der Overhead verringert. Daraus folgt natürlich, dass HALCON dann nicht mehr thread-safe ist.

Hierdurch ergibt sich als Seiteneffekt, dass durch das Setzen von *'reentrant'* auf *'false'* auch die interne Parallelisierung außer Kraft gesetzt wird, die HALCON normalerweise einsetzt, um die Abarbeitung von Operatoren auf Mehrprozessormaschinen zu beschleunigen. Wird *'reentrant'* hingegen auf *'true'* gesetzt, so geht HALCON in den Standardbetrieb über, das heißt, es arbeitet thread-safe und es setzt die automatische Parallelisierung von Operatoren ein.
`Value`: *'true'* oder *'false'*
Default: 'true'

'thread_num' ***), *'tsp_thread_num'*:** Setzt die Anzahl der Threads, die von HALCON für die automatische Operator Parallelisierung (AOP) verwendet werden. Die Zahl beinhaltet den Main-Thread und kann aus Effizienzgründen die Anzahl der Prozessoren nicht übersteigen. Das Herabsetzen der Threadanzahl ist interessant, wenn neben der automatischen Parallelisierung weitere Arbeitsthreads des Benutzers Prozessoren belegen. Damit kann die Anzahl von verarbeitenden Threads in einer Applikation der Prozessoranzahl angeglichen werden. Wurde für den HALCON-Prozess eine Prozessorzugehörigkeit festgelegt, kann mit *'default'* auf diese festgelegte Anzahl zurückgeschaltet werden. Andernfalls setzt *'default'* die Threadanzahl auf die Anzahl der Prozessoren. Wenn die Thread-spezifische Variante (*'tsp_thread_num'*) verwendet wird, reserviert HALCON die angegebene Anzahl an Threads exklusiv für den aufrufenden Thread. Falls die Summe der Thread-spezifisch reservierten Threads die über *'thread_num'* spezifizierte Anzahl von Threads übersteigt, so wird letztere entsprechend erhöht. Wurden für den aufrufenden Thread noch keine AOP-Threads Thread-spezifisch reserviert, gibt `get_system` für den Parameterwert *'tsp_thread_num'* den Wert -1 zurück.
Gibt man als Anzahl der Threads 1 an, wird die automatische Parallelisierung ausgeschaltet (Thread-spezifisch, wenn angegeben).
`Value`: *1* <= Value <= *'processor_num'*, *'default'*
Default: 'default'

'thread_pool' ***):** Gibt an, ob HALCON für die automatische Parallelisierung stets neue Threads erzeugt (*'false'*) oder auf einen bestehenden Pool von Threads zurückgreift (*'true'*). Mit der Aktivierung eines Thread-Pools arbeitet die automatische Parallelisierung effizienter. Wird die automatische Parallelisierung jedoch dauerhaft nicht benötigt, kann der Pool deaktiviert werden, um Ressourcen des Betriebssystems zu sparen.
`Value`: *'true'*, *'false'*
Default: 'true'

- **Datei:**

'flush_file': Mit diesem Parameter wird festgelegt, ob die Ausgabezeichen des Operators `fwrite_string` direkt im Ausgabemedium angezeigt werden. Wenn er auf *'false'* gesetzt ist, werden die Zeichen auf dem Terminal in der Regel erst nach einem Aufruf des Operators `fnew_line` sichtbar.
`Value`: *'true'* oder *'false'*
Default: 'true'

'ocr_trainf_version': Dieser Parameter legt die Version des Dateiformats fest, mit dem eine OCR-Trainingsdatei geschrieben wird. So speichern die Operatoren `write_ocr_trainf`, `write_ocr_trainf_image` und `concat_ocr_trainf` die Trainingsdaten für Versionsnummer *1* im ASCII-Format und für Versionsnummer *2* und *3* im Binärformat ab. Versionsnummer *3* kann byte- und uint2-Bilder abspeichern. Die binäre Version schreibt und liest die Daten schneller und speichert die Datei in einem komprimierteren Format ab. Das ASCII-Format ist kompatibel zu älteren HALCON-Releases. Abhängig von der Version des Dateiformats können die OCR-Trainingsdateien von folgenden HALCON-Releases eingelesen werden:

  - `1` - Alle HALCON-Releases
  - `2` - 7.0.2 und höher
  - `3` - 7.1 und höher

`Value`: *1*, *2*, *3*
Default: 3

'filename_encoding': Dieser Parameter existiert nur aus Gründen der Rückwärtskompatibilität und sollte nicht mehr verwendet werden. Er bestimmt wie die HALCON Bibliothek intern Zeichenketten darstellt. Der Standard ist UTF-8, welches alle Zeichen des Unicode-Standards unterstützt.
Die Zeichenkodierung der Datacode-Leser, Datei-Handles, Sockets, Tupel-Operatoren, der HALCON/C++ Schnittstelle und der HALCON/C Schnittstelle ist unabhängig von diesem Parameter. Für alle kann UTF-8 oder das lokale 8-Bit-Encoding als Zeichenkodierung verwendet werden. Siehe dazu die entsprechende Dokumentation.

Die Einstellung *'filename_encoding'* ist global und beeinflusst alle Threads. Dies bedeutet, wird dieser Wert in einem Thread geändert während andere Threads Strings bearbeiten, resultiert ein undefiniertes Verhalten, welches zu Abstürzen führen kann. Erfordert eine eine ältere Anwendung weiterhin, dass die HALCON Bibliothek intern das lokale 8-Bit-Encoding verwendet, findet man Hinweise zum Legacy-Modus in den Technical Updates .
`Value`: *'utf8'* oder *'locale'*
Default: 'utf8'

'write_halcon_files_encoding': Seit HALCON 18.05 wird in allen proprietären HALCON-Dateien sowie beim Serialisieren UTF-8 zum Kodieren von benutzerdefinierten Strings verwendet. Das betrifft vor allem das Schreiben von HALCON-Tupeln als auch die OCR- und OCV-Klassifikatoren, Trainingsdaten und Sample-Identifikatoren, da diese vom Anwender definierte Klassen- bzw. Zeichennamen enthalten können. Dadurch wird der Austausch dieser Dateien zwischen unterschiedlichen Betriebssystemen, Ländern und Spracheinstellungen deutlich vereinfacht. Soll eine Datei mit Sonderzeichen allerdings auch in einer Anwendung verwendet werden, die mit einer älteren HALCON-Version (vor HALCON 18.05) gebaut wurde und sich auf das lokale Encoding der Strings verlässt, kann mit Hilfe dieser Option das Schreiben mit dem lokalen Encoding erzwungen werden. Das ist nicht notwendig, wenn die zu schreibenden HALCON-Dateien nur reine ASCII-Strings enthalten. Außerdem gilt diese Option nur für proprietäre HALCON-Dateien, nicht jedoch für Anwender-Dateien, die z.B. mit `open_file` geöffnet werden.
`Value`: *'locale'* or *'utf8'*
Default: 'utf8'

'read_halcon_files_encoding_fallback': Ältere proprietäre HALCON-Dateien (vor HALCON 18.05) werden standardmäßig mit dem lokalen System-Encoding gelesen. Wurde stattdessen die einzulesende Datei bereits in UTF-8 geschrieben, erlaubt diese Option das Encoding für das Lesen zu ändern. Das betrifft vor allem das Lesen von HALCON-Tupeln als auch die OCR- und OCV-Klassifikatoren, Trainingsdaten und Sample-Identifikatoren, da diese vom Anwender definierte Klassen- bzw. Zeichennamen enthalten können. Dabei ist zu beachten, dass diese Option keinen Einfluss auf das Lesen von neueren HALCON-Dateien (ab HALCON 18.05) hat und auch nicht angegeben werden muss für Dateien, die nur reine ASCII-Strings enthalten. Außerdem gilt diese Option nur für proprietäre HALCON-Dateien, nicht jedoch für Anwender-Dateien, die z.B. mit `open_file` geöffnet werden.
`Value`: *'locale'* or *'utf8'*
Default: 'locale'

'tuple_string_operator_mode', *'tsp_tuple_string_operator_mode'*: Dieser Parameter bestimmt, wie UTF-8 Strings durch Tupel-String-Operatoren, wie `tuple_ord`, `tuple_strlen` oder `tuple_substr` verarbeitet werden, wenn *'filename_encoding'* auf den Wert *'utf8'* gesetzt ist, welcher dem Defaultwert entspricht. Standardmäßig werden Strings nicht byteweise, sondern zeichenweise bzw. nach Unicode-Codepoints verarbeitet. D.h. alle Zeichen, die aus einem einzelnen Unicode-Codepoint gebildet werden, haben eine Stringlänge von 1, und der Zugriff auf das n-te Element eines Strings gibt immer den Unicode-Codepoint des entsprechenden Zeichens zurück, unabhängig davon, wie viel Bytes benötigt werden, um den Codepoint oder die Codepoints davor in UTF-8 darzustellen. Dieses Verhalten kann geändert werden, indem *'tuple_string_operator_mode'* auf *'byte'* gesetzt wird. In diesem Modus werden Strings byteweise verarbeitet.
`Value`: *'codepoint'* or *'byte'*
Default: 'codepoint'

- **Verzeichnisse:**

'halcon_dir' *): Dieser Parameter liefert das Hauptverzeichnis der HALCON-Installation zurück.
`Value`: Name des Verzeichnisses.

'example_dir' *): HDevelop-Beispielprogramme werden im Verzeichnis *'example_dir'* gesucht, wenn sie z.B. über den Dialog `HDevelop Beispielprogramme durchsuchen` oder aus den Abschnitten `HDevelop Beispiele` der Operator-Referenz geladen werden. Es ist zu beachten, dass diese Mechanismen zum Laden von HDevelop Beispielprogrammen nur dann funktionieren, wenn *'example_dir'* auf ein Verzeichnis gesetzt ist, welches die installierten HDevelop Beispielprogramme enthält.
`Value`: Name des Verzeichnisses.

'calib_dir' *): Kameramodellparameter (z.B. `read_cam_par`) werden im aktuellen Verzeichnis und im *'calib_dir'* gesucht (sofern keine absoluten Pfade angegeben werden). Man kann mehrere Verzeichnisse angeben (Suchpfad), jeweils getrennt durch ein Semikolon (Windows) bzw. einen Doppelpunkt (unixartige Systeme).
`Value`: Name des Verzeichnisses.

*'dl_dir' **):** Deep Learning Modelldaten (z.B. `read_dl_model`) werden im aktuellen Verzeichnis und im *'dl_dir'* gesucht (sofern keine absoluten Pfade angegeben werden). Man kann mehrere Verzeichnisse angeben (Suchpfad), jeweils getrennt durch ein Semikolon (Windows) bzw. einen Doppelpunkt (unixartige Systeme).
`Value`: Name des Verzeichnisses.

*'filter_dir' **):** Filterparameter (z.B. `convol_image`) werden im aktuellen Verzeichnis und im *'filter_dir'* gesucht (sofern keine absoluten Pfade angegeben werden). Man kann mehrere Verzeichnisse angeben (Suchpfad), jeweils getrennt durch ein Semikolon (Windows) bzw. einen Doppelpunkt (unixartige Systeme).
`Value`: Name des Verzeichnisses.

*'image_dir' **):** Bilddaten (z.B. `read_image` und `read_sequence`) werden im aktuellen Verzeichnis und im *'image_dir'* gesucht (sofern keine absoluten Pfade angegeben werden). Man kann mehrere Verzeichnisse angeben (Suchpfad), jeweils getrennt durch ein Semikolon (Windows) bzw. einen Doppelpunkt (unixartige Systeme). Der Pfad kann auch mit der Umgebungsvariablen HALCONIMAGES festgelegt werden.
`Value`: Name des Verzeichnisses.

*'3d_model_dir' **):** 3D-Objektmodelle (z.B. `read_object_model_3d`) werden im aktuellen Verzeichnis und im *'3d_model_dir'* gesucht (sofern keine absoluten Pfade angegeben werden). Man kann mehrere Verzeichnisse angeben (Suchpfad), jeweils getrennt durch ein Semikolon (Windows) bzw. einen Doppelpunkt (unixartige Systeme).
`Value`: Name des Verzeichnisses.

*'lut_dir' **):** Farbtabellen (`set_lut`), die als ASCII-Datei realisiert sind, werden im aktuellen Verzeichnis und im *'lut_dir'* gesucht (sofern keine absoluten Pfade angegeben werden). Per Default-Einstellung sucht HALCON die Farbtabellen im Unterverzeichnis `'lut'`.
`Value`: Name des Verzeichnisses.

*'help_dir' **):** Die Online-Textdateien german oder english.hlp, .key, .sta, .num und .idx werden im aktuellen Verzeichnis und im *'help_dir'* gesucht. Dieser Systemparameter wird u.a. für `get_operator_info` und `get_param_info` benötigt. Dieser Parameter kann auch vor dem Aufruf von HALCON durch die Umgebungsvariable HALCONROOT gesetzt werden. Dabei muss die Variable auf das Verzeichnis oberhalb des Hilfe-Verzeichnisses zeigen (das ist das HALCON Home-Verzeichnis): z.B.: '/usr/local/halcon'
`Value`: Name des Verzeichnisses.

*'ocr_dir' **):** OCR Klassifikatoren (z.B. `read_ocr_class_knn`) werden im aktuellen Verzeichnis und im *'ocr_dir'* gesucht (sofern keine absoluten Pfade angegeben werden). Man kann mehrere Verzeichnisse angeben (Suchpfad), jeweils getrennt durch ein Semikolon (Windows) bzw. einen Doppelpunkt (unixartige Systeme).
`Value`: Name des Verzeichnisses.

- **Lizenzierung:**

'available_license_files': Die Liste aller von HALCON gefundenen Lizenzdateien.

'current_license_info': Ein Tupel mit Informationen zur aktuellen Lizenz. Ist keine gültige Lizenz vorhanden, besteht das Tupel aus einem einzigen Element mit dem Wert *'invalid'*. Ansonsten enthält das Tupel folgende Elemente in dieser Reihenfolge:

  - Der Name der Lizenzdatei, aus der die aktuelle Lizenz gelesen wurde.
  - Die Art der Lizenz, entweder *'runtime'*, *'development'*, oder *'sasl'*.
  - Die höchste Version von HALCON, die aktuell lizenziert ist.
  - Das Ablaufdatum der aktuellen Lizenz im ISO 8601 Format, oder der Wert *'permanent'* für Lizenzen ohne Ablaufdatum.
  - Die Liste der lizenzierten HALCON-Module, getrennt mit `' | '`.
  - Die Anzahl der gleichzeitig nutzbaren Lizenzen, oder 0 für unbeschränkte Lizenzen.
  - Die Anzahl der gleichzeitig nutzbaren Sessions, oder 0 für eine unbegrenzte Anzahl von Sessions.
  - Die Host-ID, die für die aktuelle Lizenz verwendet wurde.
  - Die Flags der aktuellen Lizenz.
  - Der Inhalt des Informationsfeldes der Lizenz.

  Wenn eine bestimmte Information zur Lizenz nicht verfügbar ist, wird statt dessen ein leerer String zurückgegeben.

'expiration_date': Das Ablaufdatum der aktuellen Lizenz im ISO 8601 Format, oder der Wert *'permanent'* for Lizenzen ohne Ablaufdatum.

'failed_license_rechecks': Die Anzahl der fehlgeschlagenen Lizenzversuche seit Prozessstart.

'is_license_valid': Dieser Parameter löst eine sofortige Lizenzprüfung aus und gibt *'true'* zurück, falls eine gültige Lizenz gefunden wurde, ansonsten *'false'*.

'licensed_hostid': Die Host-ID, die für die aktuelle Lizenz verwendet wurde.

'licensed_modules': Die Liste aller aktuell lizenzierter Module.

'licensed_version': Die maximale Version von HALCON, die aktuelle lizenziert ist.

'unlicensed_operators': Eine Liste aller HALCON-Operatoren, die nicht ausgeführt werden können, weil das benötigte Modul nicht lizenziert ist.

- **Andere:**

'available_parameters': Die Liste aller von `get_system` unterstützten Parametern.

'disabled_operators': Mit diesem Parameter können einzelne HALCON-Operatoren deaktiviert werden. Dies kann sinnvoll sein, wenn ein Programm Benutzer-definierte HDevelop-Skripten ausführen kann, aber aus Sicherheitsgründen verhindert werden soll, dass bestimmte Operatoren wie z.B. `system_call` ausgeführt werden können. Jeder Versuch, einen deaktivierten Operator auszuführen, liefert den Fehler H_ERR_OP_DISABLED (9055) zurück. Es ist zu beachten, dass ein einmal deaktivierter Operator während der Restlaufzeit des aktuellen Prozesses nicht wieder aktiviert werden kann.
Der Wert für diesen Parameter ist ein Tupel, das die Namen aller zu deaktivierenden Operatoren enthält.

'do_low_error': Bestimmt das Verhalten im Fall von HALCON Low-Level-Fehlern.

- Ist der Parameter *'do_low_error'* auf *'false'* gesetzt, so werden keine Low-Level-Fehler ausgegeben. Die Low-Level-Fehler werden jedoch weiterhin in der Ausgabekonsole (einem Fenster, in dem man ein Protokoll der letzten Meldungen ansehen kann) angezeigt, die über das Fenster-Menü aufgerufen werden kann.
- Ist der Parameter auf *'disabled'* gesetzt, so werden die Low-Level-Fehler ganz unterdrückt und erscheinen auch in der Ausgabekonsole nicht mehr.
- Ist der Parameter auf *'stderr'* bzw. *'message_box'* gesetzt, so wird der entsprechende Low-Level-Fehlertext in die Standard-Fehlerausgabe geschrieben, bzw. wird ein Dialog geöffnet, der den Fehlertext anzeigt (letztere Funktionalität ist nur auf Windows Systemen implementiert).
- Mithilfe des Parameterwertes *'callback'* kann eine Callback-Prozedur angegeben werden, die im Falle eines Low-Level-Fehlers aufgerufen werden soll. Die Adresse dieser Callback-Prozedur wird im zweiten Indexeintrag des Parameters `Value` angegeben.
  Die Signatur der Callback-Funktion lautet folgendermaßen:
  ```
  Herror LowErrorCallbackProc(const char* err_text)
  ```
  Auf Windows 32-Bit-Systemen wird die __stdcall Namenskonvention verwendet:
  ```
  Herror (__stdcall LowErrorCallbackProc)(const char* err_text)
  ```
  Der Parameterwert *'callback'* kann in HDevelop nur verwendet werden, falls als Callback-Prozedur 0 angegeben wird. In diesem Fall erfolgt keine Fehlerausgabe. Die Zeichenkodierung der Fehlermeldung entspricht der Zeichenkodierung der HALCON Bibliothek wie sie mit *'filename_encoding'* bestimmt ist.
- Sollen Low-Level-Fehlertexte in eine Datei geschrieben werden, so kann dies über den Parameterwert *'file'* angegeben werden. Dabei muss das Handle einer zuvor mittels `open_file` geöffneten Datei zusätzlich übergeben werden.
- Die Parameterwerte *'callback'* bzw. *'file'* können nur in Kombination mit einer entsprechenden callback-Prozedur bzw. einem File-Handle verwendet werden.
- Für *'do_low_error'* können, mit Ausnahme von *'false'*, auch mehrere Parameterwerte in einem Tupel übergeben werden. Die entsprechenden Aktionen werden im Fall eines Low-Level-Fehlers in der Reihenfolge der im Parameterwertetupel angegebenen Werte ausgeführt. Beinhaltet das Tupel mehrmals den gleichen Parameterwert, so wird lediglich das erste Vorkommnis berücksichtigt, d.h. es ist nicht möglich, im Fehlerfall die gleiche Aktion öfters ausführen zu lassen. Falls für *'do_low_error'* mehrere Parameterwerte oder einer der Werte *'file'* oder *'callback'* angegeben werden, so kann *'do_low_error'* nicht zusammen mit anderen Systemparametern in einem Aufruf von `set_system` gesetzt werden.
- Jedes Setzen von *'do_low_error'* mittels `set_system` überschreibt die vorherigen Einstellungen dieses Systemparameters.
- Wird *'do_low_error'* ausschließlich auf den Wert *'true'* gesetzt, so entspricht dies auf Windows-Systemen dem Parameterwert *'message_box'* und auf unixartigen Systemen dem Parameterwert *'stderr'*.

Value: *'true'*, *'false'*, *'disabled'*, *'stderr'*, *'message_box'*, *'callback'*, *'file'*
Default: 'false'

'cancel_draw_result', *'tsp_cancel_draw_result'*: Viele "draw_"Operatoren (wie z.B. draw_region oder draw_rectangle1) können abgebrochen werden, indem das Zeichnen mit der rechten Maustaste beendet wird, ohne zuvor neue Objekte gezeichnet oder bestehende modifiziert zu haben. "draw_"Operatoren können auch über die Stopptaste im HDevelop abgebrochen werden. Der abgebrochene Operator liefert leere Objekte bzw. leere Tupel zurück. *'cancel_draw_result'* steuert das Verhalten von "draw_"Operatoren die abgebrochen wurden.
Mögliche Werte für Value:

- *'true'*: Es wird kein Fehler zurückgegeben.
- *'exception'*: Eine Fehlerbehandlung wird durchgeführt.

Default: 'true'

'seed_rand': Setzt den Seed des threadspezifischen Zufallszahlengenerators, der in den Operatoren tuple_rand, add_noise_white, add_noise_distribution, gen_random_region und add_noise_white_contour_xld verwendet wird. Wurde kein Seed gesetzt, wird die Systemzeit beim ersten Aufruf von einem der genannten Operatoren als Seed verwendet. Der verwendete Seed kann dann mit get_system abgefragt werden.
Dieser Parameter akzeptiert nur ganzzahlige Werte oder den String *'default'* um das das Defaultverhalten wiederherzustellen.

'cudnn_deterministic': Legt fest, ob die cuDNN-Bibliothek für Deep Learning deterministische Algorithmen verwendet, um die bitweise Reproduzierbarkeit bei Ausführung auf GPUs mit der gleichen Architektur und der gleichen Anzahl von SMs zu erzwingen. Beachten Sie, dass die Reproduzierbarkeit über verschiedene cuDNNN-Versionen hinweg nicht gewährleistet ist. Weiter kann die Verwendung von deterministischen Algorithmen auf manchen Architekturen zur Verlangsamung der Berechnungen führen.
Value: *'true'* oder *'false'*
Default: 'false'

'clock_mode': Legt fest, auf welche Art die Messung von Zeitintervallen in count_seconds erfolgt.
Value:

- *'performance_counter'* misst die vergangene Systemzeit mit möglichst hoher Genauigkeit. In Abhängigkeit vom verwendeten Betriebssystem werden unterschiedliche Verfahren verwendet. Sollten bestimmte Messfunktionen vom jeweiligen System nicht unterstützt werden, wird auf weniger hoch aufgelöste Verfahren zurückgegriffen. Unter Windows wird der Performance Counter benutzt. Für nähere Informationen über den Performance Counter wird auf die Windows Dokumentation im Microsoft Developer Network (MSDN) verwiesen. Der Performance Counter bietet, soweit vorhanden, die beste Genauigkeit (unter einer Millisekunde). Allerdings besteht die Möglichkeit, dass er aufgrund von Problemen mit der Energieverwaltung und/oder dem Multithreading falsche Werte liefert. Wird unter Windows kein Performance Counter unterstützt, so greift die Einstellung *'multimedia_timer'*. Auf unixartigen Systemen (z.B. Linux oder macOS) ist die Genauigkeit ebenso unter einer Millisekunde und auch hier könnten Probleme durch Energieverwaltung und Multithreading auftreten. Sollten Probleme dieser Art auftreten, können Sie die Einstellung *'elapsed_time'* verwenden.
- *'elapsed_time'* misst die vergangene Systemzeit ähnlich wie bei der Einstellung *'performance_counter'*. Der Unterschied besteht darin, dass Messfunktionen verwendet werden, die eine geringere Auflösung haben, aber dafür weniger sensibel auf Energieverwaltung und Multithreading reagieren. Die Genauigkeit liegt in der Regel bei einer Millisekunde. Diese Einstellung sollte verwendet werden, wenn die Messungen im Modus *'performance_counter'* unzuverlässig sind und Ihnen die Auflösung von einer Millisekunde ausreicht.
- *'multimedia_timer'* verwendet unter Windows die Multimedia Timer. Für nähere Informationen über Multimedia Timer wird auf die Windows Dokumentation im Microsoft Developer Network (MSDN) verwiesen. Multimedia Timer stellen eine Auflösung von einer Millisekunde bereit. Sollte auf Ihrem System diese Auflösung nicht erreicht werden, können Sie versuchen diese mittels der Windows Funktionen 'timerBeginPeriod' und 'timerEndPeriod' zu erhöhen (mehr Informationen darüber finden Sie ebenso im MSDN). Unter unixartigen Betriebssystemen gibt es keine Multimedia Timer, und daher werden dort dieselben Verfahren wir im Modus *'performance_counter'* verwendet.
- *'processor_time'* misst die CPU-Rechenzeit des aktuellen Prozesses. Diese Messung ist weitgehend unabhängig von der Systemlast durch andere Prozesse. Ein großer Nachteil ist jedoch, dass auf den meisten Systemen die Auflösung gering und daher die Messung ungenau ist. Weiterhin ist zu

beachten, dass die Laufzeit vieler Anwendungen nicht nur durch CPU-Rechenzeit, sondern auch durch Input/Output oder das Speicherverhalten stark beeinflusst wird. In solchen Anwendungen sollte die Einstellung *'performance_counter'* oder *'elapsed_time'* verwendet werden.

Default: 'performance_counter'
Man beachte, dass bei den Einstellungen *'performance_counter'* und *'elapsed_time'* die tatsächlich vergangene Systemzeit gemessen wird. Die Länge eines gemessenen Intervalls hängt folglich auch von der Auslastung des Systems durch weitere Prozesse ab. Wird dieser Modus zur Messung der Verarbeitungszeit des eigenen Prozesses verwendet, muss der Benutzer dafür sorgen, dass die Zeitmessung nicht durch andere Prozesse gestört wird.

'timer_mode': Legt fest, auf welche Art die Zeitmessung bei Timeouts durchgeführt werden soll.
`Value`:

- *'elapsed_time'*: Siehe Beschreibung in Abschnitt *'clock_mode'*.
- *'multimedia_timer'*: Siehe Beschreibung in Abschnitt *'clock_mode'*.
- *'performance_counter'*: Siehe Beschreibung in Abschnitt *'clock_mode'*.

Default: 'multimedia_timer' auf Windows-Systemen, *'elapsed_time'* auf unixartigen Systemen

'max_connection': Legt die maximale Anzahl von Regionen fest, die von `connection` zurückgeliefert wird. Für `Value`=*0* werden alle Regionen zurückgeliefert.
`Value`: >=*0*
Default: 0

'extern_alloc_funct' ***)**: Zeiger auf externe Funktion zur Speicherallokation von Ergebnisbildern. Diese Funktion sollte die folgende Signatur haben: 'void* ExternAllocFunc(size_t)'. Wenn *0* übergeben wird, dann wird die HALCON-Allokationsfunktion verwendet.
`Value`: Funktionszeiger.
Default: 0

'extern_free_funct' ***)**: Zeiger auf externe Funktion zur Speicherfreigabe von Ergebnisbildern. Diese Funktion sollte die folgende Signatur haben: 'void ExternFreeFunct(void*)'. Wenn *0* übergeben wird, dann wird die HALCON-Freigabefunktion verwendet.
`Value`: Funktionszeiger.
Default: 0

'image_cache_capacity' ***)**: Um das Allokieren von neuen Bildern zu beschleunigen, wird bestehender Bildspeicher nicht freigegeben sondern wiederverwendet, solange die Speichergröße aller gepufferten Bilder die angegebene Grenze nicht überschreitet. Mit diesem Parameter kann man die Obergrenze für Speicher in Bytes setzen. Diese Funktionalität kann abgeschaltet werden, indem *'image_cache_capacity'* auf 0 gesetzt wird.
`Value`: Obergrenze für die Größe des HALCON Bildspeichers.
Default: 16777216 (16MByte)

'global_mem_cache': Modus des Caches für globalen, außerhalb von Operatoren sichtbaren Speicher. Er regelt, ob unbenutzter globaler Speicher zwischengespeichert (*'shared'*) oder freigegeben (*'idle'*) wird. Zusätzlich gibt es die Möglichkeit, den globalen Speicher für jeden Thread separat zwischenzuspeichern (*'exclusive'*). In diesem Fall kann die Rechengeschwindigkeit auf Kosten des Speicherbedarfs sinken. Mit Hilfe des Aktionsparameters *'cleanup'* können die im Cache gehaltenen Speicherblöcke bei Bedarf wieder physikalisch freigegeben werden.
`Value`: *'idle'*, *'exclusive'*, *'shared'*, *'cleanup'*
Default: 'exclusive'

'temporary_mem_cache' ***)**, *'tsp_temporary_mem_cache'*: Dieser Parameter steuert den Betriebsmodus des Caches für temporären Speicher, mit dessen Hilfe Anwendungen beschleunigt werden können. Für die meisten Anwendungen ist die Default-Einstellung (*'exclusive'*) die beste Wahl. Folgende Modi werden unterstützt:

- *'idle'* Der Cache ist abgeschaltet. In diesem Modus wird am wenigsten Speicher verbraucht, dafür wird die Anwendung aber langsamer laufen als in den anderen Modi.
- *'shared'* Alle temporären Speicherblöcke werden im globalen Reservoir für temporäre Speicherblöcke gecached. Dieser Modus benötigt weniger Speicher als der *'exclusive'*-Modus, allerdings wird die Anwendung im Normalfall etwas langsamer laufen.
- *'exclusive'* Alle temporären Speicherblöcke werden lokal von jedem Thread gecached. Dieser Modus verwendet am meisten Speicher, dafür wird aber normalerweise die Anwendung am schnellsten laufen.

– *'aggregate'* Temporäre Speicherblöcke, die größer als der mit dem Parameter *'alloctmp_max_blocksize'* eingestellte Schwellenwert sind, werden im globalen Reservoir gecached, während alle kleineren Blöcke zu einem einzigen Speicherblock zusammengefasst werden, den jeder Thread lokal cached. Die Größe des zusammengefassten Blocks bestimmt sich aus dem maximalen Bedarf an temporärem Speicher, den ein Thread bisher festgestellt hat, wird aber nicht größer sein als *'alloctmp_max_blocksize'*, bzw. kleiner als *'alloctmp_min_blocksize'* (sofern der entsprechende Parameter gesetzt ist). Dieser Modus versucht ein Gleichgewicht zwischen Ausführungszeit und Speicherverbrauch zu finden — für eine effektive Funktion müssen aber *'alloctmp_min_blocksize'* und *'alloctmp_max_blocksize'* gemäß der tatsächlichen Speichernutzung der Anwendung korrekt eingestellt werden.

Das Setzen des Cachemodus auf *'idle'* (oder *'false'*) geschieht exklusiv, siehe den obigen Abschnitt „Parallelisierungsinformation“. Im Gegensatz dazu geschieht das Setzen des Cachemodus auf einen anderen Modus im reentrant Modus, also ohne Blockieren anderer HALCON Operatoren.
Aus Kompatibilitätsgründen akzeptiert der Parameter auch die Werte *'false'* und *'true'*; dies entspricht *'idle'* bzw. *'exclusive'*.
`Value`: *'idle'*, *'shared'*, *'exclusive'*, oder *'aggregate'*
Default: 'exclusive'

'temporary_mem_reservoir', *'tsp_temporary_mem_reservoir'* Wird dieser Parameter auf *'true'* gesetzt, ist das globale Reservoir für temporäre Speicherblöcke aktiviert. Ist er auf *'false'* gesetzt, ist das Reservoir deaktiviert und alle Speicherblöcke, die sonst im Reservoir gecached würden, werden statt dessen freigegeben.
Jeder Thread, der einen neuen temporären Speicherblock benötigt, prüft zuerst, ob im Reservoir ein entsprechender Block verfügbar ist, bevor er vom System neuen Speicher anfordert. Speicherblöcke werden von Threads belegt, deren Caches für temporären Speicher im Modus *'shared'* oder *'aggregate'* betrieben werden.
Die globale Einstellung dieses Parameters hat Vorrang vor der thread-spezifischen Einstellung, d.h. wenn *'temporary_mem_reservoir'* auf *'true'* gesetzt ist, wird die Einstellung für *'tsp_temporary_mem_reservoir'* ignoriert.
`Value`: *'false'* oder *'true'*
Default: 'true'

'temporary_mem_reservoir_size': Die Anzahl von Bytes, die maximal im globalen Reservoir für temporäre Speicherblöcke gecached werden. Wird dieser Parameter auf *-1* gesetzt, ist das Reservoir nur von der insgesamt verfügbaren Speichermenge begrenzt.
`Value`: *-1* oder >= *0*
Default: -1

'alloctmp_min_blocksize', *'tsp_alloctmp_min_blocksize'* Minimale Größe der Speicherblöcke (in Bytes), die von der temporären Speicherverwaltung gecached werden. (Ohne Effekt, falls *'temporary_mem_cache'* == *'false'*). Mit der Default-Einstellung *-1* verwendet HALCON eine auf der Bildgröße basierende Heuristik, um eine vernünftige Blockgröße zu bestimmen. Für die meisten Anwendungen ist dies die beste Einstellung. Wird dieser Parameter auf einen Wert größer gleich Null gesetzt, verwendet HALCON an Stelle der Heuristik diesen Wert. Die minimale Größe wird ignoriert, sofern ein HALCON Operator ein Speicherobjekt benötigt, das nicht in einen Block dieser Größe hineinpasst.
`Value`: *-1* oder >= *0*
Default: -1

'alloctmp_max_blocksize', *'tsp_alloctmp_max_blocksize'* Maximale Größe der Speicherblöcke (in Bytes), die von der temporären Speicherverwaltung gecached werden. (Ohne Effekt, falls *'temporary_mem_cache'* == *'false'*). Mit Hilfe dieses Parameters kann die maximale Blockgröße begrenzt werden, die von der von HALCON verwendeten Heuristik bestimmt wird, wenn *'alloctmp_min_blocksize'* auf *-1* gesetzt ist. Sofern der temporäre Speichercache nicht im Cachemodus *'aggregate'* betrieben wird, sollte es generell unnötig sein, diesen Parameter zu setzen. Im *'aggregate'*-Modus begrenzt dieser Parameter zusätzlich die maximale Größe des kombinierten Speicherblocks und bestimmt, ab welcher Größe temporäre Speicherblöcke im globalen Reservoir gecached werden. In allen Modi wird dieser Parameter ignoriert, wenn ein HALCON Operator ein temporäres Speicherobjekt benötigt, das größer als die aktuelle maximale Blockgröße ist.
`Value`: *-1* oder >= *0*
Default: -1

'database' *): Legt fest, ob die in HALCON instanziierten Bilddatenobjekte in den fünf Relationen (Grauwertdaten, Regionendaten, XLDs, Bildobjekte und Objekttupel) der HALCON-Datenbank gelistet wer-

den. Die Relationen können dann beispielsweise zur Fehlerbeseitigung genutzt werden. Beachte, dass das Sammeln von Informationen für die Datenbank nicht thread-safe ist, wenn ikonisch Objekte zwischen den Threads ausgetauscht werden, d.h. wenn Objekte in einem anderen Thread zerstört werden als in dem sie erzeugt wurden.
Siehe auch count_relation,reset_obj_db.
Value: *'true'* oder *'false'*
Default: 'false'

'mmx_enable': Bestimmt, ob MMX-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'mmx_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU MMX unterstützt, sonst *'false'*

'sse_enable': Bestimmt, ob SSE-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'sse_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU SSE unterstützt, sonst *'false'*

'sse2_enable': Bestimmt, ob SSE2-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'sse2_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU SSE2 unterstützt, sonst *'false'*

'sse3_enable': Bestimmt, ob SSE3-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'sse3_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU SSE3 unterstützt, sonst *'false'*

'ssse3_enable': Bestimmt, ob SSSE3-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'ssse3_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU SSSE3 unterstützt, sonst *'false'*

'sse41_enable': Bestimmt, ob SSE41-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'sse41_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU SSE41 unterstützt, sonst *'false'*

'sse42_enable': Bestimmt, ob SSE42-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'sse42_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU SSE42 unterstützt, sonst *'false'*

'avx_enable': Bestimmt, ob AVX-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX unterstützt, sonst *'false'*

'avx2_enable': Bestimmt, ob AVX2-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx2_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX2 unterstützt, sonst *'false'*

'avx512f_enable': Bestimmt, ob AVX512F-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512f_supported'* == *'false'*, siehe hierzu Operator get_system)
Value: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512F unterstützt, sonst *'false'*

'avx512dq_enable': Bestimmt, ob AVX512DQ-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512dq_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512DQ unterstützt, sonst *'false'*

'avx512bw_enable': Bestimmt, ob AVX512BW-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512bw_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512BW unterstützt, sonst *'false'*

'avx512er_enable': Bestimmt, ob AVX512ER-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512er_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512ER unterstützt, sonst *'false'*

'avx512cd_enable': Bestimmt, ob AVX512CD-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512cd_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512CD unterstützt, sonst *'false'*

'avx512vl_enable': Bestimmt, ob AVX512VL-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512vl_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512VL unterstützt, sonst *'false'*

'avx512vbmi_enable': Bestimmt, ob AVX512VBMI-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512vbmi_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512VBMI unterstützt, sonst *'false'*

'avx512ifma_enable': Bestimmt, ob AVX512IFMA-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512ifma_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512IFMA unterstützt, sonst *'false'*

'avx512vbmi2_enable': Bestimmt, ob AVX512VBMI2-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512vbmi2_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512VBMI2 unterstützt, sonst *'false'*

'avx512vpopcntdq_enable': Bestimmt, ob AVX512VPOPCNTDQ-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512vpopcntdq_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512VPOPCNTDQ unterstützt, sonst *'false'*

'avx512bitalg_enable': Bestimmt, ob AVX512BITALG-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512bitalg_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512BITALG unterstützt, sonst *'false'*

'avx512vnni_enable': Bestimmt, ob AVX512VNNI-Operationen zur Beschleunigung ausgewählter Bildverarbeitungs-Operatoren verwendet werden sollen (*'true'*) oder nicht (*'false'*). (Ohne Effekt, falls *'avx512vnni_supported'* == *'false'*, siehe hierzu Operator `get_system`)
`Value`: *'true'* oder *'false'*
Default: 'true', falls die CPU AVX512VNNI unterstützt, sonst *'false'*

'language' **: Sprache, in der die Fehlermeldungen ausgegeben werden.
`Value`: *'english'* oder *'german'*.
Default: 'english'

'add_progress_callback', *'remove_progress_callback'*: Setzt bzw. entfernt eine Callback-Funktion zum Anzeigen von Operatorfortschritt und -nachrichten. Die Callback-Funktion wird von Operatoren, die dies unterstützten, in unregelmäßigen Abständen aufgerufen um den Fortschritt oder eine Nachricht anzuzeigen.
Die Signatur der Callback-Funktion ist

```
void HProgressBarCallback(Hlong id, const char *operator_name,
double progress, const char *message)
```

Auf Windows 32-Bit-Systemen wird die __stdcall Namenskonvention verwendet:

```
void (__stdcall HProgressBarCallback)(Hlong id, const char
*operator_name, double progress, const char *message)
```

Ein entsprechender Delegate HalconAPI.HProgressBarCallback ist für HALCON/.NET definiert und kann via Marshal.GetFunctionPointerForDelegate an `set_system` übergeben werden. Die Instanz muss dann am Leben gehalten werden solange der Zeiger in Benutzung bleibt.

```
public delegate void HProgressBarCallback(IntPtr id, string
operatorName, double progress, string message)
```

Die Parameter sind:

- *'id'*: Thread-ID, aus der der Operator ursprünglich aufgerufen wurde. Wenn die automatische Operatorparallelisierung aktiviert ist, kann der Callback auch aus einem anderem als dem ursprünglichen Thread erfolgen. Applikationen, die den Fortschritt in der GUI anzeigen wollen, müssen dies evtl. beachten.
- *'operator_name'*: Name des gerade ausgeführten Operators.
- *'progress'*: Die Interpretation hängt vom Wert des Parameters ab. Zwischen 0 und 1: Der ungefähre derzeitige Fortschritt des Operators. -1: Der Operator zeigt lediglich eine Nachricht an und setzt keinen Fortschritt, der Parameter sollte ignoriert werden. -2: Die Ausführung des Operators wurde abgeschlossen. Dies ist der letzte Callback-Aufruf für den Operator und eine Möglichkeit für den aufgerufenen, Aufräumaktionen durchzuführen. Dieser Aufruf wird nur für Operatoren durchgeführt, die den Callback mindestens einmal aufgerufen haben.
- *'message'*: Optionale Nachricht auf Englisch, die üblicherweise den Fortschritt des Operators beschreibt, beispielsweise den aktuellen Fehler beim Trainieren von Klassifikatoren. Wenn der Operator keine Nachricht anzeigen möchte, kann dieser Parameter auf NULL gesetzt sein.

'legacy_handle_mode', *'tsp_legacy_handle_mode'*: Flag zum Aktivieren oder Deaktivieren des Legacy Handle Modus. Wenn aktiviert, werden alle von HALCON-Operatoren zurückgegebenen Handles in Ganzzahlen konvertiert und alle Operatoren, die Handles erwarten, akzeptieren auch Ganzzahlen. In diesem Fall werden Handles nicht mehr automatisch freigegeben, wenn alle Tupel, in denen sie enthalten sind, überschrieben oder gelöscht wurden. Stattdessen muss `clear_handle` oder der clear-Operator des entsprechenden Handle-Typs aufgerufen werden.
Der Legacy Handle Mode ist nur zum Ausführen von Legacy-Code gedacht. Es wird nicht empfohlen, den Modus in neuem Code zu aktivieren. Bei aktiviertem Legacy Handle Modus funktionieren neuere Sprachfeatures oder Operatoren unter Umständen nicht mehr.
`Value`: *'true'* or *'false'*
Default: 'false'

'edition': Die Edition von HALCON, entweder *'progress'*, *'steady'*, oder *'student'*.

'product_edition': Der HALCON Editionsname. Ist entweder *'Progress'*, *'Steady'*, *'Steady Deep Learning'*, oder *'Student Edition'*.

Achtung

Es ist zu beachten, dass trotz der Information unter 'Parallelisierung' bezüglich des Multithreading-Typs nicht alle Parameter 'reentrant' sind. Parameter, auf die ein *) folgt, werden exklusiv ausgeführt.

Parameter

▷ **`SystemParameter`** (input_control) ..... attribute.name(-array) ⇝ *string*
Name des zu verändernden Systemparameters.
Defaultwert: 'init_new_image'
Werteliste: `SystemParameter` ∈ {'3d_model_dir', 'add_progress_callback', 'alloctmp_min_blocksize', 'alloctmp_max_blocksize', 'avx_enable', 'avx2_enable', 'avx512f_enable', 'avx512dq_enable', 'avx512bw_enable', 'avx512er_enable', 'avx512cd_enable', 'avx512vl_enable', 'avx512vbmi_enable', 'avx512ifma_enable', 'avx512vbmi2_enable', 'avx512vpopcntdq_enable', 'avx512bitalg_enable', 'avx512vnni_enable', 'backing_store', 'border_shape_models', 'calib_dir', 'cancel_draw_result',

'clip_region', 'clock_mode', 'cudnn_deterministic', 'current_runlength_number', 'database', 'default_font', 'disabled_operators', 'dl_dir', 'do_low_error', 'empty_region_result', 'example_dir', 'extern_alloc_funct', 'extern_free_funct', 'filename_encoding', 'filter_dir', 'flush_file', 'flush_graphic', 'global_mem_cache', 'height', 'help_dir', 'icon_name', 'image_cache_capacity', 'image_dir', 'image_dpi', 'init_new_image', 'int2_bits', 'int_zooming', 'language', 'legacy_handle_mode', 'lut_dir', 'max_connection', 'mmx_enable', 'neighborhood', 'no_object_result', 'opengl_hidden_surface_removal_enable', 'opengl_compatibility_mode_enable', 'opengl_context_cache_enable', 'ocr_dir', 'ocr_trainf_version', 'parallelize_operators', 'pregenerate_shape_models', 'read_halcon_files_encoding_fallback', 'reentrant', 'remove_progress_callback', 'seed_rand', 'sse_enable', 'sse2_enable', 'sse3_enable', 'ssse3_enable', 'sse41_enable', 'sse42_enable', 'store_empty_region', 'temporary_mem_cache', 'temporary_mem_reservoir', 'temporary_mem_reservoir_size', 'thread_num', 'thread_pool', 'timer_mode', 'tuple_string_operator_mode', 'tsp_cancel_draw_result', 'tsp_alloctmp_max_blocksize', 'tsp_alloctmp_min_blocksize', 'tsp_clip_region', 'tsp_current_runlength_number', 'tsp_empty_region_result', 'tsp_height', 'tsp_init_new_image', 'tsp_legacy_handle_mode', 'tsp_neighborhood', 'tsp_no_object_result', 'tsp_store_empty_region', 'tsp_temporary_mem_cache', 'tsp_temporary_mem_reservoir', 'tsp_thread_num', 'tsp_tuple_string_operator_mode', 'tsp_width', 'update_lut', 'use_window_thread', 'width', 'write_halcon_files_encoding', 'x_package'}

▷ **`Value`** (input_control) . . . . . attribute.value(-array) ⇝ *string* / integer / real
Neuer Wert des Systemparameters.
Defaultwert: 'true'
Wertevorschläge: `Value` ∈ {'true', 'false', 0, 4, 8}

Ergebnis

`set_system` liefert den Wert TRUE, falls die Parameter korrekt sind. Ansonsten wird eine Ausnahme ausgelöst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`reset_obj_db`, `get_system`, `set_check`

Siehe auch

`get_system`, `set_check`, `count_seconds`

Modul

Foundation

25.10 Serialisiertes Element

`clear_serialized_item` (: : `SerializedItemHandle` :)

Löschen eines serialisierten Elements.

`clear_serialized_item` löscht ein serialisiertes Element, welches in dem Handle `SerializedItemHandle` übergeben wird (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Nicht gelöscht wird ein serialisiertes Element, welches nur mittels eines Datenzeigers angelegt wurde. Erzeugt wird ein solcher Datenzeiger, der auf den Anfang eines serialisierten Elements zeigt, mit dem Operator `get_serialized_item_ptr`. Nach dem Aufruf ist das Handle des serialisierte Elements ungültig.

Parameter

▷ **`SerializedItemHandle`** (input_control) . . . . . serialized_item(-array) ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `clear_serialized_item` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SerializedItemHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`fwrite_serialized_item`, `receive_serialized_item`

Modul

Foundation

`create_serialized_item_ptr` (: : `Pointer`, `Size`, `Copy` : `SerializedItemHandle`)

Anlegen eines serialisierten Elements.

Es wird nicht empfohlen, den Operator `create_serialized_item_ptr` in HDevelop zu verwenden.

`create_serialized_item_ptr` legt ein serialisiertes Element an, und gibt dessen Handle `SerializedItemHandle` zurück (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Mit dem Parameter `Pointer` wird der Datenzeiger, d.h. der Anfang des serialisierten Elements, übergeben. Mit dem Parameter `Size` wird die Größe des serialisierten Elements in Byte übergeben. Mit dem Parameter `Copy` wird das Verhalten des Operators gesteuert. Falls `Copy` auf den Wert *'true'* gesetzt ist, wird ein neues serialisiertes Element erzeugt und die Werte des vorhandenen serialisierten Elements kopiert. Falls `Copy` auf den Wert *'false'* gesetzt ist, wird nur der Datenzeiger des übergebenen serialisierten Elements gespeichert ohne die Daten in ein neues serialisiertes Element zu kopieren. In diesem Fall erhöht sich die Geschwindigkeit des Operators.

Achtung

Falls der Parameter `Copy` auf den Wert *'false'* gesetzt ist, darf der zum Datenzeiger des serialisierten Elements gehörende Speicher während der Nutzung des neuen serialisierten Elements nicht gelöscht werden.

Parameter

- ▷ **`Pointer`** (input_control) . . . . . pointer ⇝ *integer*
  Datenzeiger auf ein serialisiertes Element.
- ▷ **`Size`** (input_control) . . . . . integer ⇝ *integer*
  Größe des serialisierten Elements.
- ▷ **`Copy`** (input_control) . . . . . string ⇝ *string*
  Kopiermodus des serialisierten Elements.
  Defaultwert: 'true'
  Werteliste: `Copy` ∈ {'true', 'false'}
- ▷ **`SerializedItemHandle`** (output_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_serialized_item_ptr` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`get_serialized_item_ptr`

Modul

Foundation

fread_serialized_item (: : FileHandle : SerializedItemHandle)

Einlesen eines serialisierten Elements aus einer Datei.

`fread_serialized_item` liest ein serialisiertes Element, welches mit `fwrite_serialized_item` geschrieben wurde, aus der aktuellen Eingabedatei mit dem Datei-Handle `FileHandle` ein (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Hierzu wird ein serialisiertes Element erzeugt und die aus der Datei gelesenen Werte darin gespeichert. Der Operator gibt das Handle des serialisierten Elements in dem Parameter `SerializedItemHandle` zurück. Mit dem Operator `open_file` kann die Datei geöffnet werden. Zu beachten ist, dass die Datei im Binärformat geöffnet werden muss. Um mehrere serialisierte Elemente aus einer einzigen Eingabedatei zu lesen, muss `fread_serialized_item` mehrmals aufgerufen werden.

Parameter

- ▷ **FileHandle** (input_control) ..... file ⇝ *handle*
  Datei-Handle.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `fread_serialized_item` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Der Fehlercode 9581 bedeutet hierbei, dass das Dateiende erreicht wird, bevor ein serialisiertes Element gelesen werden kann.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`open_file`, `fwrite_serialized_item`

Nachfolger

`close_file`, `get_serialized_item_ptr`, `deserialize_shape_model`, `deserialize_ncc_model`

Modul

Foundation

fwrite_serialized_item (: : FileHandle, SerializedItemHandle :)

Schreiben eines serialisierten Elements in eine Datei.

`fwrite_serialized_item` schreibt ein serialisiertes Element in die Ausgabedatei mit dem Datei-Handle `FileHandle`. Das serialisierte Element wird durch das Handle `SerializedItemHandle` übergeben.

Mit dem Operator open_file kann die Datei geöffnet werden. Zu beachten ist, dass sie im Binärdateiformat geöffnet werden muss. Um mehrere serialisierte Elemente in eine einzige Ausgabedatei zu schreiben, muss fwrite_serialized_item mehrmals aufgerufen werden. Ein serialisiertes Element kann mit fread_serialized_item wieder aus der Datei eingelesen werden.

Grundlagen der Serialisierung und Deserialisierung

Um ikonische Objekte, Daten oder Modelle in einer Datei abspeichern zu können oder zu einem anderen HALCON-Prozess über die Socket-Verbindung schicken zu können müssen die ikonischen Objekte, Daten oder Modelle zunächst in serialisierte Elemente überführt werden. Zu diesem Zweck stehen eine Vielzahl Operatoren zur Verfügung. Beispielsweise gibt es für ein Formmodell den Operator serialize_shape_model, der die Serialisierung durchführt. Ein serialisiertes Element kann dann mit fwrite_serialized_item in eine Datei geschrieben oder mit send_serialized_item zu einem anderen HALCON-Prozess über die Socket-Verbindung übertragen werden.

Um ein serialisiertes Element zu deserialisieren wird zunächst ein serialisiertes Element aus einer Datei (siehe fread_serialized_item) eingelesen oder über die Socket-Verbindung von einem anderen HALCON-Prozess empfangen (siehe receive_serialized_item). Um ein serialisiertes Element wieder in seine ursprüngliche Form, d.h. in das ikonische Objekt, die Daten oder das Modell zu überführen, gibt es ebenfalls Operatoren. Diese nehmen die Deserialisierung vor und erzeugen automatisch das jeweilige ikonische Objekt, die Daten oder das Modell. Beispielsweise gibt es für ein serialisiertes Formmodell den Operator deserialize_shape_model, der die Deserialisierung durchführt.

Parameter

- ▷ **FileHandle** (input_control) ..... file ⇝ *handle*
  Datei-Handle.
- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert fwrite_serialized_item den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_file, serialize_shape_model, serialize_ncc_model

Nachfolger

close_file, fread_serialized_item

Modul

Foundation

get_serialized_item_ptr (: : SerializedItemHandle : Pointer, Size)

Zugreifen auf den Datenzeiger eines serialisierten Elements.

Es wird nicht empfohlen, den Operator get_serialized_item_ptr in HDevelop zu verwenden.

get_serialized_item_ptr gibt den Datenzeiger auf das serialisierte Element zurück, welches in dem Handle SerializedItemHandle übergeben wird (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Der Datenzeiger, d.h. der Anfang des serialisierten Elements, wird im Parameter Pointer zurückgegeben. Der Parameter Size gibt die Größe des serialisierten Elements in Byte zurück.

Parameter

- ▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **Pointer** (output_control) . . . . . pointer ⇝ *integer*
  Datenzeiger auf das serialisierte Element.
- ▷ **Size** (output_control) . . . . . integer ⇝ *integer*
  Größe des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_serialized_item_ptr` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_serialized_item_ptr`

Modul

Foundation

25.11 Seriell

clear_serial (: : SerialHandle, Channel :)

Löschen des Puffers einer seriellen Verbindung.

`clear_serial` löscht Daten, die an das serielle Gerät, das durch `SerialHandle` bezeichnet wird, geschickt, aber noch nicht übertragen worden sind (`Channel` = *'output'*), oder löscht Daten, die bereits empfangen, aber noch nicht gelesen worden sind (`Channel` = *'input'*), oder führt beide Operationen gleichzeitig aus (`Channel` = *'in_out'*).

Parameter

- ▷ **SerialHandle** (input_control) . . . . . serial ⇝ *handle*
  Handle der seriellen Schnittstelle.
- ▷ **Channel** (input_control) . . . . . string ⇝ *string*
  Zu löschender Puffer.
  Defaultwert: 'input'
  Werteliste: `Channel` ∈ {'input', 'output', 'in_out'}

Ergebnis

Wenn die Parameter korrekt sind und die Puffer des seriellen Geräts gelöscht werden konnten, liefert `clear_serial` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SerialHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_serial`

Nachfolger

`read_serial`, `write_serial`

Siehe auch

`read_serial`

Modul

Foundation

close_serial (: : SerialHandle :)

Schließen eines seriellen Gerätes.

`close_serial` schließt ein serielles Gerät, das mit `open_serial` geöffnet wurde.

Parameter

▷ **SerialHandle** (input_control) . . . . . serial ⇝ *handle*
Handle der seriellen Schnittstelle.

Ergebnis

Wenn die Parameter korrekt sind und das angegebene Gerät geschlossen werden konnte, liefert `close_serial` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SerialHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_serial`

Siehe auch

`open_serial`, `close_file`

Modul

Foundation

get_serial_param (: : SerialHandle : BaudRate, DataBits, FlowControl, Parity, StopBits, TotalTimeOut, InterCharTimeOut)

Auslesen der Parameter eines seriellen Geräts.

`get_serial_param` liefert die aktuellen Parameter des seriellen Geräts, das in `SerialHandle` übergeben wird, zurück. Für eine Beschreibung der Parameter der seriellen Schnittstelle siehe `set_serial_param`.

Parameter

- ▷ **SerialHandle** (input_control) . . . . . serial ⇝ *handle*
  Handle der seriellen Schnittstelle.
- ▷ **BaudRate** (output_control) . . . . . integer ⇝ *integer*
  Übertragungsgeschwindigkeit der seriellen Schnittstelle.
- ▷ **DataBits** (output_control) . . . . . integer ⇝ *integer*
  Anzahl Datenbits der seriellen Schnittstelle.
- ▷ **FlowControl** (output_control) . . . . . string ⇝ *string*
  Datenflusskontrolle der seriellen Schnittstelle.
- ▷ **Parity** (output_control) . . . . . string ⇝ *string*
  Parität der seriellen Schnittstelle.
- ▷ **StopBits** (output_control) . . . . . integer ⇝ *integer*
  Anzahl Stopbits der seriellen Schnittstelle.
- ▷ **TotalTimeOut** (output_control) . . . . . integer ⇝ *integer*
  Gesamt-Timeout der seriellen Schnittstelle in ms.
- ▷ **InterCharTimeOut** (output_control) . . . . . integer ⇝ *integer*
  Inter-character Timeout der seriellen Schnittstelle in ms.

Ergebnis

Wenn die Parameter korrekt sind und die Parameter des Gerätes gelesen werden konnten, liefert `get_serial_param` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SerialHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_serial`

Nachfolger

`read_serial`, `write_serial`

Siehe auch

`set_serial_param`

Modul

Foundation

open_serial (: : PortName : SerialHandle)

Öffnen eines seriellen Gerätes.

`open_serial` öffnet ein serielles Gerät. Der Name des Geräts wird in `PortName`. Er ist Betriebssystem-abhängig: auf Windows-Rechnern wird typischerweise *'COM1'-'COM4'* verwendet, während die seriellen Schnittstellen unter unixartigen Systemen normalerweise *'/dev/tty*'* heißen. Die Parameter der seriellen Schnittstelle, wie die Übertragungsgeschwindigkeit oder die Anzahl der Datenbits, sind nach dem Öffnen auf die Standardwerte des jeweiligen Gerätes gesetzt. Sie können mit `set_serial_param` gesetzt oder verändert werden.

Parameter

▷ **PortName** (input_control) . . . . . filename ⇝ *string*
Name der seriellen Schnittstelle.
Defaultwert: 'COM1'
Wertevorschläge: PortName ∈ {'COM1', 'COM2', 'COM3', 'COM4', '/dev/ttya', '/dev/ttyb', '/dev/tty00', '/dev/tty01', '/dev/ttyd1', '/dev/ttyd2', '/dev/cua0', '/dev/cua1'}

▷ **SerialHandle** (output_control) . . . . . serial ⇝ *handle*
Handle der seriellen Schnittstelle.

Ergebnis

Wenn die Parameter korrekt sind und das angegebene Gerät geöffnet werden konnte, liefert open_serial den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

set_serial_param, read_serial, write_serial, close_serial

Siehe auch

set_serial_param, get_serial_param, open_file

Modul

Foundation

read_serial (: : SerialHandle, NumCharacters : Data)

Lesen von einem seriellen Gerät.

read_serial versucht, NumCharacters von dem seriellen Gerät, das durch SerialHandle angegeben wird, zu lesen. Die gelesenen Daten werden in Data als Tupel von Integer-Werten zurückgegeben. Dies erlaubt das Lesen von NUL-Zeichen, die sonst als String-Ende interpretiert würden. Falls der Timeout des Gerätes mit set_serial_param auf einen Wert größer als 0 gesetzt wurde, wird so lange auf die Ankunft des ersten Zeichens gewartet, wie durch den Timeout angegeben ist, ansonsten kehrt die Funktion sofort zurück. In jedem Fall werden die zur Zeit der Rückkehr verfügbaren Zeichen zurückgeliefert. D.h., es können weniger Zeichen als angefordert zurückgeliefert werden. Dies kann über die Länge des Tupels Data überprüft werden.

Parameter

▷ **SerialHandle** (input_control) . . . . . serial ⇝ *handle*
Handle der seriellen Schnittstelle.

▷ **NumCharacters** (input_control) . . . . . integer ⇝ *integer*
Anzahl der zu lesenden Zeichen.
Defaultwert: 1
Wertevorschläge: NumCharacters ∈ {1, 2, 3, 4, 5, 10, 20, 40, 100}

▷ **Data** (output_control) . . . . . integer(-array) ⇝ *integer*
Gelesene Zeichen (als Integer-Tupel).

Ergebnis

Wenn die Parameter korrekt sind und das Lesen vom seriellen Gerät erfolgreich war, liefert read_serial den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_serial`

Siehe auch

`write_serial`

Modul

Foundation

`set_serial_param` `( : : SerialHandle, BaudRate, DataBits, FlowControl, Parity, StopBits, TotalTimeOut, InterCharTimeOut : )`

Setzen der Parameter eines seriellen Gerätes.

Mit `set_serial_param` können die Parameter eines seriellen Gerätes gesetzt werden. Der Parameter `BaudRate` gibt die Ein- und Ausgabegeschwindigkeit des Gerätes an. Es ist zu beachten, dass nicht jedes Gerät alle Geschwindigkeiten unterstützt. Die Anzahl der übertragenen Datenbits wird mit `DataBits` festgelegt. Der Parameter `FlowControl` legt fest, ob und welche Art von Datenflusskontrolle verwendet werden. Software-Kontrolle (*'xon_xoff'*) und Hardware-Kontrolle (*'cts_rts'*, *'dtr_dsr'* können gewählt werden. Mehrere Werte können hierfür innerhalb einer Zeichenkette gesetzt werden, durch Leerzeichen getrennt (z.B. *'cts_rts dtr_dsr'*). Ob und welche Art von Paritätskontrolle der übertragenen Daten verwendet wird, lässt sich mit `Parity` festlegen. Die Anzahl der übertragenen Stop-Bits kann mit `StopBits` gesetzt werden. Schließlich können noch zwei Timeouts für das Lesen von dem seriellen Gerät gesetzt werden. Der Parameter `TotalTimeOut` bezieht sich auf die Zeit, die bei `read_serial` bis zum Eintreffen des ersten Zeichens, unabhängig von der Anzahl der insgesamt zu lesenden Zeichen, höchstens vergehen darf. Der Parameter `InterCharTimeOut` gibt die Zeit an, die zwischen dem Lesen einzelner Zeichen verstreichen darf, falls mehrere Zeichen mit `read_serial` gelesen werden sollen. Falls einer der beiden Timeouts auf *-1* gesetzt wird, so wird beliebig lange auf das Eintreffen eines Zeichens gewartet. Werden beide Timeouts auf *0* gesetzt, wartet das System nicht, sondern liest die vorhandenen oder keine Zeichen. Auf Windows-Systemen ergibt sich so ein maximaler Timeout von $\texttt{TotalTimeOut} + n\texttt{InterCharTimeOut}$, wenn n Zeichen gelesen werden sollen. Auf unixartigen Systemen kann systembedingt nur einer der beiden Timeouts gesetzt werden. Wenn beide Timeouts größer als -1 übergeben werden, wird nur der Total-Timeout beachtet. Die Einheit der beiden Timeouts ist Millisekunden. Es ist jedoch zu beachten, dass der Timeout unter unixartigen Systemen nur in Zehntelsekundenschritten spezifizieren lässt, d.h. der minimale wirksame Timeout ist *100*. Für jeden dieser Parameter kann die aktuelle Einstellung beibehalten werden, indem *'unchanged'* übergeben wird.

Parameter

▷ **`SerialHandle`** (input_control) . . . . . serial ⇝ *handle*
Handle der seriellen Schnittstelle.

▷ **`BaudRate`** (input_control) . . . . . integer ⇝ *integer* / string
Übertragungsgeschwindigkeit der seriellen Schnittstelle.
Defaultwert: 'unchanged'
Werteliste: `BaudRate` ∈ {50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200, 153600, 230400, 307200, 460800, 'unchanged'}

▷ **`DataBits`** (input_control) . . . . . integer ⇝ *integer* / string
Anzahl Datenbits der seriellen Schnittstelle.
Defaultwert: 'unchanged'
Werteliste: `DataBits` ∈ {5, 6, 7, 8, 'unchanged'}

▷ **`FlowControl`** (input_control) . . . . . string ⇝ *string*
Datenflusskontrolle der seriellen Schnittstelle.
Defaultwert: 'unchanged'
Werteliste: `FlowControl` ∈ {'none', 'cts_rts', 'dtr_dsr', 'xon_xoff', 'dtr_dsr xon_xoff', 'cts_rts xon_xoff', 'cts_rts dtr_dsr', 'cts_rts dtr_dsr xon_xoff', 'unchanged'}

▷ **Parity** (input_control) ..... string ⇝ *string*
Parität der seriellen Schnittstelle.
Defaultwert: 'unchanged'
Werteliste: `Parity` ∈ {'none', 'odd', 'even', 'unchanged'}

▷ **StopBits** (input_control) ..... integer ⇝ *integer* / string
Anzahl Stopbits der seriellen Schnittstelle.
Defaultwert: 'unchanged'
Werteliste: `StopBits` ∈ {1, 2, 'unchanged'}

▷ **TotalTimeOut** (input_control) ..... integer ⇝ *integer* / string
Gesamt-Timeout der seriellen Schnittstelle in ms.
Defaultwert: 'unchanged'
Wertevorschläge: `TotalTimeOut` ∈ {-1, 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 'unchanged'}

▷ **InterCharTimeOut** (input_control) ..... integer ⇝ *integer* / string
Inter-character Timeout der seriellen Schnittstelle in ms.
Defaultwert: 'unchanged'
Wertevorschläge: `InterCharTimeOut` ∈ {-1, 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 'unchanged'}

Ergebnis

Wenn die Parameter korrekt sind und die Parameter des Gerätes gesetzt werden konnten, liefert `set_serial_param` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- SerialHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_serial`, `get_serial_param`

Nachfolger

`read_serial`, `write_serial`

Siehe auch

`get_serial_param`

Modul

Foundation

write_serial (: : SerialHandle, Data :)

Schreiben in eine seriellen Verbindung.

`write_serial` schreibt die Zeichen, die in `Data` übergeben werden, auf das serielle Gerät, das durch `SerialHandle` angegeben wird. Die zu schreibenden Daten werden als Tupel von Integer-Werten übergeben. Dies erlaubt das Schreiben von NUL-Zeichen, die sonst als String-Ende interpretiert würden. Beim Schreiben wird immer gewartet, bis die Übertragung erfolgt ist, d.h., es kann kein Timeout gesetzt werden.

Parameter

▷ **SerialHandle** (input_control) . . . . . serial ⇝ *handle*
Handle der seriellen Schnittstelle.

▷ **Data** (input_control) . . . . . integer(-array) ⇝ *integer*
Zu schreibende Zeichen (als Integer-Tupel).

Ergebnis

Wenn die Parameter korrekt sind und das Schreiben zum seriellen Gerät erfolgreich war, liefert `write_serial` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_serial`

Siehe auch

`read_serial`

Modul

Foundation

25.12 Sockets

close_socket (: : Socket :)

Schließen einer Socket-Verbindung.

`close_socket` schließt eine Socket-Verbindung, die zuvor mit `open_socket_accept`, `open_socket_connect` oder `socket_accept_connect` geöffnet worden ist. Für ein ausführliches Beispiel siehe `open_socket_accept`.

Parameter

▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- Socket

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`open_socket_accept`, `open_socket_connect`, `socket_accept_connect`

Modul

Foundation

get_next_socket_data_type (: : Socket : DataType)

Abfragen des nächsten an einem Socket anliegenden HALCON-Datentyps.

get_next_socket_data_type liefert den Datentyp der nächsten am Socket Socket anliegenden Daten in DataType zurück. Die möglichen Werte für DataType sind:

'no_data': Es liegen keine Daten an.

'no_halcon_data': Es liegen Daten an, sie sind aber keine HALCON-Daten.

'tuple': Die nächsten Daten sind ein Tupel.

'region': Die nächsten Daten sind ein Regionenobjekt.

'image': Die nächsten Daten sind ein Bildobjekt.

'xld_cont': Die nächsten Daten sind XLD-Konturen.

'xld_poly': Die nächsten Daten sind XLD-Polygone.

'xld_para': Die nächsten Daten sind XLD-Parallelen.

'xld_mod_para': Die nächsten Daten sind modifizierte XLD-Parallelen.

'xld_ext_para': Die nächsten Daten sind erweiterte XLD-Parallelen.

'serialized_item': Die nächsten Daten sind ein serialisiertes Element.

Parameter

▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

▷ **DataType** (output_control) . . . . . string ⇝ *string*
Anliegender HALCON-Datentyp.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

send_image, receive_image, send_region, receive_region, send_tuple, receive_tuple, send_serialized_item, receive_serialized_item

Modul

Foundation

get_socket_descriptor (: : Socket : SocketDescriptor)

Rückgabe des Socket-Deskriptors einer Socket-Verbindung.

get_socket_descriptor liefert den Socket-Deskriptor, der vom Betriebssystem verwendet wird, für die Socket-Verbindung, die in Socket übergeben wird, zurück. Der Socket-Deskriptor kann in Betriebssystemaufrufen wie select, read, write, recv oder send verwendet werden.

Parameter

▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

▷ **SocketDescriptor** (output_control) . . . . . integer ⇝ *integer*
Socket-Deskriptor des Betriebssystems.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_accept, open_socket_connect, socket_accept_connect

Siehe auch

set_socket_param

Modul

Foundation

get_socket_param (: : Socket, GenParamName : GenParamValue)

Auslesen von Socket-Einstellungen.

get_socket_param liest den Wert GenParamValue der mit GenParamName bezeichneten Einstellung für die angegebene Socket-Verbindung aus. Verfügbare Parameter sind *'timeout'*, *'address_info'*, *'SO_SNDBUF'*, *'SO_RCVBUF'*, *'SO_BROADCAST'* und *'TCP_NODELAY'*.

Mittels *'address_info'* erhält man die IP Adresse und den Netzwerkport der lokalen und entfernten Verbindungsendpunkte des angegebenen Sockets.

Die Beschreibung der restlichen Parameter finden Sie in der Dokumentation des Operators set_socket_param.

Parameter

▷ **Socket** (input_control) ..... socket ⇝ *handle*
Nummer des Sockets.

▷ **GenParamName** (input_control) ..... string(-array) ⇝ *string*
Name der Socket-Einstellung.
Werteliste: GenParamName ∈ {'timeout', 'address_info', 'SO_SNDBUF', 'SO_RCVBUF', 'SO_BROADCAST', 'TCP_NODELAY'}

▷ **GenParamValue** (output_control) ..... string ⇝ *string* / real / integer
Wert der Socket-Einstellung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect

Nachfolger

set_socket_param

Siehe auch

set_socket_param

Modul

Foundation

open_socket_accept (: : Port, GenParamName, GenParamValue : AcceptingSocket)

Öffnen eines Sockets, der Verbindungsanfragen zulässt.

open_socket_accept öffnet einen Socket, der ankommende Verbindungsanfragen akzeptiert. Dieser Operator ist der notwendige erste Schritt, um einen Kommunikationskanal zwischen zwei HALCON-Prozessen aufzubauen. Der Socket horcht nach ankommenden Verbindungsanfragen auf dem durch Port bestimmten Port.

Als Port kann auch der Wert 0 angegeben werden. Die genauen Auswirkungen sind Implementierungs-spezifisch. Bei den meisten Betriebssystemen bedeutet eine 0, dass das Betriebssystem eine freie Portnummer selber per Zufall bestimmen soll.

Mit dem Parameter *'protocol'* kann das verwendete Kommunikationsprotokoll spezifiziert werden. Voreingestellt ist das *'HALCON'* Protokoll ohne festgelegte Adressfamilie. Somit wird entsprechend den aktuellen Netzwerkeinstellungen des Rechners entweder IPv4 oder IPv6 automatisch verwendet. Um eine bestimmte Adressfamilie auszuwählen, muss eine '4' für IPv4 bzw. eine '6' für IPv6 an den Namen des Protokolls angehängt werden. Z.B. bezeichnet *'HALCON4'* eine HALCON Verbindung über IPv4. Mögliche Werte für eine generische Socketkommunikation sind *'UDP'* und *'TCP'*. Auch hier kann eine '4' oder '6' angehängt werden. Beide Kommunikationspartner müssen dasselbe Protokoll verwenden. Der Datenaustausch über generische Sockets erfolgt ausschließlich mit send_data und receive_data.

Mit dem generischen Parameter *'address'* kann das Socket angewiesen werden, nur Verbindungen zu akzeptieren, die an eine bestimme Adresse gerichtet sind. So würde der Wert *'localhost'* das Socket anweisen, nur Verbindungen zu akzeptieren, die an die Netzwerkaddresse *'localhost'* gerichtet sind, welche normalerweise der lokalen Loopback-Schnittstelle *'127.0.0.1'* entspricht.

Mit dem generischen Parameter *'timeout'* kann ein Timeout für diesen Socket angegeben werden. Er wird in der Einheit Sekunden oder als Zeichenkette *'infinite'* angegeben und ist im Besonderen dazu bestimmt, den Timeout und Wartemodus für einen folgenden socket_accept_connect Operatoraufruf, bei dem der Parameter Wait auf *'auto'* gesetzt ist, zu setzen.

Mit dem generischen Parameter *'string_encoding'* kann die Kodierung für das Senden und Empfangen von Zeichenketten angegeben werden. Die Kodierung wird für send_tuple, receive_tuple, send_data und receive_data verwendet. Wenn das Kommunikationsprotokoll *'HALCON'* verwendet wird, hat diese Einstellung nur Auswirkungen auf Verbindungen mit HALCON Versionen vor 18.11. Wenn der Kommunikationspartner eine HALCON Version 18.11 oder neuer verwendet, wird UTF-8 bei der Kodierung von Tupeln verwendet.

Mit dem generischen Parameter *'reuseaddr'* kann die Einstellung der Socket Option SO_REUSEADDR kontrolliert werden. Der Standardwert ist *'true'*. Hat *'reuseaddr'* den Wert *'false'*, so wird SO_REUSEADDR nicht gesetzt. Auf Windows Systemen werden konkurrierende Zugriffe auf einem einzelnen Anschluss nur detektiert, wenn *'reuseaddr'* auf den Wert *'false'* gesetzt ist.

Mit dem generischen Parameter *'tls_enable'* kann die Nutzung der Transport Layer Security (TLS) für den Socket kontrolliert werden. Der Standardwert ist *'false'*. Hat *'tls_enable'* den Wert *'true'*, so wird ein Socket für die Annahme TLS-Verschlüsselter Verbindungen erzeugt. Bei eingeschalteter TLS-Option müssen weiterhin die generischen Parameter *'tls_private_key'* und *'tls_certificate'* gesetzt sein. Mit *'tls_private_key'* wird der Pfad zur Datei mit dem privaten Zertifikats-Schlüssel gesetzt, in *'tls_certificate'* wird der Pfad zum öffentlichen Server-Zertifikat angegeben. TLS-Zertifikat und privater Schlüssel müssen im PEM-Format vorliegen.

Der akzeptierende Socket wird in AcceptingSocket zurückgegeben. open_socket_accept kehrt sofort zurück, ohne auf eine Verbindungsanfrage eines anderen Prozesses zu warten, die durch den Aufruf von open_socket_connect eingeleitet wird. Dies ermöglicht es mehreren Prozessen mit dem Prozess, der open_socket_accept aufruft, eine Verbindung herzustellen. Um eine ankommende *'HALCON'* oder *'TCP'* Verbindungsanfrage zu akzeptieren, muss socket_accept_connect aufgerufen werden, so dass ein Socket für die eigentliche Kommunikation zur Verfügung steht.

Parameter

▷ **Port** (input_control) . . . . . number ⇝ *integer*
Nummer des Ports.
Defaultwert: 3000
Wertevorschläge: Port ∈ {3000, 4570}
Typischer Wertebereich: 1024 ≤ Port ≤ 65535
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für die Socket-Verbindung eingestellt werden sollen.
Defaultwert: []
Werteliste: GenParamName ∈ {'address', 'protocol', 'timeout', 'string_encoding', 'tls_enable', 'tls_private_key', 'tls_certificate'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / real / integer
Werte der generischen Parameter, die für die Socket-Verbindung eingestellt werden sollen.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0, 3.0, 'infinite', 'HALCON', 'UDP', 'TCP', 'HALCON4', 'UDP4', 'TCP4', 'HALCON6', 'UDP6', 'TCP6', 'utf8', 'locale', 'ignore', 'true', 'false'}

▷ **AcceptingSocket** (output_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

Beispiel

```
* Process 1
dev_set_colored (12)
open_socket_accept (3000, [], [], AcceptingSocket)
* Busy wait for an incoming connection
dev_error_var (Error, 1)
dev_set_check ('~give_error')
OpenStatus := 5
while (OpenStatus != 2)
    socket_accept_connect (AcceptingSocket, 'false', Socket)
    OpenStatus := Error
    wait_seconds (0.2)
endwhile
dev_set_check ('give_error')
* Connection established
receive_image (Image, Socket)
threshold (Image, Region, 0, 63)
send_region (Region, Socket)
receive_region (ConnectedRegions, Socket)
area_center (ConnectedRegions, Area, Row, Column)
send_tuple (Socket, Area)
send_tuple (Socket, Row)
send_tuple (Socket, Column)
close_socket (Socket)
close_socket (AcceptingSocket)

* Process 2
dev_set_colored (12)
open_socket_connect ('localhost', 3000, [], [], Socket)
read_image (Image, 'fabrik')
send_image (Image, Socket)
receive_region (Region, Socket)
connection (Region, ConnectedRegions)
send_region (ConnectedRegions, Socket)
receive_tuple (Socket, Area)
receive_tuple (Socket, Row)
receive_tuple (Socket, Column)
close_socket (Socket)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

socket_accept_connect

Siehe auch

open_socket_connect, close_socket, get_socket_param, set_socket_param, send_image, receive_image, send_region, receive_region, send_tuple, receive_tuple, send_data, receive_data

Modul

Foundation

open_socket_connect (: : HostName, Port, GenParamName, GenParamValue : Socket)

Öffnen einer Socket-Verbindung zu einem bestehenden Socket.

open_socket_connect öffnet eine Socket-Verbindung zu einem akzeptierenden Socket auf dem Rechner mit dem Namen HostName, der auf dem Port Port nach Verbindungsanfragen horcht.

Mit dem Parameter *'protocol'* kann das verwendete Kommunikationsprotokoll spezifiziert werden. Voreingestellt ist das *'HALCON'* Protokoll ohne festgelegte Adressfamilie. Somit wird entsprechend den aktuellen Netzwerkeinstellungen des Rechners entweder IPv4 oder IPv6 automatisch verwendet. Um eine bestimmte Adressfamilie auszuwählen, muss eine '4' für IPv4 bzw. eine '6' für IPv6 an den Namen des Protokolls angehängt werden. Z.B. bezeichnet *'HALCON4'* eine HALCON Verbindung über IPv4. Mögliche Werte für eine generische Socketkommunikation sind *'UDP'* und *'TCP'*. Auch hier kann eine '4' oder '6' angehängt werden. Auch die Verwendung von IPv4 (z.B. '127.0.0.1') bzw. IPv6 (z.B. '::1') Adressen für 'HostName' legt die verwendete Adressfamilie fest. Beide Kommunikationspartner müssen dasselbe Protokoll verwenden. Der Datenaustausch über generische Sockets erfolgt ausschließlich mit send_data und receive_data.

Mit dem generischen Parameter *'timeout'* kann ein Timeout für diesen Socket angegeben werden. Er wird in der Einheit Sekunden oder als Zeichenkette *'infinite'* angegeben.

Mit dem generischen Parameter *'tls_enable'* kann die Nutzung der Transport Layer Security (TLS) für den Socket kontrolliert werden. Der Standardwert ist *'false'*. Hat *'tls_enable'* den Wert *'true'*, so wird Transport Layer Security (TLS) zur Verschlüsselung der Socket-Verbindung genutzt. Der akzeptierenden Socket muss zuvor mit open_socket_accept und ebenfalls eingeschaltetem *'tls_enable'* erzeugt worden sein. Mit dem generischen Parameter *'tls_certificate'* kann der Pfad zu einem Zertifikat angegeben werden, um die Authentizität des Servers zu überprüfen. Ohne TLS-Zertifikat ist die Verbindung zwar verschlüsselt, die Authentizität des Servers kann allerdings nicht überprüft werden. Das TLS-Zertifikat muss im PEM-Format vorliegen.

Für das *'HALCON'* Protokoll muss der akzeptierende Socket zuvor in einem anderen HALCON-Prozess mit open_socket_accept erzeugt worden sein. Um eine Verbindung herzustellen, muss der HALCON-Prozess, in dem der akzeptierende Socket läuft, socket_accept_connect aufrufen. Für ein ausführliches Beispiel siehe open_socket_accept.

Bei der Verwendung generischer Sockets kann der Socket, mit dem man sich verbindet, ein beliebiger Socket vom gleichen Protokolltyp sein.

Mit dem generischen Parameter *'string_encoding'* kann die Kodierung für das Senden und Empfangen von Zeichenketten angegeben werden. Die Kodierung wird für send_tuple, receive_tuple, send_data und receive_data verwendet. Wenn das Kommunikationsprotokoll *'HALCON'* verwendet wird, hat diese Einstellung nur Auswirkungen auf Verbindungen mit HALCON Versionen vor 18.11. Wenn der Kommunikationspartner eine HALCON Version 18.11 oder neuer verwendet, wird UTF-8 bei der Kodierung von Tupeln verwendet.

Parameter

▷ **HostName** (input_control) . . . . . string ⇝ *string*
Name des Rechners, zu dem die Verbindung aufgebaut werden soll.
Defaultwert: 'localhost'

▷ **Port** (input_control) . . . . . number ⇝ *integer*
Nummer des Ports.
Wertevorschläge: Port ∈ {3000, 4570}
Typischer Wertebereich: 1024 ≤ Port ≤ 65535
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Namen der generischen Parameter, die für die Socket-Verbindung eingestellt werden sollen.
Defaultwert: []
Werteliste: GenParamName ∈ {'timeout', 'protocol', 'string_encoding', 'tls_enable', 'tls_certificate'}

▷ **GenParamValue** (input_control) . . . . . attribute.value(-array) ⇝ *string* / real / integer
Werte der generischen Parameter, die für die Socket-Verbindung eingestellt werden sollen.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0, 3.0, 'infinite', 'HALCON', 'UDP', 'TCP', 'HALCON4', 'UDP4', 'TCP4', 'HALCON6', 'UDP6', 'TCP6', 'utf8', 'locale', 'ignore', 'true', 'false'}

▷ **Socket** (output_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

send_image, receive_image, send_region, receive_region, send_tuple, receive_tuple, send_data, receive_data

Siehe auch

open_socket_accept, socket_accept_connect, get_socket_param, set_socket_param, close_socket

Modul

Foundation

receive_data (: : Socket, Format : Data, From)

Empfangen von Daten über eine generische Socket-Verbindung auch von externen Geräten oder Anwendungen.

receive_data liest Daten über die durch Socket bestimmte generische Socket-Verbindung. Die Daten werden anhand des Parameters Format aus einem binären Netzwerkpaket in einen Wert oder ein Tupel von Werten umgewandelt und sind gut zur Kommunikation mit externen Geräten oder Anwendungen geeignet. Dieser Operator unterstützt nicht das Standard *'HALCON'* Protokoll, sondern ist zum Empfang beliebiger Daten gedacht. Mit dem Operator send_data können entsprechende Daten mit einem HALCON Prozess gesendet werden.

Die empfangenen Daten werden mittels des Parameters Format in Werte oder Tupel von Werten konvertiert. Es können mehrere Formate angeben werden, in diesem Fall enthält der From Parameter einen dritten Wert, welcher den Index des verwendeten Formats angibt. Um ein passendes Format zu finden, wird die Größe des notwendigen Buffers initial berechnet. Beim Empfang von Daten wird das erste Format verwendet, welches eine passende Größe spezifiziert, um die binären Daten in Werte zu verwandeln.

Der Parameter From enthält die IP Adresse oder den Netzwerknamen und den Netzwerkport des Kommunikationspartners. Bei UDP Verbindungen können diese Werte für eine Antwort mittels send_data benutzt werden.

Eine detaillierte Beschreibung des Formats finden Sie in der Dokumentation des Operators send_data.

Parameter

▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

▷ **Format** (input_control) . . . . . string(-array) ⇝ *string*
Vorschrift zur Konvertierung der Daten.
Defaultwert: 'z'

▷ **Data** (output_control) . . . . . string ⇝ *string* / real / integer
Wert (oder Tupel von Werten) mit den empfangenen und konvertierten Daten.

▷ **From** (output_control) . . . . . string ⇝ *string* / integer
IP-Adresse oder Netzwerkname und Netzwerkport der Gegenstelle.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_socket_connect`, `socket_accept_connect`, `get_socket_param`, `set_socket_param`

Nachfolger

`close_socket`

Siehe auch

`send_data`

Modul

Foundation

receive_image (: Image : Socket :)

Empfangen von Bildern über eine Socket-Verbindung.

`receive_image` liest ein Bildobjekt, das von einem anderen HALCON-Prozess über die Socket-Verbindung `Socket` mit dem Operator `send_image` geschickt worden ist. Wenn noch kein Bild verschickt worden sind, wird der Prozess, der `receive_image` aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Für ein ausführliches Beispiel siehe `open_socket_accept`.

Achtung

'int8' Bilder können nur von 64-Bit-Systemen empfangen werden!

Parameter

▷ **Image** (output_object) . . . . . image(-array) ⇝ *object* : byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Empfangenes Bild.

▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_socket_connect`, `socket_accept_connect`, `get_socket_param`, `set_socket_param`

Siehe auch

`send_image`, `send_region`, `receive_region`, `send_tuple`, `receive_tuple`, `get_next_socket_data_type`

Modul

Foundation

receive_region (: Region : Socket :)

Empfangen von Regionen über eine Socket-Verbindung.

receive_region liest ein Regionenobjekt, das von einem anderen HALCON-Prozess über die Socket-Verbindung Socket mit dem Operator send_region geschickt worden ist. Wenn noch keine Regionen verschickt worden sind, wird der Prozess, der receive_region aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

▷ **Region** (output_object) ..... region(-array) ⇝ *object*
Empfangene Regionen.

▷ **Socket** (input_control) ..... socket ⇝ *handle*
Nummer des Sockets.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect, get_socket_param, set_socket_param

Siehe auch

send_region, send_image, receive_image, send_tuple, receive_tuple, get_next_socket_data_type

Modul

Foundation

receive_serialized_item (: : Socket : SerializedItemHandle)

Empfangen eines serialisierten Elements über eine Socket-Verbindung.

receive_serialized_item liest ein serialisiertes Element, welches von einem anderen HALCON-Prozess über die Socket-Verbindung Socket mit dem Operator send_serialized_item geschickt worden ist (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Wenn noch kein serialisiertes Element verschickt worden ist, wird der Prozess, der receive_serialized_item aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Um die Daten zu speichern wird eine neues serialisiertes Element erzeugt und die Daten in diesem gespeichert. Der Operator gibt das Handle des serialisierten Elementes in dem Parameter SerializedItemHandle zurück. Für ein ausführliches Beispiel über die Nutzung einer Socket-Verbindung siehe open_socket_accept.

Parameter

▷ **Socket** (input_control) ..... socket ⇝ *handle*
Nummer des Sockets.

▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert receive_serialized_item den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

send_serialized_item

Nachfolger

get_serialized_item_ptr, deserialize_matrix, deserialize_metrology_model

Modul

Foundation

receive_tuple (: : Socket : Tuple)

Empfangen eines Tupels über eine Socket-Verbindung.

receive_tuple liest ein Tupel, das von einem anderen HALCON-Prozess über die Socket-Verbindung Socket mit dem Operator send_tuple geschickt worden ist. Wenn noch kein Tupel verschickt worden ist, wird der Prozess, der receive_tuple aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

- ▷ **Socket** (input_control) ..... socket ⇝ *handle*
  Nummer des Sockets.
- ▷ **Tuple** (output_control) ..... string ⇝ *string* / real / integer
  Empfangenes Tupel.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect, get_socket_param, set_socket_param

Siehe auch

send_tuple, send_image, receive_image, send_region, receive_region, get_next_socket_data_type

Modul

Foundation

receive_xld (: XLD : Socket :)

Empfangen von XLD-Objekten über eine Socket-Verbindung.

receive_xld liest ein XLD-Objekt, das von einem anderen HALCON-Prozess über die Socket-Verbindung Socket mit dem Operator send_xld geschickt worden ist. Wenn noch kein XLD-Objekt verschickt worden sind, wird der Prozess, der receive_xld aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Für ein ausführliches Beispiel siehe send_xld.

Parameter

- ▷ **XLD** (output_object) ..... xld(-array) ⇝ *object*
  Empfangenes XLD-Objekt.
- ▷ **Socket** (input_control) ..... socket ⇝ *handle*
  Nummer des Sockets.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`open_socket_connect`, `socket_accept_connect`, `get_socket_param`, `set_socket_param`

Siehe auch

`send_xld`, `send_image`, `receive_image`, `send_region`, `receive_region`, `send_tuple`, `receive_tuple`, `get_next_socket_data_type`

Modul

Foundation

send_data (: : Socket, Format, Data, To :)

Senden von Daten über eine generische Socket-Verbindung auch an externe Geräte oder Anwendungen.

`send_data` verschickt Daten über die durch `Socket` bestimmte generische Socket-Verbindung. Die Daten werden anhand des Parameters `Format` aus einem Wert oder einem Tupel von Werten in ein binäres Netzwerkpaket umgewandelt und sind gut zur Kommunikation mit externen Geräten oder Anwendungen geeignet. Dieser Operator unterstützt das Standard *'HALCON'* Protokoll nicht, sondern ist zum Transfer beliebiger Daten gedacht. Mit dem Operator `receive_data` können entsprechende Daten mit einem HALCON Prozess empfangen werden.

Der Parameter `Format` gibt an, wie Tupel in ein binäres Paket umgewandelt werden. Das Format besteht aus einem oder mehreren Bezeichnern, wobei jedem ein optionaler Modifikator und eine Wiederholungszahl folgen kann. Die meisten Bezeichner erfordern einen einzelnen Wert im Parameter `Data` der entsprechend konvertiert wird.

Die Zeichenkette in `Format` kann die folgenden Zeichen enthalten:

Ganzzahlwerte:

'c': ein Byte = 8 bit,

'C': wie 'c', aber vorzeichenlos

's': zwei Bytes = 16 bit, vorzeichenbehaftet

'S': wie 's', aber vorzeichenlos

'i': vier Bytes = 32 bit, vorzeichenbehaftet

'I': wie 'i', aber vorzeichenlos

'q': acht Bytes = 64 bit, vorzeichenbehaftet (nur für 64bit HALCON verfügbar)

'Q': wie 'q', aber vorzeichenlos

Fließkommawerte:

'f': float, vier Bytes = 32 bit

'd': double, acht Bytes = 64 bit

String values:

'A': Zeichenkette (Standardlänge 1024 Bytes), aufgefüllt mit Leerzeichen

'Z': Zeichenkette (Standardlänge 1024 Bytes), aufgefüllt mit NULL-Bytes, wird beim Senden auf jeden Fall mit einem NULL-Byte abgeschlossen

'z': Zeichenkette variabler Länge, angegeben wird die maximale Länge (Standard 1024 Bytes)

Spezielle Zeichen, die keinen zugehörigen Wert im `Data` Parameter erfordern:

'-': beim Schreiben wird ein einzelnes Byte als binäre NULL übertragen, beim Lesen ein solches NULL-Byte übersprungen

'_': beim Schreiben wird ein einzelnes Byte als Leerzeichen (binär 0x20) übertragen, beim Lesen wird ein Leerzeichen übersprungen

' ': dieses Zeichen wird in der Format Zeichenkette ignoriert und kann somit zur Erhöhung der Lesbarkeit der Format Zeichenkette benutzt werden

Mögliche Modifikationszeichen, die nach einem der oben stehenden Zeichen benutzt werden können:

'n': konvertiere den Ganzzahl- oder Fließkommawert beim Schreiben in die und beim Lesen von der Netzwerk Byte Reihenfolge (big endian) in die lokal benutzte Byte Reihenfolge

'N': konvertiere den Ganzzahl- oder Fließkommawert beim Schreiben in die und beim Lesen von der Intel Byte Reihenfolge (little endian) in die lokal benutzte Byte Reihenfolge

'0-n': Angabe der Wiederholung des vorher stehenden Formatzeichens. Dabei steht z.B. 'c5' für 'ccccc' (benötigt also ein Tupel aus 5 Werten), aber 'A10' bezeichnet eine Zeichenkette mit 10 Zeichen und benötigt nur einen Wert.

Die Modifikationszeichen 'n' und 'N' können auch als erstes Zeichen im Format Zeichenkette angegeben werden und setzen den Standardwert für die Byte Reihenfolge. Wird nichts anderes angeben, so wird 'n', also die Netzwerk Byte Reihenfolge (big endian) benutzt.

Bei UDP Verbindungen müssen die binären Daten in einem Netzwerkpaket übertragen werden. Daher dürfen diese nicht mehr Platz benötigen, als in einem Netzwerkpaket verfügbar ist. Üblicherweise bedeutet dies, dass die Größe kleiner als die MTU (Maximum Transfer Unit) der Netzwerkschnittstelle sein muss, welche normalerweise etwa 1500 Byte beträgt. Garantiert sind aber nur 576 Byte (1280 Byte für IPv6).

Der Parameter To sollte für Socketverbindungen, welche bereits gebunden sind (alle TCP Verbindungen und gebundene UDP Verbindungen), leer gelassen werden. Er ist aber für eine ungebundene UDP Verbindung unbedingt nötig, um die IP Adresse oder den Netzwerknamen und den Netzwerkport der Gegenstelle anzugeben.

Parameter

▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

▷ **Format** (input_control) . . . . . string ⇝ *string*
Vorschrift zur Konvertierung der Daten.
Defaultwert: 'z'

▷ **Data** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
Wert (oder Tupel von Werten) mit den zu sendenden Daten.

▷ **To** (input_control) . . . . . string(-array) ⇝ *string* / integer
IP-Adresse oder Netzwerkname und Netzwerkport der Gegenstelle.
Defaultwert: []
Werteliste: To ∈ {[], ['localhost',3000]}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect, get_socket_param, set_socket_param

Nachfolger

close_socket

Siehe auch

receive_data

Modul

Foundation

send_image (Image : : Socket :)

Senden von Bildern über eine Socket-Verbindung.

send_image verschickt ein Bildobjekt über die durch Socket bestimmte Socket-Verbindung. Der empfangende HALCON-Prozess muss receive_image aufrufen, um das Bild vom Socket auszulesen. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *object :* byte / direction / cyclic / int1 / int2 / uint2 / int4 / int8 / real / complex / vector_field
Zu schickendes Bild.

▷ **Socket** (input_control) ..... socket ⇝ *handle*
Nummer des Sockets.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect

Siehe auch

receive_image, send_region, receive_region, send_tuple, receive_tuple, get_next_socket_data_type

Modul

Foundation

send_region (Region : : Socket :)

Senden von Regionen über eine Socket-Verbindung.

send_region verschickt ein Regionenobjekt über die durch Socket bestimmte Socket-Verbindung. Der empfangende HALCON-Prozess muss receive_region aufrufen, um die Regionen vom Socket auszulesen. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *object*
Zu schickende Regionen.

▷ **Socket** (input_control) ..... socket ⇝ *handle*
Nummer des Sockets.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect

Siehe auch

receive_region, send_image, receive_image, send_tuple, receive_tuple, get_next_socket_data_type

Modul

Foundation

send_serialized_item (: : Socket, SerializedItemHandle :)

Senden eines serialisierten Elements über eine Socket-Verbindung.

send_serialized_item verschickt ein serialisiertes Element über die durch Socket bestimmte Socket-Verbindung (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Der empfangende HALCON-Prozess muss receive_serialized_item aufrufen, um das serialisierte Element vom Socket auszulesen. Das serialisierte Element wird in dem Handle SerializedItemHandle zurückgeben. Für ein ausführliches Beispiel über die Nutzung einer Socket-Verbindung siehe open_socket_accept.

Parameter

▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert send_serialized_item den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_serialized_item_ptr, serialize_matrix, serialize_metrology_model

Nachfolger

receive_serialized_item

Modul

Foundation

send_tuple (: : Socket, Tuple :)

Senden eines Tupels über eine Socket-Verbindung.

send_tuple verschickt ein Tupel über die durch Socket bestimmte Socket-Verbindung. Der empfangende HALCON-Prozess muss receive_tuple aufrufen, um das Tupel vom Socket auszulesen. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

▷ **Tuple** (input_control) . . . . . string ⇝ *string* / real / integer
Zu sendendes Tupel.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect

Siehe auch

receive_tuple, send_image, receive_image, send_region, receive_region, get_next_socket_data_type

Modul

Foundation

send_xld (XLD : : Socket :)

Senden von XLD-Objekten über eine Socket-Verbindung.

send_xld verschickt ein XLD-Objekt über die durch Socket bestimmte Socket-Verbindung. Der empfangende HALCON-Prozess muss receive_xld aufrufen, um das XLD-Objekt vom Socket auszulesen.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu schickendes XLD-Objekt.

▷ **Socket** (input_control) ..... socket ⇝ *handle*
Nummer des Sockets.

Beispiel

```
* Process 1
dev_set_colored (12)
open_socket_accept (3000, [], [], AcceptingSocket)
socket_accept_connect (AcceptingSocket, 'true', Socket)
receive_image (Image, Socket)
edges_sub_pix (Image, Edges, 'canny', 1.5, 20, 40)
send_xld (Edges, Socket)
receive_xld (Polygons, Socket)
split_contours_xld (Polygons, Contours, 'polygon', 1, 5)
gen_parallels_xld (Polygons, Parallels, 10, 30, 0.15, 'true')
send_xld (Parallels, Socket)
receive_xld (ModParallels, Socket)
receive_xld (ExtParallels, Socket)
stop ()
close_socket (Socket)
close_socket (AcceptingSocket)

* Process 2
dev_set_colored (12)
open_socket_connect ('localhost', 3000, [], [], Socket)
read_image (Image, 'mreut')
send_image (Image, Socket)
receive_xld (Edges, Socket)
gen_polygons_xld (Edges, Polygons, 'ramer', 2)
send_xld (Polygons, Socket)
split_contours_xld (Polygons, Contours, 'polygon', 1, 5)
receive_xld (Parallels, Socket)
mod_parallels_xld (Parallels, Image, ModParallels, ExtParallels, \
                   0.4, 160, 220, 10)
send_xld (ModParallels, Socket)
send_xld (ExtParallels, Socket)
stop ()
close_socket (Socket)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect

Siehe auch

receive_xld, send_image, receive_image, send_region, receive_region, send_tuple, receive_tuple, get_next_socket_data_type

Modul

Foundation

set_socket_param (: : Socket, GenParamName, GenParamValue :)

Setzen einer Socket-Einstellung.

set_socket_param stellt für die angegebene Socket-Verbindung die mit GenParamName bezeichnete Einstellung auf den entsprechenden Wert GenParamValue. Verfügbare Parameter sind *'timeout'*, *'SO_SNDBUF'*, *'SO_RCVBUF'*, *'SO_BROADCAST'* und *'TCP_NODELAY'*.

Mit dem Parameter *'timeout'* wird ein Timeout für diesen Socket gesetzt. Er wird als Fließkommazahl in der Einheit Sekunden oder als Zeichenkette *'infinite'* (endlos) angegeben.

Die mit 'SO_' oder 'TCP_' beginnenden Parameter setzen die entsprechenden Socketoptionen. *'SO_SNDBUF'* beziehungsweise *'SO_RCVBUF'* geben die Größe des Sende- und Empfangsbuffers des Betriebssystems für diesen Socket an. Dies ist nicht die Größe eines Netzwerkpakets, sondern die Größe des temporären Speichers für Netzwerkpakete, den das Betriebssystem benutzt. *'SO_BROADCAST'* kann nur mit UDP Verbindungen benutzt werden und ermöglicht die gleichzeitige Übertragung von Netzwerkpaketen an mehrere Empfänger. *'TCP_NODELAY'* kontrolliert den Nagle Algorithmus, mit dem der Fluss kleiner TCP Pakete optimiert wird, und kann nur mit TCP Verbindungen benutzt werden.

Parameter

- ▷ **Socket** (input_control) . . . . . socket ⇝ *handle*
  Nummer des Sockets.
- ▷ **GenParamName** (input_control) . . . . . string(-array) ⇝ *string*
  Name der Socket-Einstellung.
  Werteliste: GenParamName ∈ {'timeout', 'SO_SNDBUF', 'SO_RCVBUF', 'SO_BROADCAST', 'TCP_NODELAY'}
- ▷ **GenParamValue** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
  Wert der Socket-Einstellung.
  Defaultwert: 'on'
  Wertevorschläge: GenParamValue ∈ {'on', 'off', 0, 1, 3.0, 'infinite', 530, 1460}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_connect, socket_accept_connect

Nachfolger

send_data, receive_data

Siehe auch

get_socket_param

Modul

Foundation

socket_accept_connect (: : AcceptingSocket, Wait : Socket)

Annehmen einer Verbindungsanfrage auf einem akzeptierenden Socket vom Protokolltyp 'HALCON' oder 'TCP'/'TCP4'/'TCP6'.

socket_accept_connect nimmt eine ankommende Verbindungsanfrage, die mit open_socket_connect von einem anderen HALCON-Prozess oder einer externen Applikation erzeugt worden ist, auf dem akzeptierenden Socket AcceptingSocket an. Der akzeptierende Socket muss zuvor mit open_socket_accept erzeugt worden sein. Sein Timeout bestimmt die Wartezeit für den socket_accept_connect Aufruf. Falls Wait='*true*', wartet socket_accept_connect, bis eine Verbindung zustande kommt. Falls Wait='*false*', kehrt socket_accept_connect mit der Fehlermeldung FAIL zurück, falls derzeit keine Verbindungsanfrage eines anderen Prozesses vorliegt. Mit dem Wert *'auto'* für Wait wartet socket_accept_connect automatisch wenn der akzeptierende Socket einen Timeout ungleich 0 hat. Das Ergebnis von socket_accept_connect ist ein weiterer Socket Socket, der für eine bidirektionale Kommunikation mit einem anderen Prozess verwendet werden kann. Nachdem die Verbindung hergestellt ist, können Daten zwischen den zwei Prozessen durch Aufruf der entsprechenden Sende- und Empfangsoperatoren ausgetauscht werden. Für ein ausführliches Beispiel siehe open_socket_accept.

Der Datenaustausch über generische Sockets erfolgt ausschließlich mit send_data und receive_data.

Parameter

▷ **AcceptingSocket** (input_control) . . . . . socket ⇝ *handle*
Nummer des akzeptierenden Sockets.

▷ **Wait** (input_control) . . . . . string ⇝ *string*
Soll auf eine Verbindungsanfrage gewartet werden?
Defaultwert: 'auto'
Werteliste: Wait ∈ {'auto', 'true', 'false'}

▷ **Socket** (output_control) . . . . . socket ⇝ *handle*
Nummer des Sockets.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

open_socket_accept

Nachfolger

send_image, receive_image, send_region, receive_region, send_tuple, receive_tuple, send_data, receive_data

Siehe auch

open_socket_connect, close_socket, get_socket_param, set_socket_param

Modul

Foundation

Kapitel 26

Tools

26.1 Funktion

abs_funct_1d (: : Function : FunctionAbsolute)

Absolutbetrag der y-Werte.

abs_funct_1d berechnet die Absolutbeträge der y-Werte der Funktion Function.

Parameter

▷ **Function** (input_control) . . . . . function_1d ⇝ *real* / integer
Eingabefunktion.

▷ **FunctionAbsolute** (output_control) . . . . . function_1d ⇝ *real* / integer
Funktion mit den Absolutbeträgen der y-Werte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Modul

Foundation

compose_funct_1d (: : Function1, Function2,
Border : ComposedFunction)

Zusammensetzung zweier Funktionen.

compose_funct_1d berechnet die Zusammensetzung zweier Funktionen, d.h.

$$\texttt{ComposedFunction}(x) = \texttt{Function2}(\texttt{Function1}(x)) \ .$$

ComposedFunction hat denselben Definitionsbereich (x-Bereich) wie Function1. Falls der Wertebereich von Function1 größer ist als der Definitionsbereich von Function2, gibt Border die Randbehandlung für Function2 an. Für Border='*zero*' wird Function2 mit 0 fortgesetzt, für Border='*constant*' mit dem jeweiligen Randwert, für Border='*mirror*' werden die Funktionswerte am Rand gespiegelt und für Border='*cyclic*' zyklisch fortgesetzt. Für die Berechnung der Funktionswerte wird Function2 linear interpoliert.

Parameter

▷ **Function1** (input_control) ..... function_1d ⇝ *real* / integer
Eingabefunktion 1.

▷ **Function2** (input_control) ..... function_1d ⇝ *real* / integer
Eingabefunktion 2.

▷ **Border** (input_control) ..... string ⇝ *string*
Randbehandlung für die Eingabefunktionen.
Defaultwert: 'constant'
Werteliste: `Border` ∈ {'zero', 'constant', 'mirror', 'cyclic'}

▷ **ComposedFunction** (output_control) ..... function_1d ⇝ *real* / integer
Zusammengesetzte Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_funct_1d_pairs`, `create_funct_1d_array`

Modul

Foundation

create_funct_1d_array (: : YValues : Function)

Erzeugen einer Funktion aus einer Folge von y-Werten.

`create_funct_1d_array` erzeugt aus einer Folge von y-Werten eine eindimensionale Funktion. Diese Funktion kann dann mit den Operatoren für Funktionen verarbeitet und analysiert werden. Die eingegebenen Werte werden in der folgenden Weise interpretiert: Der erste Wert von `YValues` ist der y-Wert an der Stelle Null, der zweite Wert an der Stelle Eins usw. Die Werte erzeugen also eine Funktion aufgebaut aus äquidistanten Stützstellen (mit Abstand $\Delta x = 1$), die bei dem x-Wert $x_1 = 0$ beginnt.

Die erzeugte Funktion ist folgendermaßen zusammengesetzt:

`Function` $= [0, \Delta x, x_1, y_1, y_2, ..., y_n]$

wobei

`Function`$[0] = 0$ die Funktion als Funktion mit äquidistanten Stützstellen kennzeichnet,

Δx die Äquidistanz der x-Werte ist,

x_1 der Anfangswert für x ist (stets als 0 angenommen), und

y_i die y-Werte, die in `YValues` übergeben werden sind (mit $i \in [1, n]$).

Alternativ kann eine Funktion auch mit dem Operator `create_funct_1d_pairs` erzeugt werden, der durch die explizite Angabe der x-Werte prinzipiell auch nicht-äquidistante Stützstellen ermöglicht. Um dieselbe Spezifikation zu erhalten, wie sie bei `create_funct_1d_array` definiert wird, müssten die x-Werte bei `create_funct_1d_pairs` daher als ein Tupel übergeben werden, das beginnend bei Null um 1 aufsteigende Werte enthält. Es ist aber zu beachten, dass die Verwendung von `create_funct_1d_pairs` zu einer anderen internen Darstellung der Funktion führt, die mehr Speicher bedarf (da alle Paare gespeichert werden) und die in der Verarbeitung etwas langsamer sein kann.

Um aus einer mittels `create_funct_1d_array` erzeugten Funktion mit äquidistanten Stützstellen des Abstandes 1 eine neue zu generieren, die einen anderen Abstand bei den äquidistanten Stützstellen aufweist, kann der Operator `transform_funct_1d` verwendet werden.

Parameter

▷ **YValues** (input_control) . . . . . number(-array) ⇝ *real* / integer
X-Werte der Stützstellen.

▷ **Function** (output_control) . . . . . function_1d ⇝ *real* / integer
Erzeugte Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`write_funct_1d`, `y_range_funct_1d`, `get_pair_funct_1d`, `transform_funct_1d`

Alternativen

`create_funct_1d_pairs`, `read_funct_1d`

Siehe auch

`funct_1d_to_pairs`

Modul

Foundation

create_funct_1d_pairs (: : XValues, YValues : Function)

Erzeugen einer Funktion aus Paaren von (x,y)-Werten.

`create_funct_1d_pairs` erzeugt aus Paaren von (x,y)-Werten eine eindimensionale Funktion. Die x-Werte `XValues` der Funktion müssen aufsteigend sortiert sein. Die resultierende Funktion kann dann mit den Operatoren für Funktionen verarbeitet und analysiert werden.

Die erzeugte Funktion ist folgendermaßen zusammengesetzt:

`Function` $= [1, x_1, y_1, x_2, y_2, ..., x_n, y_n]$

wobei

`Function`$[0] = 1$ die Funktion als eindimensionale Funktion aus Paaren von (x,y)-Werten kennzeichnet und

$[x_i, y_i]$ mit $i \in [1, n]$ die in `XValues` und `YValues` übergebenen Wertepaare sind.

Alternativ kann eine Funktion mit dem Operator `create_funct_1d_array` erzeugt werden. Im Gegensatz zu diesem Operator können hier die Stützpunkte jedoch beliebige Abstände haben. `create_funct_1d_pairs` ist also der allgemeinere Operator. Dabei ist jedoch zu beachten, dass die Verarbeitung einer mit `create_funct_1d_pairs` erzeugten Funktion aufgrund dieser Allgemeinheit oftmals nicht so effizient ausgeführt werden kann. Insbesondere sind nicht alle Verarbeitungen implementiert. Gegebenenfalls kann die Funktion mit dem Operator `sample_funct_1d` in eine äquidistante Darstellung umgewandelt werden.

Achtung

`create_funct_1d_pairs` prüft, ob die x-Werte vom Datentyp 'double' aufsteigend sind. Manche Operatoren prüfen dies ebenfalls, benutzen dafür aber den Datentyp 'float'. Wird solch ein Operator nach `create_funct_1d_pairs` aufgerufen, kann es passieren, dass zwei benachbarte Werte in der 'float' Variante gleich erscheinen, die in der 'double' Variante aufsteigend waren. Dies führt zu einer Fehlermeldung.

Parameter

▷ **XValues** (input_control) . . . . . number(-array) ⇝ *real* / integer
X-Werte der Stützstellen.

▷ **YValues** (input_control) . . . . . number(-array) ⇝ *real* / integer
Y-Werte der Stützstellen.

▷ **Function** (output_control) . . . . . function_1d ⇝ *real* / integer
Erzeugte Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`write_funct_1d`, `y_range_funct_1d`, `get_pair_funct_1d`

Alternativen

`create_funct_1d_array`, `read_funct_1d`

Siehe auch

`funct_1d_to_pairs`

Modul

Foundation

derivate_funct_1d (: : Function, Mode : Derivative)

Berechnung der Ableitungen einer Funktion.

`derivate_funct_1d` bestimmt die Ableitungen der Funktion `Function` bis zum Grad 2. Dabei wird ein Differenzenverfahren mit Ordnung $O(h^2)$ verwendet. Die Ableitung `Derivative` ist ebenfalls eine Funktion mit denselben Stützstellen wie `Function`. Mit dem Parameter `Mode` kann zwischen der ersten (*'first'*) und der zweiten (*'second'*) Ableitung gewählt werden.

Parameter

▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
Eingabefunktion

▷ **Mode** (input_control) ..... string ⇝ *string*
Grad der Ableitung
Defaultwert: 'first'
Werteliste: `Mode` ∈ {'first', 'second'}

▷ **Derivative** (output_control) ..... function_1d ⇝ *real* / integer
Ableitung der Funktion

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_funct_1d_pairs`, `create_funct_1d_array`, `smooth_funct_1d_gauss`, `smooth_funct_1d_mean`

Modul

Foundation

distance_funct_1d (: : Function1, Function2, Mode, Sigma : Distance)

Abstand zweier Funktionen.

`distance_funct_1d` berechnet den Abstand zweier Funktionen. Die beiden Funktionen können verschieden lang sein.

Parameter

- ▷ **Function1** (input_control) . . . . . function_1d ⇝ *real* / integer
  Eingabefunktion 1.
- ▷ **Function2** (input_control) . . . . . function_1d ⇝ *real* / integer
  Eingabefunktion 2.
- ▷ **Mode** (input_control) . . . . . string(-array) ⇝ *string*
  Art der Invarianten.
  Defaultwert: 'length'
  Werteliste: `Mode` ∈ {'length', 'mean'}
- ▷ **Sigma** (input_control) . . . . . number(-array) ⇝ *real*
  Standardabweichung der optionalen Gaußglättung
  Defaultwert: 0.0
  Wertevorschläge: `Sigma` ∈ {0.0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0, 15.0, 20.0, 25.0, 30.0, 40.0, 50.0}
- ▷ **Distance** (output_control) . . . . . real-array ⇝ *real* / integer
  Abstand der Funktionen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

funct_1d_to_pairs (: : Function : XValues, YValues)

Zugriff auf die x/y-Werte einer Funktion.

`funct_1d_to_pairs` zerlegt die Eingabefunktion `Function` in Tupel von x- und y- Werten.

Parameter

- ▷ **Function** (input_control) . . . . . function_1d ⇝ *real* / integer
  Eingabefunktion.
- ▷ **XValues** (output_control) . . . . . number-array ⇝ *real* / integer
  X-Werte der Funktion.
- ▷ **YValues** (output_control) . . . . . number-array ⇝ *real* / integer
  Y-Werte der Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_funct_1d_pairs`, `create_funct_1d_array`

Modul

Foundation

get_pair_funct_1d (: : Function, Index : X, Y)

Zugriff auf Funktionswerte über den Index der Stützstelle.

get_pair_funct_1d greift auf einen Funktionswert der Funktion Function zu. Hierzu wird der Index einer oder mehrerer Stützstellen angegeben.

Parameter

▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
Eingabefunktion.

▷ **Index** (input_control) ..... integer(-array) ⇝ *integer*
Index der Stützstelle(n).

▷ **X** (output_control) ..... number(-array) ⇝ *real*
X-Wert an der vorgegebenen Stützstelle.

▷ **Y** (output_control) ..... number(-array) ⇝ *real*
Y-Wert an der vorgegebenen Stützstelle.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Modul

Foundation

get_y_value_funct_1d (: : Function, X, Border : Y)

Berechnung des Funktionswertes an einer beliebigen Position.

get_y_value_funct_1d liefert einen Funktionswert der Funktion Function an den durch X gegebenen Koordinaten zurück. Für die Berechnung der Funktionswerte wird die Eingabefunktion linear interpoliert. Der Parameter Border entscheidet, welchen Wert die Funktion Function außerhalb des gültigen Bereichs hat. Für Border=*'zero'* wird der Wert auf 0 gesetzt, für Border=*'constant'* auf den jeweiligen Randwert, für Border=*'mirror'* werden die Funktionswerte am Rand gespiegelt, für Border=*'cyclic'* werden sie zyklisch fortgesetzt und für Border=*'error'* wird eine Fehlerbehandlung durchgeführt.

Parameter

▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
Eingabefunktion.

▷ **X** (input_control) ..... number(-array) ⇝ *real*
X-Koordinate, an der der Funktionswert berechnet werden soll.

▷ **Border** (input_control) ..... string ⇝ *string*
Randbehandlung für die Eingabefunktion.
Defaultwert: 'constant'
Werteliste: Border ∈ {'zero', 'constant', 'mirror', 'cyclic', 'error'}

▷ **Y** (output_control) ..... number(-array) ⇝ *real*
Y-Wert des vorgegebenen x-Wertes.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Modul

Foundation

integrate_funct_1d (: : Function : Positive, Negative)

Berechnung der positiven und negativen Flächen unter einer Funktion.

integrate_funct_1d integriert die FunktionFunction (siehe create_funct_1d_array und create_funct_1d_pairs) und gibt das Integral der positiven und negativen Teile der Funktion in Positive bzw. Negative zurück. Das Integral der Funktion ist also die Differenz Positive - Negative. Die Integration wird über das Intervall, auf dem die Funktion definiert ist, ausgeführt. Zur Integration wird die Funktion linear interpoliert.

Parameter

- ▷ **Function** (input_control) . . . . . function_1d ⇝ *real*
  Eingabefunktion.
- ▷ **Positive** (output_control) . . . . . number ⇝ *real*
  Fläche unter dem positiven Teil der Funktion.
- ▷ **Negative** (output_control) . . . . . number-array ⇝ *real*
  Fläche unter dem negativen Teil der Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Siehe auch

create_funct_1d_array, create_funct_1d_pairs

Modul

Foundation

invert_funct_1d (: : Function : InverseFunction)

Berechnung der Inversen einer Funktion.

invert_funct_1d berechnet die Inverse der Eingabefunktion Function und liefert sie in InverseFunction. Die Funktion Function muss monoton sein. Falls das nicht der Fall ist, wird eine Fehlermeldung zurückgeliefert.

Parameter

- ▷ **Function** (input_control) . . . . . function_1d ⇝ *real* / integer
  Eingabefunktion.
- ▷ **InverseFunction** (output_control) . . . . . function_1d ⇝ *real* / integer
  Inverse der Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Modul

Foundation

local_min_max_funct_1d (: : Function, Mode, Interpolation : Min, Max)

Berechnung der lokalen Minima und Maxima einer Funktion.

local_min_max_funct_1d bestimmt die lokalen Minima Min und Maxima Max der Funktion Function.

Da die Werte der Funktion nur an diskreten Punkten bekannt sind wird zwischen diesen Punkten quadratisch interpoliert, falls der Parameter Interpolation den Wert *'true'* hat. Wird Interpolation auf *'false'* gesetzt, so werden Extrema nur an den Stützstellen angenommen.

Wird Mode auf *'strict_min_max'* gesetzt, so werden Extrema nur in der Umgebung von solchen Punkten gesucht, deren Funktionswert echt größer bzw. echt kleiner ist als die Werte der benachbarten Punkte.

Ist Mode auf *'plateaus_center'* eingestellt, so werden auch Bereiche betrachtet, in denen der Funktionswert über mehrere Stützstellen hinweg konstant bleibt. Wird solch ein Bereich als abgeflachtes Extremum identifiziert, so wird sein Mittelpunkt zurückgeliefert.

Parameter

- ▷ **Function** (input_control) . . . . . function_1d ⇝ *real / integer*
  Eingabefunktion
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Plateaubehandlung
  Defaultwert: 'strict_min_max'
  Werteliste: Mode ∈ {'strict_min_max', 'plateaus_center'}
- ▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
  Interpolation der Funktion
  Defaultwert: 'true'
  Werteliste: Interpolation ∈ {'true', 'false'}
- ▷ **Min** (output_control) . . . . . real-array ⇝ *real*
  Minima der Funktion
- ▷ **Max** (output_control) . . . . . real-array ⇝ *real*
  Maxima der Funktion

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array, smooth_funct_1d_gauss, smooth_funct_1d_mean

Modul

Foundation

match_funct_1d_trans (: : Function1, Function2, Border, ParamsConst, UseParams : Params, ChiSquare, Covar)

Berechnung der Transformationsparameter zwischen zwei Funktionen.

match_funct_1d_trans berechnet die Transformationsparameter zwischen zwei Funktionen, die als die Tupel Function1 und Function2 übergeben werden (siehe create_funct_1d_array und create_funct_1d_pairs). Es wird folgendes Modell der Transformation zwischen den zwei Funktionen verwendet:

$$y_1(x) = a_1 y_2(a_3 x + a_4) + a_2 \ .$$

Die Transformationsparameter werden durch eine Ausgleichsrechnung berechnet, indem die folgende Funktion minimiert wird:

$$\sum_{i=0}^{n-1} \left(y_1(x_i) - a_1 y_2(a_3 x_i + a_4) + a_2\right)^2 \ .$$

Dabei werden die Funktionswerte von y_2 linear interpoliert. Der Parameter `Border` entscheidet, welchen Wert die Funktion `Function2` außerhalb des gültigen Bereichs hat. Für `Border`=*'zero'* wird der Wert auf 0 gesetzt, für `Border`=*'constant'* auf den jeweiligen Randwert, für `Border`=*'mirror'* werden die Funktionswerte am Rand gespiegelt und für `Border`=*'cyclic'* werden sie zyklisch fortgesetzt. Die berechneten Transformationsparameter werden als Tupel $[a_1, a_2, a_3, a_4]$ in `Params` zurückgeliefert. Falls einige der Parameter bekannte Werte besitzen, so kann der jeweilige Parameter von der Ausgleichsrechnung ausgeschlossen werden, indem an der entsprechenden Stelle im Tupel `UseParams` der Wert *'false'* eingetragen wird. In diesem Fall muss im Tupel `ParamsConst` der bekannte Wert des Parameters eingetragen werden. Wenn der Parameter für die Ausgleichung verwendet wird (`UseParams` = *'true'*), wird der entsprechende Parameter in `ParamsConst` ignoriert. Als Ausgabe liefert `match_funct_1d_trans` weiterhin die Summe der quadratischen Fehler `ChiSquare` der Ergebnisfunktion, also der mit den Transformationsparametern transformierten Funktion, sowie die Kovariazmatrix `Covar` der Transformationsparameter `Params` zurück. Diese Werte können verwendet werden, um festzustellen, ob ein erfolgreiches Matching der Funktionen möglich war.

Für den Fall, dass es keine eindeutige Lösung für die Transformationsparameter gibt, liefert `match_funct_1d_trans` entweder eine einzelne ausgewählte Lösung oder den Fehler 9205 (Matrix ist singulär) zurück.

Parameter

- ▷ **`Function1`** (input_control) ..... function_1d ⇝ *real* / integer
  Funktion 1.
- ▷ **`Function2`** (input_control) ..... function_1d ⇝ *real* / integer
  Funktion 2.
- ▷ **`Border`** (input_control) ..... string ⇝ *string*
  Randbehandlung für Funktion 2.
  Defaultwert: 'constant'
  Werteliste: `Border` ∈ {'zero', 'constant', 'mirror', 'cyclic'}
- ▷ **`ParamsConst`** (input_control) ..... number-array ⇝ *real*
  Werte der Parameter, die konstant gehalten werden.
  Parameteranzahl: 4
  Defaultwert: [1.0,0.0,1.0,0.0]
- ▷ **`UseParams`** (input_control) ..... string-array ⇝ *string*
  Soll ein Parameter angepasst werden?
  Parameteranzahl: 4
  Defaultwert: ['true','true','true','true']
  Werteliste: `UseParams` ∈ {'true', 'false'}
- ▷ **`Params`** (output_control) ..... number-array ⇝ *real*
  Transformationsparameter zwischen den Funktionen.
  Parameteranzahl: 4
- ▷ **`ChiSquare`** (output_control) ..... number ⇝ *real*
  Quadratischer Fehler der Ausgabefunktion.
- ▷ **`Covar`** (output_control) ..... number-array ⇝ *real*
  Kovarianzmatrix der Transformationsparameter
  Parameteranzahl: 16

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_funct_1d_array`, `create_funct_1d_pairs`

Siehe auch

gray_projections

Modul

Foundation

negate_funct_1d (: : Function : FunctionInverted)

Negation aller y-Werte.

negate_funct_1d negiert alle y-Werte der Funktion Function.

Parameter

- ▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
  Eingabefunktion.
- ▷ **FunctionInverted** (output_control) ..... function_1d ⇝ *real* / integer
  Funktion mit den negierten y-Werten.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Modul

Foundation

num_points_funct_1d (: : Function : Length)

Anzahl der Stützstellen der Funktion.

num_points_funct_1d bestimmt die Anzahl von Stützstellen der Funktion Function.

Parameter

- ▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
  Eingabefunktion.
- ▷ **Length** (output_control) ..... integer ⇝ *integer*
  Anzahl der Stützstellen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Modul

Foundation

read_funct_1d (: : FileName : Function)

Eine Funktion von Datei lesen.

read_funct_1d liest den Inhalt der Datei FileName und wandelt diesen in die Funktion Function, dargestellt als Tupel, um. Die Datei muss mit write_funct_1d erzeugt worden sein.

Parameter

- ▷ **FileName** (input_control) ..... filename.read ⇝ *string*
  Name der zu lesenden Datei.
- ▷ **Function** (output_control) ..... function_1d ⇝ *real* / integer
  Eingelesene Funktion.

Ergebnis

Sind die Parameter korrekt, dann liefert read_funct_1d den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

fread_string, read_tuple

Siehe auch

write_funct_1d, write_image, write_region, open_file

Modul

Foundation

sample_funct_1d (: : Function, XMin, XMax, XDist, Border : SampledFunction)

Taste eine Funktion in einem Intervall äquidistant ab.

sample_funct_1d tastet die Eingabefunktion Function im Intervall [XMin,XMax] an äquidistanten Stützstellen mit dem Abstand XDist ab. Die letzte Stützstelle liegt im Intervall, falls XMax-XMin kein ganzzahliges Vielfaches von XDist ist. Bei der Abtastung wird die Eingabefunktion linear interpoliert. Der Parameter Border entscheidet, welchen Wert die Funktion Function außerhalb des gültigen Bereichs hat. Für Border='zero' wird der Wert auf 0 gesetzt, für Border='constant' auf den jeweiligen Randwert, für Border='mirror' werden die Funktionswerte am Rand gespiegelt und für Border='cyclic' werden sie zyklisch fortgesetzt.

Parameter

- ▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
  Eingabefunktion.
- ▷ **XMin** (input_control) ..... number ⇝ *real* / integer
  Minimaler x-Wert der Ausgabefunktion.
- ▷ **XMax** (input_control) ..... number ⇝ *real* / integer
  Maximaler x-Wert der Ausgabefunktion.
  Restriktion: XMax > XMin
- ▷ **XDist** (input_control) ..... number ⇝ *real* / integer
  Abstand der Abtastwerte.
  Restriktion: XDist > 0
- ▷ **Border** (input_control) ..... string ⇝ *string*
  Randbehandlung für die Eingabefunktion.
  Defaultwert: 'constant'
  Werteliste: Border ∈ {'zero', 'constant', 'mirror', 'cyclic'}
- ▷ **SampledFunction** (output_control) ..... function_1d ⇝ *real* / integer
  Abgetastete Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`transform_funct_1d`, `create_funct_1d_array`, `create_funct_1d_pairs`

Modul

Foundation

scale_y_funct_1d (: : Function, Mult, Add : FunctionScaled)

Multiplizieren und Addieren der y-Werte der Funktion.

`scale_y_funct_1d` multipliziert und addiert die y-Werte der Funktion `Function` mit den Parametern `Mult` und `Add`.

Parameter

▷ **Function** (input_control) . . . . . function_1d ⇝ *real* / integer
Eingabefunktion.

▷ **Mult** (input_control) . . . . . number ⇝ *real*
Faktor für Skalierung der y-Werte.
Defaultwert: 2.0
Wertevorschläge: `Mult` ∈ {0.1, 0.3, 0.5, 1.0, 2.0, 5.0, 10.0}

▷ **Add** (input_control) . . . . . number ⇝ *real*
Konstante, die auf die y-Werte addiert wird.
Defaultwert: 0.0
Wertevorschläge: `Add` ∈ {-10.0, -5.0, 1.0, 0.0, 5.0, 10.0}

▷ **FunctionScaled** (output_control) . . . . . function_1d ⇝ *real* / integer
Transformierte Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_funct_1d_pairs`, `create_funct_1d_array`

Modul

Foundation

smooth_funct_1d_gauss (: : Function, Sigma : SmoothedFunction)

Glättung einer äquidistanten 1D Funktion mit einer Gaußfunktion.

`smooth_funct_1d_gauss` führt die Glättung einer eindimensionalen Funktion durch die Anwendung einer Gaußfunktion durch. Die Funktion muss äquidistant sein, d.h. mit `create_funct_1d_array`, `sample_funct_1d` o.ä. erzeugt worden sein. An den Rändern der Funktion werden die Funktionswerte gespiegelt.

Der Glättungsparameter `Sigma` darf nicht größer sein als (`Length`-2) / 7.8 * `Function`[1], wobei `Length`-2 die Anzahl der Stützstellen der Funktion `Function` und `Function`[1] die Äquidistanz der Stützstellen ist. `Length` kann mit `num_points_funct_1d` bestimmt werden.

Parameter

▷ **Function** (input_control) . . . . . function_1d ⇝ *real* / integer
Zu glättende Funktion.

▷ **Sigma** (input_control) . . . . . number ⇝ *real*
Sigma der Gaußfunktion für die Glättung.
Defaultwert: 2.0
Wertevorschläge: `Sigma` ∈ {0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
Typischer Wertebereich: 0.1 ≤ `Sigma` ≤ 50.0 (lin)
Minimale Schrittweite: 0.01
Empfohlene Schrittweite: 0.2

▷ **SmoothedFunction** (output_control) . . . . . function_1d ⇝ *real* / integer
Geglättete Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_funct_1d_array`, `sample_funct_1d`

Nachfolger

`match_funct_1d_trans`, `distance_funct_1d`

Alternativen

`smooth_funct_1d_mean`

Modul

Foundation

smooth_funct_1d_mean (: : Function, SmoothSize, Iterations : SmoothedFunction)

Glättung einer äquidistanten 1D Funktion durch Mittelung.

`smooth_funct_1d_mean` führt die Glättung einer 1-dimensionalen Funktion mit Hilfe der mehrmaligen Anwendung eines Mittelwertfilters durch. Die Funktion muss äquidistant sein, d.h. mit `create_funct_1d_array`, `sample_funct_1d` o.ä. erzeugt worden sein. An den Rändern der Funktion werden die Funktionswerte gespiegelt.

Achtung

Wird für `SmoothSize` ein gerader statt einem ungerade Wert übergeben, verwendet die Routine an ihrer Stelle den nächstgrößeren ungeraden Wert (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Parameter

▷ **Function** (input_control) . . . . . function_1d ⇝ *integer* / real
1D Funktion.

▷ **SmoothSize** (input_control) . . . . . integer ⇝ *integer*
Größe der Glättungsmaske.
Defaultwert: 9
Wertevorschläge: `SmoothSize` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 21, 31, 51}
Typischer Wertebereich: 1 ≤ `SmoothSize` ≤ 1001 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 2
Restriktion: SmoothSize > 0

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Iterationen der Glättung.
Defaultwert: 3
Wertevorschläge: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9}
Typischer Wertebereich: 1 ≤ Iterations ≤ 100 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1
Restriktion: Iterations >= 1

▷ **SmoothedFunction** (output_control) ..... function_1d ⇝ *real* / integer
Geglättete Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_array, sample_funct_1d

Alternativen

smooth_funct_1d_gauss

Modul

Foundation

transform_funct_1d (: : Function, Params : TransformedFunction)

Transformiere eine Funktion mit gegebenen Transformationsparametern.

transform_funct_1d transformiert die Eingabefunktion Function mit den Transformationsparametern, die in Params übergeben werden. Die Funktion Function wird dabei als Tupel übergeben (siehe create_funct_1d_array und create_funct_1d_pairs). Es wird folgendes Modell der Transformation zwischen den zwei Funktionen verwendet (siehe match_funct_1d_trans):

$$y_t(x) = a_1 y(a_3 x + a_4) + a_2 \ .$$

Die Ausgabefunktion TransformedFunction entsteht dadurch, dass die x- und y-Werte der Eingabefunktion getrennt nach obiger Formel transformiert werden, d.h. die Ausgabefunktion wird nicht neu abgetastet. Daher muss $a_3 \neq 0.0$ gelten. Um eine Funktion neu abzutasten steht der Operator sample_funct_1d zur Verfügung.

Parameter

▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
Eingabefunktion.

▷ **Params** (input_control) ..... number-array ⇝ *real*
Transformationsparameter zwischen den Funktionen.
Parameteranzahl: 4

▷ **TransformedFunction** (output_control) ..... function_1d ⇝ *real* / integer
Transformierte Funktion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array, match_funct_1d_trans

Modul

Foundation

write_funct_1d (: : Function, FileName :)

Eine Funktion auf Datei schreiben.

write_funct_1d schreibt den Inhalt von Function auf Datei. Es handelt sich hierbei um ein ASCII-Format, d.h. die Daten sind (siehe auch Tupel / String-Operationen) zwischen unterschiedlichen Rechnertypen austauschbar. Die Daten können mit dem Operator read_funct_1d wieder eingelesen werden. Für den Dateiname ist keine Extension festgelegt.

Parameter

▷ **Function** (input_control) . . . . . function_1d ⇝ *real / integer*
Funktion die geschrieben werden soll.

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der zu schreibenden Datei.

Ergebnis

Sind die Parameter korrekt, dann liefert write_funct_1d den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Alternativen

write_tuple, fwrite_string

Siehe auch

read_funct_1d, write_image, write_region, open_file

Modul

Foundation

x_range_funct_1d (: : Function : XMin, XMax)

Kleinster und größter x-Wert der Funktion.

x_range_funct_1d bestimmt den kleinsten und den größten x-Wert der Funktion Function.

Parameter

▷ **Function** (input_control) . . . . . function_1d ⇝ *real / integer*
Eingabefunktion.

▷ **XMin** (output_control) . . . . . number ⇝ *real*
Kleinster x-Wert.

▷ **XMax** (output_control) . . . . . number ⇝ *real*
Größter x-Wert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Modul

Foundation

y_range_funct_1d (: : Function : YMin, YMax)

Kleinster und größter y-Wert der Funktion.

y_range_funct_1d bestimmt den kleinsten und den größten y-Wert der Funktion Function.

Parameter

- ▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
  Eingabefunktion.
- ▷ **YMin** (output_control) ..... number ⇝ *real*
  Kleinster y-Wert.
- ▷ **YMax** (output_control) ..... number ⇝ *real*
  Größter y-Wert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array

Modul

Foundation

zero_crossings_funct_1d (: : Function : ZeroCrossings)

Berechnung der Nullstellen einer Funktion.

zero_crossings_funct_1d bestimmt die Nullstellen ZeroCrossings der Funktion Function. Dabei werden die Werte der Funktion zwischen ihren Stützstellen linear interpoliert und so die Koordinate des Nulldurchgangs genau bestimmt. Besitzt die gesamte Gerade zwischen zwei Stützstellen den Wert *0*, so werden nur der Anfangs- und Endpunkt dieses Intervalls zurückgeliefert.

Parameter

- ▷ **Function** (input_control) ..... function_1d ⇝ *real* / integer
  Eingabefunktion
- ▷ **ZeroCrossings** (output_control) ..... real-array ⇝ *real*
  Nullstellen der Funktion

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_funct_1d_pairs, create_funct_1d_array, smooth_funct_1d_gauss, smooth_funct_1d_mean

Modul

Foundation

26.2 Geometrie

angle_ll (: : RowA1, ColumnA1, RowA2, ColumnA2, RowB1, ColumnB1, RowB2, ColumnB2 : Angle)

Berechnen des Winkels zwischen zwei Geraden.

angle_ll berechnet den Winkel zwischen zwei Geraden. Als Eingabe werden die Koordinaten von zwei Punkten der ersten Geraden (RowA1,ColumnA1, RowA2,ColumnA2) und der zweiten Geraden (RowB1,ColumnB1, RowB2,ColumnB2) erwartet. Die Berechnung des Winkels geschieht wie folgt: Man interpretiert die beiden Geraden als Vektoren, wobei RowA1,ColumnA1 bzw. RowB1,ColumnB1 die Startpunkte und RowA2,ColumnA2 bzw. RowB2,ColumnB2 die Endpunkte sind. Dreht man nun den Vektor A gegen den Uhrzeigersinn auf den Vektor B (der Drehpunkt ist der Schnittpunkt der Geraden), so ist der benötigte Rotationswinkel der gesuchte Wert. Der Winkel ist also abhängig von der Reihenfolge der Koordinaten und der Reihenfolge der beiden Geraden. Das Ergebnis, d.h. der Winkel in Bogenmaß, wird im Parameter Angle übergeben. Die Winkel liegen im Bereich von $-\pi \leq \text{Angle} \leq \pi$.

Parameter

- ▷ **RowA1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der ersten Geraden.
- ▷ **ColumnA1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der ersten Geraden.
- ▷ **RowA2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der ersten Geraden.
- ▷ **ColumnA2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der ersten Geraden.
- ▷ **RowB1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der zweiten Geraden.
- ▷ **ColumnB1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der zweiten Geraden.
- ▷ **RowB2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der zweiten Geraden.
- ▷ **ColumnB2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der zweiten Geraden.
- ▷ **Angle** (output_control) ..... angle.rad(-array) ⇝ *real*
  Winkel zwischen den Geraden [rad].

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
RowA1 := 255
ColumnA1 := 10
RowA2 := 255
ColumnA2 := 501
gen_contour_polygon_xld (Contour, [RowA1,RowA2], [ColumnA1,ColumnA2])
RowB1 := 255
ColumnB1 := 255
for I := 5 to 360 by 5
  RowB2 := 255 - sin(rad(I)) * 200
  ColumnB2 := 255 + cos(rad(I)) * 200
  gen_contour_polygon_xld (Contour, [RowB1,RowB2], [ColumnB1,ColumnB2])
  angle_ll (RowA1, ColumnA1, RowA2, ColumnA2,\
            RowB1, ColumnB1, RowB2, ColumnB2, Angle)
  AngleDeg := deg(Angle)
endfor
```

Ergebnis

angle_ll liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

angle_lx

Modul

Foundation

angle_lx (: : Row1, Column1, Row2, Column2 : Angle)

Berechnen des Winkels zwischen einer Geraden und der horizontalen Achse.

angle_lx berechnet den Winkel zwischen einer Geraden und der Horizontalen. Als Eingabe werden die Koordinaten von zwei Punkten auf der Geraden (Row1,Column1, Row2,Column2) erwartet. Die Berechnung des Winkels geschieht wie folgt: Man interpretiert die Gerade als einen Vektor, wobei Row1,Column1 der Startpunkt und Row2,Column2 der Endpunkt sind. Der Startpunkt liegt dabei auf der Horizontalen und dient in der folgenden Überlegung als Drehpunkt. Liegt der Endpunkt des Vektors oberhalb der Horizontalen, ergibt sich der gesuchte Winkel (mit positivem Vorzeichen) aus der Rotation dieser Horizontalen gegen den Uhrzeigersinn auf den Vektor. Liegt der Endpunkt des Vektors unterhalb der Horizontalen, ergibt sich der gesuchte Winkel (mit negativem Vorzeichen) aus der Rotation dieser Horizontalen im Uhrzeigersinn auf den Vektor. Der Winkel ist also abhängig von der Reihenfolge der Koordinaten. Das Ergebnis, d.h. der Winkel in Bogenmaß, wird im Parameter Angle übergeben. Der Winkel liegt im Bereich von $-\pi \leq$ Angle $< \pi$.

Parameter

- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Geraden.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Geraden.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Geraden.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Geraden.
- ▷ **Angle** (output_control) ..... angle.rad(-array) ⇝ *real*
  Winkel zwischen der Geraden und der Horizontalen [rad].

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
RowA1 := 255
ColumnA1 := 10
RowA2 := 255
ColumnA2 := 501
gen_contour_polygon_xld (Contour, [RowA1,RowA2], [ColumnA1,ColumnA2])
Row1 := 255
Column1 := 255
for I := 5 to 360 by 5
  Row2 := 255 - sin(rad(I)) * 200
  Column2 := 255 + cos(rad(I)) * 200
  gen_contour_polygon_xld (Contour, [Row1,Row2], [Column1,Column2])
  angle_lx (Row1, Column1, Row2, Column2, Angle)
  AngleDeg := deg(Angle)
endfor
```

Ergebnis

angle_lx liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

angle_ll

Modul

Foundation

apply_distance_transform_xld (
Contour : ContourOut : DistanceTransformID :)

Berechnen des punktweisen Abstands von Konturen mittels einer XLD-Distanztransformation.

Der Operator apply_distance_transform_xld berechnet für jeden Punkt in Contour den minimalen Abstand zur Referenzkontur mithilfe ihrer XLD-Distanztransformation DistanceTransformID. Die zurückgelieferte Kontur ContourOut enthält die Kontur Contour mit dem Attribut *'distance'*, in dem die berechneten Abstände gespeichert sind. Diese können anschließend durch die Abfrage des Attributs *'distance'* mit dem Operator get_contour_attrib_xld abgerufen werden. Weitere Informationen zu Konturattributen können der Referenz von get_contour_attrib_xld entnommen werden.

Die berechneten Abstände werden von den Parametern des Operators create_distance_transform_xld beeinflusst: Die Abstände werden bei der maximalen Distanz abgeschnitten, die im Parameter MaxDistance festgelegt ist. Der Parameter Mode legt fest, ob die Abstände *'point_to_point'* oder *'point_to_segment'* berechnet werden. Weitere Informationen finden sich auch in der Dokumentation von create_distance_transform_xld.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont(-array) ⇝ *object*
  Kontur(en) für deren Punkte die Abstände berechnet werden.
- ▷ **ContourOut** (output_object) ..... xld_cont(-array) ⇝ *object*
  Kopie von Contour mit den Abständen als Attribut.
- ▷ **DistanceTransformID** (input_control) ..... xld_dist_trans ⇝ *handle*
  Handle der XLD-Distanztransformation der Referenzkontur.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert der Operator den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_distance_transform_xld, read_distance_transform_xld, deserialize_distance_transform_xld, set_distance_transform_xld_param

Nachfolger

get_contour_attrib_xld, segment_contour_attrib_xld

Alternativen

distance_contours_xld

Siehe auch

get_contour_attrib_xld, set_distance_transform_xld_param, write_distance_transform_xld, serialize_distance_transform_xld, clear_distance_transform_xld

Modul

Foundation

area_intersection_rectangle2 (: : Rect1Row, Rect1Column, Rect1Phi, Rect1Length1, Rect1Length2, Rect2Row, Rect2Column, Rect2Phi, Rect2Length1, Rect2Length2 : AreaIntersection)

Berechnen der Schnittfläche von zwei orientierten Rechtecken.

area_intersection_rectangle2 berechnet die Schnittfläche von zwei orientierten Rechtecken (d.h. vom Typ Rectangle2), welche jeweils durch ihre Parameter (Schwerpunkt (Rect1Row, Rect1Column), Orientierung Rect1Phi sowie die halben Kantenlängen Rect1Length1 und Rect1Length2), respektive (Schwerpunkt (Rect2Row, Rect2Column), Orientierung Rect2Phi, sowie die halben Kantenlängen Rect2Length1 und Rect2Length2), definiert sind. Die Schnittfläche wird in AreaIntersection zurückgegeben. Wenn mehrere Werte für die Parameter des zweiten Rechtecks übergeben werden, werden die Schnittflächen des ersten Rechtecks mit allen gegebenen zweiten Rechtecken berechnet und zurückgegeben. Wenn leere Tupel für die Parameter des zweiten Rechtecks übergeben werden, wird ein leeres Tupel in AreaIntersection zurückgegeben.

Parameter

- ▷ **Rect1Row** (input_control) ..... rectangle2.center.y ⇝ *real* / integer
  Zeilenkoordinate des Mittelpunkts des ersten Rechtecks.
- ▷ **Rect1Column** (input_control) ..... rectangle2.center.x ⇝ *real* / integer
  Spaltenkoordinate des Mittelpunkts des ersten Rechtecks.
- ▷ **Rect1Phi** (input_control) ..... rectangle2.angle.rad ⇝ *real* / integer
  Winkel zwischen der positiven horizontalen Achse und der ersten Kante des ersten Rechtecks (im Bogenmaß).
- ▷ **Rect1Length1** (input_control) ..... rectangle2.hwidth ⇝ *real* / integer
  Halbe Länge der ersten Kante des ersten Rechtecks.
- ▷ **Rect1Length2** (input_control) ..... rectangle2.hheight ⇝ *real* / integer
  Halbe Länge der zweiten Achse des ersten Rechtecks.
- ▷ **Rect2Row** (input_control) ..... rectangle2.center.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Mittelpunkts des zweiten Rechtecks.
- ▷ **Rect2Column** (input_control) ..... rectangle2.center.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Mittelpunkts des zweiten Rechtecks.
- ▷ **Rect2Phi** (input_control) ..... rectangle2.angle.rad(-array) ⇝ *real* / integer
  Winkel zwischen der positiven horizontalen Achse und der ersten Kante des zweiten Rechtecks (im Bogenmaß).
- ▷ **Rect2Length1** (input_control) ..... rectangle2.hwidth(-array) ⇝ *real* / integer
  Halbe Länge der ersten Kante des zweiten Rechtecks.
- ▷ **Rect2Length2** (input_control) ..... rectangle2.hheight(-array) ⇝ *real* / integer
  Halbe Länge der zweiten Kante des zweiten Rechtecks.
- ▷ **AreaIntersection** (output_control) ..... real(-array) ⇝ *real*
  Schnittfläche zwischen dem ersten und dem zweiten Rechteck.

Ergebnis

area_intersection_rectangle2 liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

gen_rectangle2, intersection_segments, intersection_segment_line, intersection_segment_contour_xld, intersection_line_contour_xld, intersection_contours_xld

Modul

Foundation

clear_distance_transform_xld (: : DistanceTransformID :)

Löschen einer XLD-Distanztransformation.

clear_distance_transform_xld löscht die XLD-Distanztransformation DistanceTransformID, die zuvor mit create_distance_transform_xld erzeugt wurde.

Parameter

▷ **DistanceTransformID** (input_control) ..... xld_dist_trans ⇝ *handle*
Handle der XLD-Distanztransformation.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert der Operator den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DistanceTransformID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_distance_transform_xld, write_distance_transform_xld, serialize_distance_transform_xld

Siehe auch

apply_distance_transform_xld, get_distance_transform_xld_contour, get_distance_transform_xld_param, set_distance_transform_xld_param, read_distance_transform_xld, deserialize_distance_transform_xld

Modul

Foundation

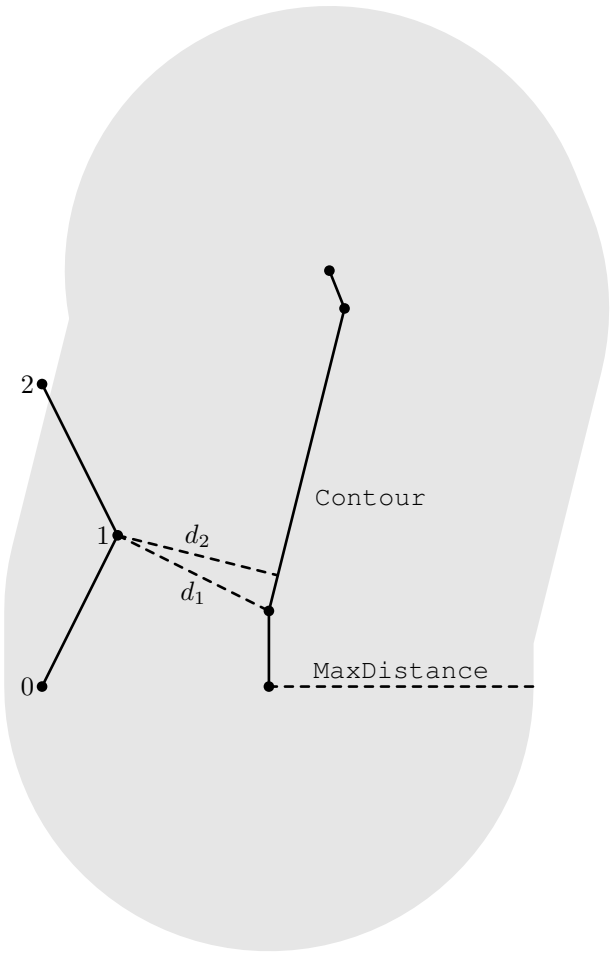
create_distance_transform_xld (Contour : : Mode, MaxDistance : DistanceTransformID)

Erzeugen einer XLD-Distanztransformation.

create_distance_transform_xld erzeugt die XLD-Distanztransformation der Referenzkontur Contour und liefert das entstandene Handle in DistanceTransformID zurück.

Ist die XLD-Distanztransformation erzeugt, so berechnet der Operator apply_distance_transform_xld punktweise Abstände von Testkonturen zur Referenzkontur Contour. Genauer: Für jeden Punkt einer Testkontur wird sein minimaler Abstand zu den in Contour übergebenen Konturen berechnet.

Der Parameter Mode legt fest, welche Abstände in apply_distance_transform_xld berechnet werden: *'point_to_point'* berechnet den minimalen Abstand zu den Stützpunkten von Contour. Dagegen berechnet *'point_to_segment'* den minimalen Abstand zu den Kontursegmenten von Contour (siehe auch nachstehende Abbildung).



Die gestrichelten Linien stellen dar, welcher Abstand ausgehend von Punkt 1 der Testkontur links berechnet wird. Abstand d_1 entspricht dem Modus *'point_to_point'*, und Abstand d_2 dem Modus 'point_to_segment'. Der grau unterlegte Bereich stellt die Fläche innerhalb von `MaxDistance` um die Referenzkontur dar, für die Abstandswerte berechnet werden. Der Abstand von Punkt 2 der Testkontur würde auf `MaxDistance` gesetzt werden, da er außerhalb dieses Bereichs liegt.

Der Parameter `MaxDistance` legt den maximalen Abstand von der Referenzkontur `Contour` fest, der von Interesse ist. Falls der Abstand eines Punktes der Testkontur zur Referenzkontur `MaxDistance` übersteigt, so wird der Ausgabe-Abstand von `apply_distance_transform_xld` auf `MaxDistance` gesetzt.

`create_distance_transform_xld` und `apply_distance_transform_xld` stellen eine Alternative zu `distance_contours_xld` dar, falls die Referenzkontur wiederholt verwendet wird. `create_distance_transform_xld` speichert für jeden Pixel in einer relevanten Fläche um `Contour` seine nächsten Punkte oder Segmente (abhängig von `Mode`) der Referenzkontur `Contour`. Damit kann `apply_distance_transform_xld` Abstände sehr schnell berechnen, fast unabhängig von der Anzahl Punkte oder Segmente der Referenzkontur, von `Mode` und der Lage der Punkte der Testkontur. Allerdings kann das Erstellen der XLD-Distanztransformation einige Sekunden oder Minuten in Anspruch nehmen, abhängig von der Anzahl Punkte oder Segmente der Referenzkontur und der relevanten Fläche um `Contour`, die durch `MaxDistance` beeinflusst wird. `create_distance_transform_xld` ist schneller, falls `Mode` auf 'point_to_point' gesetzt wird.

`get_distance_transform_xld_contour` und `get_distance_transform_xld_param` können verwendet werden, um die Referenzkontur und Parameter der XLD-Distanztransformation `DistanceTransformID` zurückzubekommen.

Parameter

▷ **`Contour`** (input_object) ..... xld_cont(-array) ⇝ *object*
Referenzkontur(en).

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Berechne die Distanz zu Punkten ('point_to_point') oder zu ganzen Segmenten ('point_to_segment').
Defaultwert: 'point_to_point'
Werteliste: Mode ∈ {'point_to_point', 'point_to_segment'}

▷ **MaxDistance** (input_control) . . . . . real ⇝ *real* / integer
Maximaler interessierender Abstand.
Defaultwert: 20.0

▷ **DistanceTransformID** (output_control) . . . . . xld_dist_trans ⇝ *handle*
Handle der XLD-Distanztransformation der Kontur(en).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert der Operator den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

apply_distance_transform_xld, write_distance_transform_xld, serialize_distance_transform_xld, clear_distance_transform_xld

Siehe auch

distance_contours_xld, get_distance_transform_xld_contour, read_distance_transform_xld, deserialize_distance_transform_xld, get_distance_transform_xld_param, set_distance_transform_xld_param

Modul

Foundation

deserialize_distance_transform_xld (: : SerializedItemHandle : DistanceTransformID)

Deserialisieren einer XLD-Distanztransformation.

deserialize_distance_transform_xld deserialisiert eine XLD-Distanztransformation. Die serialisierte XLD-Distanztransformation wird in dem Handle SerializedItemHandle übergeben. Die deserialisierte XLD-Distanztransformation wird in DistanceTransformID zurückgeliefert.

Es ist zu beachten, dass die aktuellen Werte in DistanceTransformID überschrieben werden, falls dieses Handle bereits existiert.

get_distance_transform_xld_contour kann verwendet werden, um auf die Referenzkontur zuzugreifen, die für die Erzeugung der XLD-Distanztransformation DistanceTransformID verwendet wurde.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle der serialisierten XLD-Distanztransformation.

▷ **DistanceTransformID** (output_control) . . . . . xld_dist_trans ⇝ *handle*
Handle der deserialisierten XLD-Distanztransformation.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_distance_transform_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

apply_distance_transform_xld, get_distance_transform_xld_contour, get_distance_transform_xld_param

Siehe auch

serialize_distance_transform_xld, write_distance_transform_xld, read_distance_transform_xld, create_distance_transform_xld, set_distance_transform_xld_param, clear_distance_transform_xld

Modul

Foundation

distance_cc (Contour1, Contour2 : : Mode : DistanceMin, DistanceMax)

Berechnen des Abstandes zwischen zwei Konturen.

distance_cc berechnet den minimalen und maximalen Abstand zwischen den Stützpunkten zweier Konturen. Als Eingabe werden die beiden Konturen (Contour1,Contour2) erwartet. Das jeweilige Ergebnis wird in den Parametern DistanceMin und DistanceMax zurückgegeben.

Der Parameter Mode gibt an, auf welche Weise der Abstand berechnet wird. *'point_to_point'* bestimmt nur den kleinsten und größten Abstand der Stützpunkte der Konturen, was sich zwar schneller berechnen lässt, jedoch zu Ungenauigkeiten beim minimalem Abstand führen kann. *'point_to_segment'* bestimmt dagegen den tatsächlichen minimalen Abstand der Kontursegmente. In beiden Fällen hat der Suchalgorithmus quadratische Komplexität (n*n). Wird nur der minimale Abstand der Konturen benötigt, kann auch der Operator distance_cc_min verwendet werden, welcher Algorithmen mit einer Komplexität der Ordnung n*log(n) zur Verfügung stellt.

Achtung

Die beiden ikonischen Eingabeparameter müssen die gleiche Anzahl von Konturen enthalten.

Parameter

- ▷ **Contour1** (input_object) ..... xld_cont(-array) ⇝ *object*
  Erste Eingabekontur.
- ▷ **Contour2** (input_object) ..... xld_cont(-array) ⇝ *object*
  Zweite Eingabekontur.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Art der Abstandsberechnung.
  Defaultwert: 'point_to_point'
  Werteliste: Mode ∈ {'point_to_point', 'point_to_segment'}
- ▷ **DistanceMin** (output_control) ..... real(-array) ⇝ *real*
  Minimaler Abstand zwischen beiden Konturen.
- ▷ **DistanceMax** (output_control) ..... real(-array) ⇝ *real*
  Maximaler Abstand zwischen beiden Konturen.

Beispiel

```
gen_contour_polygon_rounded_xld(Cont1, [0,100,100,0,0], [0,0,100,100,0],
                                [50,50,50,50,50], 0.5);
gen_contour_polygon_rounded_xld(Cont2, [41,91,91,41,41], [41,41,91,91,41],
                                [25,25,25,25,25], 0.5);
distance_cc(Cont1, Cont2, 'point_to_point', &distance_min, &distance_max);
```

Ergebnis

distance_cc liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

distance_sc, distance_pc, distance_cc_min, distance_contours_xld

Siehe auch

distance_sr, distance_pr

Modul

Foundation

distance_cc_min (Contour1, Contour2 : : Mode : DistanceMin)

Berechnen des minimalen Abstandes zwischen zwei Konturen.

distance_cc_min berechnet den minimalen Abstand zwischen zwei Konturen. Als Eingabe werden die beiden Konturen Contour1, Contour2 erwartet. Das Ergebnis wird in DistanceMin übergeben.

Der Parameter Mode gibt dabei an, auf welche Weise der Abstand berechnet wird. *'point_to_point'* bestimmt nur die nächstgelegenen Konturpunkte und deren Abstand, *'fast_point_to_segment'* berechnet den Abstand der Liniensegmente in der Umgebung dieser Eckpunkte und *'point_to_segment'* bestimmt den tatsächlichen minimalen Abstand der Kontursegmente.

Während *'point_to_point'* und *'fast_point_to_segment'* effiziente Algorithmen der Komplexität n*log(n) sind, hat *'point_to_segment'* quadratische Komplexität, die sich in einer längeren Rechenzeit niederschlägt, insbesondere bei Konturen mit vielen Liniensegmenten.

Achtung

Die beiden ikonischen Eingabeparameter müssen die gleiche Anzahl von Konturen enthalten.

Parameter

- ▷ **Contour1** (input_object) ..... xld_cont(-array) ⇝ *object*
  Erste Eingabekontur.
- ▷ **Contour2** (input_object) ..... xld_cont(-array) ⇝ *object*
  Zweite Eingabekontur.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Art der Abstandsberechnung.
  Defaultwert: 'fast_point_to_segment'
  Werteliste: Mode ∈ {'point_to_point', 'point_to_segment', 'fast_point_to_segment'}
- ▷ **DistanceMin** (output_control) ..... number(-array) ⇝ *real*
  Minimaler Abstand zwischen beiden Konturen.

Beispiel

```
gen_contour_polygon_rounded_xld(Cont1, [0,100,100,0,0], [0,0,100,100,0],
                                [50,50,50,50,50], 0.5);
gen_contour_polygon_rounded_xld(Cont2, [41,91,91,41,41], [41,41,91,91,41],
                                [25,25,25,25,25], 0.5);
distance_cc_min(Cont1, Cont2, "fast_point_to_segment", &distance_min);
```

Ergebnis

distance_cc_min liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

distance_sc, distance_pc, distance_cc, distance_contours_xld

Siehe auch

distance_sr, distance_pr

Modul

Foundation

distance_cc_min_points (Contour1, Contour2 : : Mode : DistanceMin, Row1, Column1, Row2, Column2)

Berechnen des minimalen Abstandes zwischen zwei Konturen und Zurückgeben der Punkte, die für die Berechnung benutzt wurden.

distance_cc_min berechnet den minimalen Abstand zwischen zwei Konturen. Als Eingabe werden die beiden Konturen Contour1, Contour2 erwartet. Das Ergebnis wird in DistanceMin zurückgegeben. Im Vergleich zu distance_cc_min werden auch die Punkte auf den Eingabekonturen zurückgegeben, zwischen denen der minimale Abstand gemessen wurde. Der Punkt auf Contour1 wird in Row1 und Column1 zurückgegeben; der Punkt auf Contour2 wird in Row2 und Column2 zurückgegeben.

Der Parameter Mode gibt dabei an, auf welche Weise der Abstand berechnet wird. *'fast_point_to_segment'* berechnet den Abstand der Liniensegmente in der Umgebung der nächstgelegenen Konturpunkte. *'point_to_segment'* bestimmt den tatsächlichen minimalen Abstand der Kontursegmente.

Während *'fast_point_to_segment'* ein effizienter Algorithmen der Komplexität n*log(n) ist, hat *'point_to_segment'* quadratische Komplexität, die sich in einer längeren Rechenzeit niederschlägt, insbesondere bei Konturen mit vielen Liniensegmenten.

Achtung

Die beiden ikonischen Eingabeparameter müssen die gleiche Anzahl von Konturen enthalten.

Parameter

- ▷ **Contour1** (input_object) ..... xld_cont(-array) ⇝ *object*
  Erste Eingabekontur.
- ▷ **Contour2** (input_object) ..... xld_cont(-array) ⇝ *object*
  Zweite Eingabekontur.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Art der Abstandsberechnung.
  Defaultwert: 'fast_point_to_segment'
  Werteliste: Mode ∈ {'point_to_segment', 'fast_point_to_segment'}
- ▷ **DistanceMin** (output_control) ..... number(-array) ⇝ *real*
  Minimaler Abstand zwischen beiden Konturen.
- ▷ **Row1** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinate des Punktes auf Contour1.
- ▷ **Column1** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinate des Punktes auf Contour1.
- ▷ **Row2** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinate des Punktes auf Contour2.
- ▷ **Column2** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinate des Punktes auf Contour2.

Beispiel

```
gen_contour_polygon_rounded_xld(Cont1, [0,100,100,0,0], [0,0,100,100,0],
                                [50,50,50,50,50], 0.5);
gen_contour_polygon_rounded_xld(Cont2, [41,91,91,41,41], [41,41,91,91,41],
                                [25,25,25,25,25], 0.5);
distance_cc_min_points(Cont1, Cont2, "fast_point_to_segment", &distance_min,
                       &Row1, &Column1, &Row2, &Column2);
```

Ergebnis

distance_cc_min_points liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

distance_sc, distance_pc, distance_cc, distance_contours_xld, distance_cc_min

Siehe auch

distance_sr, distance_pr

Modul

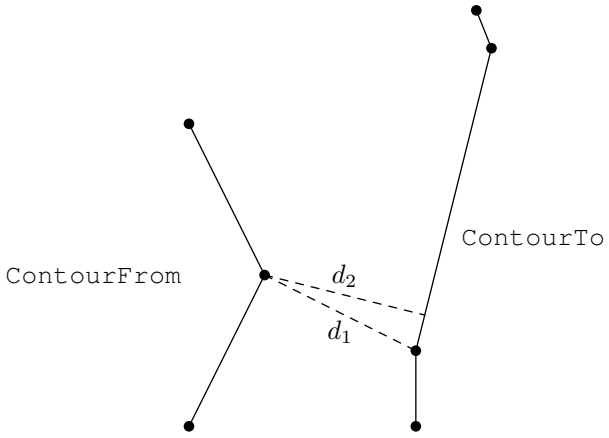
Foundation

distance_contours_xld (ContourFrom, ContourTo : ContourOut : Mode :)

Berechnen des punktweisen Abstands einer Kontur zu einer anderen.

Der Operator distance_contours_xld berechnet für jeden Punkt in ContourFrom den minimalen Abstand zu den in ContourTo übergebenen Konturen. Die zurückgelieferte Kontur ContourOut enthält ContourFrom mit dem Attribut *'distance'*, in dem die berechneten Abstände gespeichert sind. Die Abstände können durch die Abfrage des Attributs *'distance'* mit dem Operator get_contour_attrib_xld abgerufen werden. Weitere Informationen zu Konturattributen können der Referenz von get_contour_attrib_xld entnommen werden.

Der Parameter Mode legt fest, welche Abstände für jeden Punkt in ContourFrom berechnet werden: *'point_to_point'* berechnet den minimalen Abstand zu den Stützpunkten von ContourTo. Dagegen berechnet *'point_to_segment'* den minimalen Abstand zu den Kontursegmenten von ContourTo (siehe auch nachstehende Abbildung).



Die gestrichelten Linien stellen dar, welcher Abstand ausgehend vom Punkt in ContourFrom links berechnet wird. Der Abstand d_1 entspricht dem Modus *'point_to_point'*, der Abstand d_2 dem Modus *'point_to_segment'*.

In vielen Anwendungen bietet sich *'point_to_segment'* für die eigentlich korrekte Lösung an, während *'point_to_point'* deutlich schneller ist. Bezeichnet m die Anzahl der Punkte in ContourFrom, n die Anzahl der Punkte in ContourTo, dann hat distance_contours_xld die Komplexität O(mlog(n)) für *'point_to_point'*, und O(mn) für *'point_to_segment'*. Falls die Referenzkontur wiederholt verwendet wird, können die Operatoren create_distance_transform_xld und apply_distance_transform_xld alternativ zu distance_contours_xld benutzt werden. Nähere Information hierzu finden sich in der Dokumentation zu create_distance_transform_xld.

Parameter

▷ **ContourFrom** (input_object) ..... xld_cont(-array) ⇝ *object*
Konturen, für deren Punkte die Abstände berechnet werden.

▷ **ContourTo** (input_object) ..... xld_cont(-array) ⇝ *object*
Konturen, zu denen die Abstände berechnet werden.

▷ **ContourOut** (output_object) ..... xld_cont(-array) ⇝ *object*
Kopie von `ContourFrom` mit den Abständen als Attribut.

▷ **Mode** (input_control) ..... string ⇝ *string*
Berechne die Distanz zu Punkten ('point_to_point') oder zu ganzen Segmenten ('point_to_segment').
Defaultwert: 'point_to_point'
Werteliste: `Mode` ∈ {'point_to_point', 'point_to_segment'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert der Operator den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`get_contour_attrib_xld`, `segment_contour_attrib_xld`

Alternativen

`apply_distance_transform_xld`, `distance_cc`, `distance_cc_min`

Modul

Foundation

distance_lc (Contour : : Row1, Column1, Row2, Column2 : DistanceMin, DistanceMax)

Berechnen des Abstandes zwischen einer Geraden und einer Kontur.

`distance_lc` berechnet den orthogonalen Abstand zwischen einer Geraden und einer Kontur, repräsentiert durch ihre Liniensegmente. Als Eingabe werden die Koordinaten von zwei Punkten (`Row1`,`Column1`, `Row2`,`Column2`), die eine Gerade darstellen, und eine Kontur (`Contour`) erwartet. Das Ergebnis wird dann in den Parametern `DistanceMin` und `DistanceMax` zurückgeliefert.

Parameter

▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
Eingabekontur.

▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
Zeilenkoordinate des ersten Punktes der Geraden.

▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
Spaltenkoordinate des ersten Punktes der Geraden.

▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
Zeilenkoordinate des zweiten Punktes der Geraden.

▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
Spaltenkoordinate des zweiten Punktes der Geraden.

▷ **DistanceMin** (output_control) ..... real(-array) ⇝ *real*
Minimaler Abstand zwischen der Geraden und der Kontur.

▷ **DistanceMax** (output_control) ..... real(-array) ⇝ *real*
Maximaler Abstand zwischen der Geraden und der Kontur.

Ergebnis

`distance_lc` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

distance_pc, distance_sc, distance_cc, distance_cc_min

Siehe auch

distance_lr, distance_pr, distance_sr

Modul

Foundation

distance_lr (Region : : Row1, Column1, Row2, Column2 : DistanceMin, DistanceMax)

Berechnen des Abstandes zwischen einer Geraden und einer Region.

distance_lr berechnet den orthogonalen Abstand zwischen einer Geraden und einer Region. Als Eingabe werden die Koordinaten von jeweils zwei Punkten (Row1,Column1, Row2,Column2), welche eine Gerade darstellen, und eine Region erwartet. Das Ergebnis wird dann in den Parametern DistanceMin und DistanceMax übergeben.

Achtung

Aus Laufzeitgründen werden in distance_lr Löcher der Region nicht berücksichtigt.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *object*
  Eingaberegion.
- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Geraden.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Geraden.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Geraden.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Geraden.
- ▷ **DistanceMin** (output_control) ..... real(-array) ⇝ *real*
  Minimaler Abstand zwischen der Geraden und der Region
- ▷ **DistanceMax** (output_control) ..... real(-array) ⇝ *real*
  Maximaler Abstand zwischen der Geraden und der Region

Beispiel

```
gen_circle (Circle, 200, 200, 100.5)
Column1 := 300
Column2 := 400
for Row := 50 to 350 by 50
  gen_contour_polygon_xld (Line, [Row,Row], [Column1,Column2])
  distance_lr (Circle, Row, Column1, Row, Column2, DistanceMin, DistanceMax)
endfor
```

Ergebnis

distance_lr liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

distance_lc, distance_pr, distance_sr, diameter_region

Siehe auch

hamming_distance, select_region_point, test_region_point, smallest_rectangle2

Modul

Foundation

distance_pc (Contour : : Row, Column : DistanceMin, DistanceMax)

Berechnen des Abstandes zwischen Punkten und einer Kontur.

distance_pc berechnet den Abstand zwischen Punkten und einer Kontur. Als Eingabe werden die Spalten- und Zeilenkoordinaten der Punkte (Row,Column) und eine Kontur (Contour) erwartet. Das Ergebnis wird in den Parametern DistanceMin und DistanceMax zurückgegeben. Die Abstände bezeichnen die Abstände zwischen den Punkten und den Konturseglementen und nicht die Abstände zwischen den Punkten und den Stützpunkten der Kontur (siehe auch distance_contours_xld).

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
  Eingabekontur.
- ▷ **Row** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Punktes.
- ▷ **Column** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Punktes.
- ▷ **DistanceMin** (output_control) ..... real(-array) ⇝ *real*
  Minimaler Abstand zwischen dem Punkt und der Kontur.
- ▷ **DistanceMax** (output_control) ..... real(-array) ⇝ *real*
  Maximaler Abstand zwischen dem Punkt und der Kontur.

Ergebnis

distance_pc liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

distance_lc, distance_sc, distance_cc, distance_cc_min

Siehe auch

distance_pr, distance_lr, distance_sr, hamming_distance, select_xld_point, test_xld_point

Modul

Foundation

distance_pl (: : Row, Column, Row1, Column1, Row2, Column2 : Distance)

Berechnen des Abstandes zwischen einem Punkt und einer Geraden.

distance_pl berechnet den orthogonalen Abstand von Punkten (Row,Column) zu Geraden, gegeben durch zwei beliebige Punkte auf der Geraden. Das Ergebnis wird im Parameter Distance übergeben.

Es ist möglich die Distanzen zwischen n Punkten und einer Geraden oder die Distanzen zwischen n Punkten und n Geraden zu berechnen.

Parameter

- ▷ **Row** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Punktes.
- ▷ **Column** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Punktes.
- ▷ **Row1** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Geraden.
- ▷ **Column1** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Geraden.
- ▷ **Row2** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Geraden.
- ▷ **Column2** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Geraden.
- ▷ **Distance** (output_control) . . . . . real(-array) ⇝ *real*
  Abstand zwischen den Punkten.

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
draw_point (WindowHandle, Row, Column)
gen_cross_contour_xld (Cross, Row, Column, 15, 0)
draw_line (WindowHandle, Row1, Column1, Row2, Column2)
gen_contour_polygon_xld (Contour, [Row1,Row2], [Column1,Column2])
distance_pl (Row, Column, Row1, Column1, Row2, Column2, Distance)
```

Ergebnis

`distance_pl` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`distance_ps`

Siehe auch

`distance_pp`, `distance_pr`

Modul

Foundation

distance_pp (: : Row1, Column1, Row2, Column2 : Distance)

Berechnen des Abstandes zwischen zwei Punkten.

`distance_pp` berechnet den Abstand zwischen Paaren von Punkten. Der Abstand berechnet sich wie folgt:

$$\texttt{Distance} = \sqrt{((\texttt{Row1} - \texttt{Row2})^2 + (\texttt{Column1} - \texttt{Column2})^2)}$$

Das Ergebnis wird im Parameter `Distance` übergeben.

Parameter

- ▷ **Row1** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes.
- ▷ **Column1** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes.
- ▷ **Row2** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes.
- ▷ **Column2** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes.
- ▷ **Distance** (output_control) . . . . . real(-array) ⇝ *real*
  Abstand zwischen den Punkten.

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
draw_point (WindowHandle, Row1, Column1)
gen_cross_contour_xld (Cross, Row1, Column1, 15, 0)
draw_point (WindowHandle, Row2, Column2)
gen_cross_contour_xld (Cross, Row2, Column2, 15, 0)
distance_pp (Row1, Column1, Row2, Column2, Distance)
```

Ergebnis

`distance_pp` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`distance_ps`

Siehe auch

`distance_pl`, `distance_pr`

Modul

Foundation

distance_pr (Region : : Row, Column : DistanceMin, DistanceMax)

Berechnen des Abstandes zwischen einem Punkt und einer Region.

`distance_pr` berechnet den Abstand zwischen einem Punkt und einer Region. Als Eingabe werden die Spalten- und Zeilenkoordinate der Punkte (`Row`,`Column`) und eine Region erwartet. Falls ein Punkt innerhalb der Region liegt, ist sein minimaler Abstand Null. Das Ergebnis wird in den Parametern `DistanceMin` und `DistanceMax` übergeben.

Parameter

- ▷ **Region** (input_object) . . . . . region ⇝ *object*
  Eingaberegion.
- ▷ **Row** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Punktes.
- ▷ **Column** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Punktes.
- ▷ **DistanceMin** (output_control) . . . . . real(-array) ⇝ *real*
  Minimaler Abstand zwischen dem Punkt und der Region.
- ▷ **DistanceMax** (output_control) . . . . . real(-array) ⇝ *real*
  Maximaler Abstand zwischen dem Punkt und der Region.

Beispiel

```
gen_circle (Circle, 200, 200, 100.5)
draw_point (WindowHandle, Row, Column)
gen_cross_contour_xld (Cross, Row, Column, 15, 0)
distance_pr (Circle, Row, Column, DistanceMin, DistanceMax)
```

Ergebnis

`distance_pr` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`distance_pc`, `distance_lr`, `distance_sr`, `diameter_region`

Siehe auch

`hamming_distance`, `select_region_point`, `test_region_point`, `smallest_rectangle2`

Modul

Foundation

distance_ps (: : Row, Column, Row1, Column1, Row2, Column2 : DistanceMin, DistanceMax)

Berechnen der Abstände zwischen einem Punkt und einer Strecke.

`distance_ps` berechnet den minimalen und den maximalen Abstand zwischen einem Punkt (`Row`,`Column`) und einer Strecke. Die Strecke wird durch ihre Anfangs- (`Row1`,`Column1`) und Endkoordinate (`Row2`,`Column2`) beschrieben. `DistanceMax` ist der maximale Abstand zu den beiden Endpunkten der Strecke. `DistanceMin` liefert den Abstand senkrecht zur Strecke (analog zu `distance_pl`), falls der Punkt „zwischen“ den Endpunkten liegt. Ansonsten wird der minimale Anstand zu einem der Endpunkte verwendet.

Parameter

- ▷ **Row** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes.
- ▷ **Column** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes.
- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Strecke.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Strecke.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Strecke.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Strecke.
- ▷ **DistanceMin** (output_control) ..... real(-array) ⇝ *real*
  Minimaler Abstand zwischen Punkt und Strecke.
- ▷ **DistanceMax** (output_control) ..... real(-array) ⇝ *real*
  Maximaler Abstand zwischen Punkt und Strecke.

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
draw_point (WindowHandle, Row, Column)
```

```
gen_cross_contour_xld (Cross, Row, Column, 15, 0)
draw_line (WindowHandle, Row1, Column1, Row2, Column2)
gen_contour_polygon_xld (Contour, [Row1,Row2], [Column1,Column2])
distance_ps (Row, Column, Row1, Column1, Row2, Column2, \
             DistanceMin, DistanceMax)
```

Ergebnis

`distance_ps` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`distance_pl`

Siehe auch

`distance_pp`, `distance_pr`

Modul

Foundation

distance_rr_min (Regions1, Regions2 : : : MinDistance, Row1, Column1, Row2, Column2)

Ermitteln des minimalen Abstands zwischen den Konturpunkten von je zwei Regionen.

`distance_rr_min` berechnet den minimalen Abstand von Paaren von Regionen. Falls mehrere Regionen in `Regions1` und `Regions2` übergeben werden, wird der Abstand zwischen den Konturpunkten der jeweils i-ten Elementen berechnet. Dieser bildet dann den i-ten Eintrag im Ausgabeparameter `MinDistance`. Es wird der euklidische Abstand berechnet. Die Parameter (`Row1`,`Column1`) bzw. (`Row2`, `Column2`) geben die Position auf der Kontur von `Regions1` bzw. `Regions2` an, zwischen denen der Abstand minimal ist.

Die Berechnung erfolgt durch den Vergleich aller Konturpunkte (siehe `get_region_contour`). Dies bedeutet insbesondere, dass Hohlflächen innerhalb der Regionen nicht berücksichtigt werden. Weiterhin wird nicht überprüft, ob eine Region vollständig innerhalb der anderen Region liegt. In diesem Fall wird ein Abstand > 0 zurückgegeben. Es wird auch nicht überprüft, ob die beiden Regionen einen nichtleeren Schnitt besitzen. Im diesem Fall kann, je nachdem, ob die Konturen der Regionen einen gemeinsamen Punkt besitzen oder nicht, ein minimaler Abstand von 0 oder von > 0 als Ergebnis zurückgeliefert werden.

Achtung

Beide Eingabeparameter müssen die gleiche Anzahl von Regionen enthalten. Die Regionen dürfen nicht leer sein.

Parameter

- ▷ **Regions1** (input_object) . . . . . region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **Regions2** (input_object) . . . . . region(-array) ⇝ *object*
  Zu untersuchende Regionen.
- ▷ **MinDistance** (output_control) . . . . . real(-array) ⇝ *real*
  Minimaler Abstand zwischen Konturen der Regionen.
  Zusicherung: 0 <= MinDistance
- ▷ **Row1** (output_control) . . . . . point.y(-array) ⇝ *integer*
  Zeilenindex auf Kontur in `Regions1`.
- ▷ **Column1** (output_control) . . . . . point.x(-array) ⇝ *integer*
  Spaltenindex auf Kontur in `Regions1`.
- ▷ **Row2** (output_control) . . . . . point.y(-array) ⇝ *integer*
  Zeilenindex auf Kontur in `Regions2`.

▷ **Column2** (output_control) . . . . . point.x(-array) ⇝ *integer*
Spaltenindex auf Kontur in Regions2.

Komplexität

Seien $N1$,$N2$ die Längen der Konturen, dann beträgt die Laufzeitkomplexität $O(N1 * N2)$.

Ergebnis

distance_rr_min liefert den Wert TRUE, falls die Eingabe nicht leer ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

distance_rr_min_dil, dilation1, intersection

Modul

Foundation

distance_rr_min_dil (Regions1, Regions2 : : : MinDistance)

Ermitteln des minimalen Abstands zwischen je zwei Regionen mit Hilfe der Dilatation.

distance_rr_min_dil berechnet den minimalen Abstand von Paaren von Regionen durch iterative Dilation der Regionen bis sie sich überlappen. Falls mehrere Regionen in Regions1 und Regions2 übergeben werden, wird der Abstand zwischen den jeweils i-ten Elementen berechnet. Dieser bildet dann den i-ten Eintrag im Ausgabeparameter MinDistance. Die Berechnung erfolgt mit Hilfe der Dilatation mit dem Golay-Element *'h'* (siehe dilation_golay für weitere Informationen). Das Ergebnis ist:

$n_i * 2 - 1$ wobei n_i die Anzahl an Iterationen ist. Die Maske *'h'* bewirkt, dass gerade die Maximums-Metrik berechnet wird.

Achtung

Beide Parameter müssen die gleiche Anzahl von Regionen enthalten. Die Regionen dürfen nicht leer sein.

Parameter

▷ **Regions1** (input_object) . . . . . region(-array) ⇝ *object*
Zu untersuchende Regionen.

▷ **Regions2** (input_object) . . . . . region(-array) ⇝ *object*
Zu untersuchende Regionen.

▷ **MinDistance** (output_control) . . . . . integer(-array) ⇝ *integer*
Minimale Abstände der Regionen.
Zusicherung: -1 <= MinDistance

Ergebnis

distance_rr_min_dil liefert den Wert TRUE, falls die Eingabe nicht leer ist. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

threshold, regiongrowing, connection

Alternativen

distance_rr_min, dilation1, intersection

Modul

Foundation

distance_sc (Contour : : Row1, Column1, Row2, Column2 : DistanceMin, DistanceMax)

Berechnen des Abstandes zwischen einer Strecke und einer Kontur.

distance_sc berechnet den Abstand zwischen einer Strecke und den Liniensegmenten einer Kontur. Als Eingabe werden der Anfangs- und Endpunkt einer Strecke (Row1,Column1,Row2,Column2) und eine Kontur (Contour) erwartet. Das Ergebnis wird in den Parametern DistanceMin und DistanceMax übergeben.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
  Eingabekontur.
- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Strecke.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Strecke.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Strecke.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Strecke.
- ▷ **DistanceMin** (output_control) ..... real(-array) ⇝ *real*
  Minimaler Abstand zwischen der Strecke und der Kontur.
- ▷ **DistanceMax** (output_control) ..... real(-array) ⇝ *real*
  Maximaler Abstand zwischen der Strecke und der Kontur.

Ergebnis

distance_sc liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

distance_lc, distance_pc, distance_cc, distance_cc_min

Siehe auch

distance_sr, distance_lr, distance_pr, select_xld_point, test_xld_point

Modul

Foundation

distance_sl (: : RowA1, ColumnA1, RowA2, ColumnA2, RowB1, ColumnB1, RowB2, ColumnB2 : DistanceMin, DistanceMax)

Berechnen der Abstände zwischen einer Strecke und einer Geraden.

distance_sl berechnet den minimalen und den maximalen orthogonalen Abstand zwischen einer Strecke und einer Geraden. Als Eingabe werden die Koordinaten der Strecke (RowA1,ColumnA1,RowA2,ColumnA2) und der Geraden (RowB1,ColumnB1,RowB2,ColumnB2) erwartet. Das Ergebnis, d.h. minimaler und maximaler

Abstand, wird dann im Parameter DistanceMin und DistanceMax übergeben. Falls sich die beiden Strecken schneiden, dann ist DistanceMin gleich Null.

Parameter

- ▷ **RowA1** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Strecke.
- ▷ **ColumnA1** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Strecke.
- ▷ **RowA2** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Strecke.
- ▷ **ColumnA2** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Strecke.
- ▷ **RowB1** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Geraden.
- ▷ **ColumnB1** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Geraden.
- ▷ **RowB2** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Geraden.
- ▷ **ColumnB2** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Geraden.
- ▷ **DistanceMin** (output_control) . . . . . real(-array) ⇝ *real*
  Minimaler Abstand zwischen Strecke und Geraden.
- ▷ **DistanceMax** (output_control) . . . . . real(-array) ⇝ *real*
  Maximaler Abstand zwischen Strecke und Geraden.

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
Row1 := 300
Column1 := 200
Row2 := 100
Column2 := 400
gen_contour_polygon_xld (Line1, [Row1,Row2], [Column1,Column2])
dev_display (Line1)
Column := 50
Row := 100
Offset := 0
for RowEnd := 100 to 500 by 100
  gen_contour_polygon_xld (Line2, [Row,RowEnd], \
                           [Column+Offset,Column+Offset])
  dev_display (Line1)
  distance_sl (Row, Column+Offset, RowEnd, Column+Offset, Row1, Column1, \
               Row2, Column2,DistanceMin, DistanceMax)
  Offset := Offset+40
endfor
```

Ergebnis

distance_sl liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

distance_pl

Siehe auch

distance_ps, distance_pp

Modul

Foundation

distance_sr (Region : : Row1, Column1, Row2, Column2 : DistanceMin, DistanceMax)

Berechnen des Abstandes zwischen einer Strecke und einer Region.

distance_sr berechnet den Abstand zwischen einer Strecke und einer Region. Als Eingabe werden die Anfangs- und Endpunkte der jeweiligen Strecke (Row1,Column1,Row2,Column2) und eine Region erwartet. Das entsprechende Ergebnis wird in den Parametern DistanceMin und DistanceMax übergeben.

Achtung

Aus Laufzeitgründen werden in distance_sr Löcher der Region nicht berücksichtigt.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *object*
  Eingaberegion.
- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Strecke.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Strecke.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Strecke.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Strecke.
- ▷ **DistanceMin** (output_control) ..... real(-array) ⇝ *real*
  Minimaler Abstand zwischen der Strecke und der Region.
- ▷ **DistanceMax** (output_control) ..... real(-array) ⇝ *real*
  Maximaler Abstand zwischen der Strecke und der Region.

Beispiel

```
gen_circle (Circle, 200, 200, 100.5)
Column1 := 300
Column2 := 400
for Row := 50 to 350 by 50
  gen_contour_polygon_xld (Line, [Row,Row], [Column1,Column2])
  distance_sr (Circle, Row, Column1, Row, Column2, DistanceMin, DistanceMax)
endfor
```

Ergebnis

distance_sr liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

distance_sc, distance_lr, distance_pr, diameter_region

Siehe auch

hamming_distance, select_region_point, test_region_point, smallest_rectangle2

Modul

Foundation

distance_ss (: : RowA1, ColumnA1, RowA2, ColumnA2, RowB1, ColumnB1, RowB2, ColumnB2 : DistanceMin, DistanceMax)

Berechnen der Abstände zwischen zwei Strecken.

distance_ss berechnet den minimalen und den maximalen Abstand zwischen zwei Strecken. Als Eingabe werden die Koordinaten des Start- und Endpunktes der ersten Strecke (RowA1,ColumnA1, RowA2,ColumnA2) und der zweiten Strecke (RowB1,ColumnB1,RowB2,ColumnB2) erwartet. Das Ergebnis, d.h. minimaler und maximaler Abstand, wird dann im Parameter DistanceMin und DistanceMax übergeben. Falls sich die beiden Strecken schneiden, dann ist DistanceMin gleich Null.

Parameter

- ▷ **RowA1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der ersten Strecke.
- ▷ **ColumnA1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der ersten Strecke.
- ▷ **RowA2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der ersten Strecke.
- ▷ **ColumnA2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der ersten Strecke.
- ▷ **RowB1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der zweiten Strecke.
- ▷ **ColumnB1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der zweiten Strecke.
- ▷ **RowB2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der zweiten Strecke.
- ▷ **ColumnB2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der zweiten Strecke.
- ▷ **DistanceMin** (output_control) ..... real(-array) ⇝ *real*
  Minimaler Abstand zwischen den Strecken.
- ▷ **DistanceMax** (output_control) ..... real(-array) ⇝ *real*
  Maximaler Abstand zwischen den Strecken.

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
Row1 := 300
Column1 := 200
Row2 := 100
Column2 := 300
gen_contour_polygon_xld (Line1, [Row1,Row2], [Column1,Column2])
Column := 100
Offset := 30
for Row := 40 to 100 by 20
  gen_contour_polygon_xld (Line2, [Row, Row+Offset], \
                           [Column, Column+Offset])
  distance_ss (Row, Column, Row+Offset, Column+Offset, Row1, Column1, \
               Row2, Column2, DistanceMin, DistanceMax)
  Offset := Offset+50
endfor
```

Ergebnis

distance_ss liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

distance_pp

Siehe auch

distance_pl, distance_ps

Modul

Foundation

get_distance_transform_xld_contour (: Contour : DistanceTransformID :)

Zugreifen auf die Referenzkontur, mit der die XLD-Distanztransformation erzeugt wurde.

get_distance_transform_xld_contour liefert die Referenzkontur Contour zurück, die für die Erzeugung der XLD-Distanztransformation DistanceTransformID verwendet wurde. Das kann zur Visualisierung beim Vergleich einer gegebenen Kontur mit der Referenzkontur mit Hilfe des Operators apply_distance_transform_xld verwendet werden.

Parameter

▷ **Contour** (output_object) ..... xld_cont(-array) ⇝ *object*
Template-Kontur(en).

▷ **DistanceTransformID** (input_control) ..... xld_dist_trans ⇝ *handle*
Handle der XLD-Distanztransformation der Kontur(en).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert der Operator den Wert TRUE, sonst eine Fehlermeldung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_distance_transform_xld, deserialize_distance_transform_xld

Siehe auch

get_distance_transform_xld_param, set_distance_transform_xld_param, create_distance_transform_xld, apply_distance_transform_xld, clear_distance_transform_xld, write_distance_transform_xld, serialize_distance_transform_xld

Modul

Foundation

get_distance_transform_xld_param (: : DistanceTransformID, GenParamName : GenParamValue)

Zugreifen auf die Parameter mit denen die XLD-Distanztransformation erzeugt wurde.

get_distance_transform_xld_param greift auf die Parameter zu, die für die Erzeugung der XLD-Distanztransformation DistanceTransformID verwendet wurden. Die Parameternamen werden in GenParamName übergeben und ihre Werte in GenParamValue zurückgeliefert. GenParamName kann die Parameternamen *'mode'* und *'max_distance'* beinhalten.

Parameter

- ▷ **DistanceTransformID** (input_control) . . . . . xld_dist_trans ⇝ *handle*
  Handle der XLD-Distanztransformation.
- ▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
  Namen der generischen Parameter.
  Defaultwert: 'mode'
  Werteliste: GenParamName ∈ {'mode', 'max_distance'}
- ▷ **GenParamValue** (output_control) . . . . . attribute.value(-array) ⇝ *string* / real
  Werte der generischen Parameter.
  Parameteranzahl: GenParamValue == 1 || GenParamValue == 2
  Werteliste: GenParamValue ∈ {'point_to_point', 'point_to_segment', 5.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert write_distance_transform_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_distance_transform_xld, deserialize_distance_transform_xld

Nachfolger

set_distance_transform_xld_param

Siehe auch

get_distance_transform_xld_contour, create_distance_transform_xld, apply_distance_transform_xld, clear_distance_transform_xld, write_distance_transform_xld, serialize_distance_transform_xld

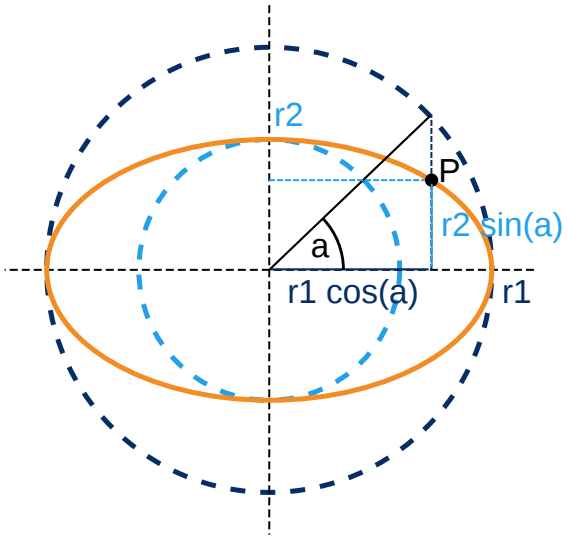
Modul

Foundation

get_points_ellipse (: : Angle, Row, Column, Phi, Radius1, Radius2 : RowPoint, ColPoint)

Berechnen eines Punktes auf dem Perimeter einer Ellipse.

get_points_ellipse liefert den Punkt (RowPoint,ColPoint) auf der übergebenen Ellipse in Abhängigkeit vom Winkel Angle. Der Parameter Angle entspricht dabei der exzentrischen Anomalie und beschreibt den Winkel in der Parameterdarstellung (siehe die Darstellung weiter unten) in Abhängigkeit von der Hauptachse der Ellipse. Die Ellipse selbst wird beschrieben durch ihren Mittelpunkt (Row, Column), die Orientierung der Hauptachse Phi relativ zur horizontalen Bildachse und die Länge der großen (Radius1) bzw. kleinen Halbachse (Radius2). Die Winkel werden in Radiant entgegen dem Uhrzeigersinn gemessen.



Der Punkt `P` mit den Koordinaten `RowPoint` und `ColPoint` gehört zur Ellipse (orange). Er wird über den Winkel `a` und die Längen der beiden Halbachsen `r1` (`Radius1`) und `r2` (`Radius2`) bestimmt. Zur Vereinfachung der Darstellung wurde ein Winkel `Phi`='*0*' verwendet.

Parameter

- ▷ **`Angle`** (input_control) ..... angle.rad(-array) ⇝ *real*
  Winkel des gesuchten Punktes auf der Ellipse [rad].
  Defaultwert: 0
  Restriktion: Angle >= 0 && Angle <= 6.283185307
- ▷ **`Row`** (input_control) ..... ellipse.center.y ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Ellipse.
- ▷ **`Column`** (input_control) ..... ellipse.center.x ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Ellipse.
- ▷ **`Phi`** (input_control) ..... ellipse.angle.rad ⇝ *real*
  Orientierung der Hauptachse in Bogenmaß.
  Restriktion: Phi >= 0 && Phi <= 6.283185307
- ▷ **`Radius1`** (input_control) ..... ellipse.radius1 ⇝ *real*
  Länge der großen Halbachse.
  Restriktion: Radius1 > 0
- ▷ **`Radius2`** (input_control) ..... ellipse.radius2 ⇝ *real*
  Länge der kleinen Halbachse.
  Restriktion: Radius2 >= 0
- ▷ **`RowPoint`** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinate des Ellipsenpunktes.
- ▷ **`ColPoint`** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinate des Ellipsenpunktes.

Beispiel

```
draw_ellipse(WindowHandle,Row,Column,Phi,Radius1,Radius2)
get_points_ellipse([0,3.14],Row,Column,Phi,Radius1,Radius2,\
                   RowPoint,ColPoint)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_points_ellipse` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fit_ellipse_contour_xld, draw_ellipse, gen_ellipse_contour_xld

Siehe auch

gen_ellipse_contour_xld

Modul

Foundation

intersection_circle_contour_xld (Contour : : CircleRow, CircleColumn, CircleRadius, CircleStartPhi, CircleEndPhi, CirclePointOrder : Row, Column)

Berechnet die Schnittpunkte von einem Kreis oder Kreisbogen und einer XLD-Kontur

intersection_circle_contour_xld berechnet die Schnittpunkte von einem Kreis oder Kreisbogen und einer XLD-Kontur Contour. Der Kreis wird über dessen Mittelpunkt (CircleRow,CircleColumn) und Radius CircleRadius definiert. Ein Kreisbogen wird zusätzlich durch den Anfangs- und Endwinkel CircleStartPhi und CircleEndPhi des Start- bzw. Endpunktes, sowie durch dessen Umlaufsinn CirclePointOrder charakterisiert. CirclePointOrder = *'positive'* definiert den Kreisbogen gegen den Uhrzeigersinn. CirclePointOrder = *'negative'* definiert den Kreisbogen im Uhrzeigersinn. Die Schnittpunkte, falls vorhanden, werden in (Row,Column) zurückgegeben.

Parameter

▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
XLD-Kontur.

▷ **CircleRow** (input_control) ..... circle.center.y ⇝ *real* / integer
Zeilenkoordinate des Kreismittelpunktes.

▷ **CircleColumn** (input_control) ..... circle.center.x ⇝ *real* / integer
Spaltenkoordinate des Kreismittelpunktes.

▷ **CircleRadius** (input_control) ..... circle.radius ⇝ *real* / integer
Radius des Kreises.

▷ **CircleStartPhi** (input_control) ..... angle.rad ⇝ *real*
Winkel des Startpunktes des Kreises oder Kreisbogens [rad].
Defaultwert: 0.0

▷ **CircleEndPhi** (input_control) ..... angle.rad ⇝ *real*
Winkel des Endpunktes des Kreises oder Kreisbogens [rad].
Defaultwert: 6.28318

▷ **CirclePointOrder** (input_control) ..... string ⇝ *string*
Mathematischer Umlaufsinn des Kreises oder Kreisbogens.
Defaultwert: 'positive'
Werteliste: CirclePointOrder ∈ {'positive', 'negative'}

▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
Zeilenkoordinate der Schnittpunkte.

▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
Spaltenkoordinate der Schnittpunkte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert intersection_circle_contour_xld den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

intersection_segment_circle

Siehe auch

intersection_line_circle

Modul

Foundation

intersection_circles (: : Circle1Row, Circle1Column, Circle1Radius, Circle1StartPhi, Circle1EndPhi, Circle1PointOrder, Circle2Row, Circle2Column, Circle2Radius, Circle2StartPhi, Circle2EndPhi, Circle2PointOrder : Row, Column, IsOverlapping)

Berechnet die Schnittpunkte von zwei Kreisen oder Kreisbögen

intersection_circles berechnet die Schnittpunkte von zwei Kreisen oder Kreisbögen, die, falls vorhanden, in (Row,Column) zurückgegeben werden. Die Kreise werden über deren Mittelpunkte (Circle1Row,Circle1Column) bzw. (Circle2Row,Circle2Column) und deren Radien Circle1Radius bzw. Circle2Radius definiert. Ein Kreisbogen wird zusätzlich durch den Anfangs- und Endwinkel Circle1StartPhi und Circle1EndPhi bzw. Circle2StartPhi und Circle2EndPhi des Start- bzw. Endpunktes, sowie durch dessen Umlaufsinn Circle1PointOrder bzw. Circle2PointOrder charakterisiert. Circle1PointOrder = *'positive'* definiert den Kreisbogen gegen den Uhrzeigersinn. Circle1PointOrder = *'negative'* definiert den Kreisbogen im Uhrzeigersinn. Das Gleiche gilt für Circle2PointOrder. Liegen Teile der Kreise oder Kreisbögen übereinander, liefert der Parameter IsOverlapping den Wert 1, ansonsten wird der Wert 0 zurückgegeben. In diesem Fall werden die Endpunkte der gemeinsamen Kreisbögen als Schnittpunkte in (Row,Column) zurückgegeben.

Parameter

- ▷ **Circle1Row** (input_control) ..... circle.center.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Kreismittelpunktes.
- ▷ **Circle1Column** (input_control) ..... circle.center.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Kreismittelpunktes.
- ▷ **Circle1Radius** (input_control) ..... circle.radius ⇝ *real* / integer
  Radius des ersten Kreises.
- ▷ **Circle1StartPhi** (input_control) ..... angle.rad ⇝ *real*
  Winkel des Startpunktes des ersten Kreises oder Kreisbogens [rad].
  Defaultwert: 0.0
- ▷ **Circle1EndPhi** (input_control) ..... angle.rad ⇝ *real*
  Winkel des Endpunktes des ersten Kreises oder Kreisbogens [rad].
  Defaultwert: 6.28318
- ▷ **Circle1PointOrder** (input_control) ..... string ⇝ *string*
  Mathematischer Umlaufsinn des ersten Kreises oder Kreisbogens.
  Defaultwert: 'positive'
  Werteliste: Circle1PointOrder ∈ {'positive', 'negative'}
- ▷ **Circle2Row** (input_control) ..... circle.center.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Kreismittelpunktes.
- ▷ **Circle2Column** (input_control) ..... circle.center.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Kreismittelpunktes.
- ▷ **Circle2Radius** (input_control) ..... circle.radius ⇝ *real* / integer
  Radius des ersten Kreises.
- ▷ **Circle2StartPhi** (input_control) ..... angle.rad ⇝ *real*
  Winkel des Startpunktes des ersten Kreises oder Kreisbogens [rad].
  Defaultwert: 0.0
- ▷ **Circle2EndPhi** (input_control) ..... angle.rad ⇝ *real*
  Winkel des Endpunktes des ersten Kreises oder Kreisbogens [rad].
  Defaultwert: 6.28318

▷ **Circle2PointOrder** (input_control) ..... string ⇝ *string*
Mathematischer Umlaufsinn des ersten Kreises oder Kreisbogens.
Defaultwert: 'positive'
Werteliste: Circle2PointOrder ∈ {'positive', 'negative'}

▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
Zeilenkoordinaten der Schnittpunkte.

▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
Spaltenkoordinaten der Schnittpunkte.

▷ **IsOverlapping** (output_control) ..... integer ⇝ *integer*
Liegen Teile der Kreise oder Kreisbogen übereinander?

Ergebnis

Sind die Parameterwerte korrekt, dann liefert intersection_circles den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

intersection_contours_xld (Contour1, Contour2 : : IntersectionType : Row, Column, IsOverlapping)

Berechnet die Schnittpunkte von zwei XLD-Konturen

intersection_contours_xld berechnet die Schnittpunkte der XLD-Konturen Contour1 und Contour2, die, falls vorhanden, in (Row,Column) zurückgegeben werden. Der Parameter IntersectionType beschreibt die Art der Schnittpunkte, die in (Row,Column) ausgegeben werden sollen. Für IntersectionType = *'self'* werden nur die Selbstüberschneidungen innerhalb der Konturen ausgegeben, d.h. die Schnittpunkte innerhalb der Contour1 und innerhalb der Contour2. Für IntersectionType = *'mutual'* werden nur die Schnittpunkte berücksichtigt, die zwischen Contour1 und Contour2 bestehen. Im Standardfall gilt IntersectionType = *'all'*. Hier werden sowohl die Selbstüberschneidungen als auch die gegenseitigen Überschneidungen als Schnittpunkte gewertet. Liegen Teile der XLD-Konturabschnitte übereinander, liefert der Parameter IsOverlapping den Wert 1, ansonsten wird der Wert 0 zurückgegeben. IsOverlapping berücksichtigt sowohl Selbstüberlappungen als auch gegenseitige Überlappungen, unabhängig von IntersectionType. Im Fall einer gegenseitigen Überlappung werden die Endpunkte des gemeinsamen Geradenabschnitts als Schnittpunkte in (Row,Column) zurückgegeben.

Parameter

▷ **Contour1** (input_object) ..... xld_cont ⇝ *object*
Erste XLD-Kontur.

▷ **Contour2** (input_object) ..... xld_cont ⇝ *object*
Zweite XLD-Kontur.

▷ **IntersectionType** (input_control) ..... string ⇝ *string*
Schnittpunkte, die zurückgegeben werden sollen.
Defaultwert: 'all'
Werteliste: IntersectionType ∈ {'all', 'self', 'mutual'}

▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
Zeilenkoordinate der Schnittpunkte.

▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
Spaltenkoordinate der Schnittpunkte.

▷ **IsOverlapping** (output_control) ..... integer ⇝ *integer*
Liegen Teile der XLD-Konturabschnitte übereinander?

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `intersection_contours_xld` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`intersection_segments`, `intersection_segment_contour_xld`

Siehe auch

`intersection_segment_line`, `intersection_lines`, `intersection_line_contour_xld`

Modul

Foundation

intersection_line_circle (: : LineRow1, LineColumn1, LineRow2, LineColumn2, CircleRow, CircleColumn, CircleRadius, CircleStartPhi, CircleEndPhi, CirclePointOrder : Row, Column)

Berechnet die Schnittpunkte von einer Geraden und einem Kreis oder Kreisbogen

`intersection_line_circle` berechnet die Schnittpunkte von einer Geraden und einem Kreis oder Kreisbogen. Die Gerade wird über die beiden Punkte (`LineRow1`,`LineColumn1`) und (`LineRow2`,`LineColumn2`), der Kreis über dessen Mittelpunkt (`CircleRow`,`CircleColumn`) und Radius `CircleRadius` definiert. Ein Kreisbogen wird zusätzlich durch den Anfangs- und Endwinkel `CircleStartPhi` und `CircleEndPhi` des Start- bzw. Endpunktes, sowie durch dessen Umlaufsinn `CirclePointOrder` charakterisiert. `CirclePointOrder` = *'positive'* definiert den Kreisbogen gegen den Uhrzeigersinn. `CirclePointOrder` = *'negative'* definiert den Kreisbogen im Uhrzeigersinn. Die Schnittpunkte, falls vorhanden, werden in (`Row`,`Column`) zurückgegeben.

Parameter

▷ **LineRow1** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des ersten Geradenpunktes.

▷ **LineColumn1** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des ersten Geradenpunktes.

▷ **LineRow2** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des zweiten Geradenpunktes.

▷ **LineColumn2** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des zweiten Geradenpunktes.

▷ **CircleRow** (input_control) . . . . . circle.center.y ⇝ *real* / integer
Zeilenkoordinate des Kreismittelpunktes.

▷ **CircleColumn** (input_control) . . . . . circle.center.x ⇝ *real* / integer
Spaltenkoordinate des Kreismittelpunktes.

▷ **CircleRadius** (input_control) . . . . . circle.radius ⇝ *real* / integer
Radius des Kreises.

▷ **CircleStartPhi** (input_control) . . . . . angle.rad ⇝ *real*
Winkel des Startpunktes des Kreises oder Kreisbogens [rad].
Defaultwert: 0.0

▷ **CircleEndPhi** (input_control) . . . . . angle.rad ⇝ *real*
Winkel des Endpunktes des Kreises oder Kreisbogens [rad].
Defaultwert: 6.28318

▷ **CirclePointOrder** (input_control) . . . . . string ⇝ *string*
Mathematischer Umlaufsinn des Kreises oder Kreisbogens.
Defaultwert: 'positive'
Werteliste: `CirclePointOrder` ∈ {'positive', 'negative'}

▷ **Row** (output_control) . . . . . point.y(-array) ⇝ *real*
Zeilenkoordinaten der Schnittpunkte.

▷ **Column** (output_control) . . . . . point.x(-array) ⇝ *real*
Spaltenkoordinaten der Schnittpunkte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `intersection_line_circle` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`intersection_segment_circle`, `intersection_circle_contour_xld`

Modul

Foundation

intersection_line_contour_xld (Contour : : LineRow1, LineColumn1, LineRow2, LineColumn2 : Row, Column, IsOverlapping)

Berechnet die Schnittpunkte von einer Geraden und einer XLD-Kontur

`intersection_line_contour_xld` berechnet die Schnittpunkte von einer Geraden und einer XLD-Kontur `Contour`. Die Gerade wird über die beiden Punkte (`LineRow1`,`LineColumn1`) und (`LineRow2`,`LineColumn2`) definiert. Die Schnittpunkte, falls vorhanden, werden in (`Row`,`Column`) zurückgegeben. Liegt ein Teil der XLD-Kontur auf der Geraden, liefert der Parameter `IsOverlapping` den Wert 1 zurück, ansonsten wird der Wert 0 zurückgegeben. In diesem Fall werden die Endpunkte des Teilstücks der XLD-Kontur als Schnittpunkte in (`Row`,`Column`) zurückgegeben.

Parameter

▷ **Contour** (input_object) . . . . . xld_cont ⇝ *object*
XLD-Kontur.

▷ **LineRow1** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des ersten Punktes der Geraden.

▷ **LineColumn1** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des ersten Punktes der Geraden.

▷ **LineRow2** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des zweiten Punktes der Geraden.

▷ **LineColumn2** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des zweiten Punktes der Geraden.

▷ **Row** (output_control) . . . . . point.y(-array) ⇝ *real*
Zeilenkoordinaten der Schnittpunkte.

▷ **Column** (output_control) . . . . . point.x(-array) ⇝ *real*
Spaltenkoordinaten der Schnittpunkte.

▷ **IsOverlapping** (output_control) . . . . . integer ⇝ *integer*
Liegt ein Teil der XLD-Kontur auf der Geraden?

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `intersection_line_contour_xld` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

intersection_segment_line

Siehe auch

intersection_segments, intersection_lines, intersection_segment_contour_xld, intersection_contours_xld

Modul

Foundation

intersection_lines (: : Line1Row1, Line1Column1, Line1Row2, Line1Column2, Line2Row1, Line2Column1, Line2Row2, Line2Column2 : Row, Column, IsOverlapping)

Berechnet den Schnittpunkt von zwei Geraden

intersection_lines berechnet den Schnittpunkt von zwei Geraden, die über jeweils zwei Punkte (Line1Row1,Line1Column1) und (Line1Row2,Line1Column2) beziehungsweise (Line2Row1,Line2Column1) und (Line2Row2,Line2Column2) definiert werden. Der Schnittpunkt, falls vorhanden, wird in (Row,Column) zurückgegeben. Sind die beiden Geraden identisch, liefert der Parameter IsOverlapping den Wert 1 zurück, ansonsten wird der Wert 0 zurückgegeben. In diesem Fall wird kein Schnittpunkt in (Row,Column) zurückgegeben.

Parameter

- ▷ **Line1Row1** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der ersten Geraden.
- ▷ **Line1Column1** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der ersten Geraden.
- ▷ **Line1Row2** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der ersten Geraden.
- ▷ **Line1Column2** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der ersten Geraden.
- ▷ **Line2Row1** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der zweiten Geraden.
- ▷ **Line2Column1** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der zweiten Geraden.
- ▷ **Line2Row2** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der zweiten Geraden.
- ▷ **Line2Column2** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der zweiten Geraden.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenkoordinate des Schnittpunktes.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenkoordinate des Schnittpunktes.
- ▷ **IsOverlapping** (output_control) ..... integer ⇝ *integer*
  Sind die beiden Geraden identisch?

Ergebnis

Sind die Parameterwerte korrekt, dann liefert intersection_lines den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

intersection_segments, intersection_segment_line,

intersection_segment_contour_xld, intersection_line_contour_xld, intersection_contours_xld

Modul

Foundation

intersection_segment_circle (: : SegmentRow1, SegmentColumn1, SegmentRow2, SegmentColumn2, CircleRow, CircleColumn, CircleRadius, CircleStartPhi, CircleEndPhi, CirclePointOrder : Row, Column)

Berechnet die Schnittpunkte von einem Geradenabschnitt und einem Kreis oder Kreisbogen

intersection_segment_circle berechnet den Schnittpunkt von einem Geradenabschnitt und einem Kreis oder Kreisbogen. Der Geradenabschnitt wird über dessen Endpunkte (SegmentRow1,SegmentColumn1) und (SegmentRow2,SegmentColumn2), der Kreis über dessen Mittelpunkt (CircleRow,CircleColumn) und Radius CircleRadius definiert. Ein Kreisbogen wird zusätzlich durch den Anfangs- und Endwinkel CircleStartPhi und CircleEndPhi des Start- bzw. Endpunktes, sowie durch dessen Umlaufsinn CirclePointOrder charakterisiert. CirclePointOrder = *'positive'* definiert den Kreisbogen gegen den Uhrzeigersinn. CirclePointOrder = *'negative'* definiert den Kreisbogen im Uhrzeigersinn. Die Schnittpunkte, falls vorhanden, werden in (Row,Column) zurückgegeben.

Parameter

- ▷ **SegmentRow1** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes des Geradenabschnitts.
- ▷ **SegmentColumn1** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes des Geradenabschnitts.
- ▷ **SegmentRow2** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes des Geradenabschnitts.
- ▷ **SegmentColumn2** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes des Geradenabschnitts.
- ▷ **CircleRow** (input_control) ..... circle.center.y ⇝ *real* / integer
  Zeilenkoordinate des Kreismittelpunktes.
- ▷ **CircleColumn** (input_control) ..... circle.center.x ⇝ *real* / integer
  Spaltenkoordinate des Kreismittelpunktes.
- ▷ **CircleRadius** (input_control) ..... circle.radius ⇝ *real* / integer
  Radius des Kreises.
- ▷ **CircleStartPhi** (input_control) ..... angle.rad ⇝ *real*
  Winkel des Startpunktes des Kreises oder Kreisbogens [rad].
  Defaultwert: 0.0
- ▷ **CircleEndPhi** (input_control) ..... angle.rad ⇝ *real*
  Winkel des Endpunktes des Kreises oder Kreisbogens [rad].
  Defaultwert: 6.28318
- ▷ **CirclePointOrder** (input_control) ..... string ⇝ *string*
  Mathematischer Umlaufsinn des Kreises oder Kreisbogens.
  Defaultwert: 'positive'
  Werteliste: CirclePointOrder ∈ {'positive', 'negative'}
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinaten der Schnittpunkte.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinaten der Schnittpunkte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert intersection_segment_circle den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

intersection_circle_contour_xld

Siehe auch

intersection_line_circle

Modul

Foundation

intersection_segment_contour_xld (Contour : : SegmentRow1, SegmentColumn1, SegmentRow2, SegmentColumn2 : Row, Column, IsOverlapping)

Berechnet die Schnittpunkte von einem Geradenabschnitt und einer XLD-Kontur

intersection_segment_contour_xld berechnet die Schnittpunkte von einem Geradenabschnitt und einer XLD-Kontur Contour. Der Geradenabschnitt wird über dessen Endpunkte (SegmentRow1,SegmentColumn1) und (SegmentRow2,SegmentColumn2) definiert. Die Schnittpunkte, falls vorhanden, werden in (Row,Column) zurückgegeben. Liegen ein Teil des Geradenabschnitts und der XLD-Kontur übereinander, liefert der Parameter IsOverlapping den Wert 1 zurück, ansonsten wird der Wert 0 zurückgegeben. In diesem Fall werden die Endpunkte des gemeinsamen Geradenabschnitts als Schnittpunkte in (Row,Column) zurückgegeben.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
  XLD-Kontur.
- ▷ **SegmentRow1** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes des Geradenabschnitts.
- ▷ **SegmentColumn1** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes des Geradenabschnitts.
- ▷ **SegmentRow2** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes des Geradenabschnitts.
- ▷ **SegmentColumn2** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes des Geradenabschnitts.
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinaten der Schnittpunkte.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinaten der Schnittpunkte.
- ▷ **IsOverlapping** (output_control) ..... integer ⇝ *integer*
  Sind ein Teil des Geradenabschnitts und der XLD-Kontur identisch?

Ergebnis

Sind die Parameterwerte korrekt, dann liefert intersection_segment_contour_xld den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

intersection_segments, intersection_contours_xld

Siehe auch

intersection_segment_line, intersection_lines, intersection_line_contour_xld

Modul

Foundation

intersection_segment_line (: : SegmentRow1, SegmentColumn1, SegmentRow2, SegmentColumn2, LineRow1, LineColumn1, LineRow2, LineColumn2 : Row, Column, IsOverlapping)

Berechnet den Schnittpunkt von einem Geradenabschnitt und einer Geraden

intersection_segment_line berechnet den Schnittpunkt von einem Geradenabschnitt und einer Geraden. Der Geradenabschnitt wird über dessen Endpunkte (SegmentRow1,SegmentColumn1) und (SegmentRow2,SegmentColumn2), die Gerade über zwei Punkte (LineRow1,LineColumn1) und (LineRow2,LineColumn2) definiert. Der Schnittpunkt, falls vorhanden, wird in (Row,Column) zurückgegeben. Liegt der Geradenabschnitt auf der Geraden, liefert der Parameter IsOverlapping den Wert 1 zurück, ansonsten wird der Wert 0 zurückgegeben. In diesem Fall werden die Endpunkte des Geradenabschnitts als Schnittpunkte in (Row,Column) zurückgegeben.

Parameter

- ▷ **SegmentRow1** (input_control) . . . . . point.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes des Geradenabschnitts.
- ▷ **SegmentColumn1** (input_control) . . . . . point.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes des Geradenabschnitts.
- ▷ **SegmentRow2** (input_control) . . . . . point.y ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes des Geradenabschnitts.
- ▷ **SegmentColumn2** (input_control) . . . . . point.x ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes des Geradenabschnitts.
- ▷ **LineRow1** (input_control) . . . . . point.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes der Geraden.
- ▷ **LineColumn1** (input_control) . . . . . point.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes der Geraden.
- ▷ **LineRow2** (input_control) . . . . . point.y ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes der Geraden.
- ▷ **LineColumn2** (input_control) . . . . . point.x ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes der Geraden.
- ▷ **Row** (output_control) . . . . . point.y(-array) ⇝ *real*
  Zeilenkoordinate des Schnittpunktes.
- ▷ **Column** (output_control) . . . . . point.x(-array) ⇝ *real*
  Spaltenkoordinate des Schnittpunktes.
- ▷ **IsOverlapping** (output_control) . . . . . integer ⇝ *integer*
  Überlappen sich der Geradenabschnitt und die Gerade in mehr als einem Punkt?

Ergebnis

Sind die Parameterwerte korrekt, dann liefert intersection_segment_line den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

intersection_line_contour_xld

Siehe auch

intersection_segments, intersection_lines, intersection_segment_contour_xld, intersection_contours_xld

Modul

Foundation

intersection_segments (: : Segment1Row1, Segment1Column1, Segment1Row2, Segment1Column2, Segment2Row1, Segment2Column1, Segment2Row2, Segment2Column2 : Row, Column, IsOverlapping)

Berechnet den Schnittpunkt von zwei Geradenabschnitten

intersection_segments berechnet den Schnittpunkt von zwei Geradenabschnitten, die über deren Endpunkte (Segment1Row1,Segment1Column1) und (Segment1Row2,Segment1Column2) beziehungsweise (Segment2Row1,Segment2Column1) und (Segment2Row2,Segment2Column2) definiert werden. Der Schnittpunkt wird, falls vorhanden, in (Row,Column) zurückgegeben. Liegen Teile der Geradenabschnitte übereinander, liefert der Parameter IsOverlapping den Wert 1 zurück, ansonsten wird der Wert 0 zurückgegeben. In diesem Fall werden die Endpunkte des gemeinsamen Geradenabschnitts als Schnittpunkte in (Row,Column) zurückgegeben.

Parameter

- ▷ **Segment1Row1** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes des ersten Geradenabschnitts.
- ▷ **Segment1Column1** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes des ersten Geradenabschnitts.
- ▷ **Segment1Row2** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes des ersten Geradenabschnitts.
- ▷ **Segment1Column2** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes des ersten Geradenabschnitts.
- ▷ **Segment2Row1** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes des zweiten Geradenabschnitts.
- ▷ **Segment2Column1** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes des zweiten Geradenabschnitts.
- ▷ **Segment2Row2** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes des zweiten Geradenabschnitts.
- ▷ **Segment2Column2** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes des zweiten Geradenabschnitts.
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinate des Schnittpunktes.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinate des Schnittpunktes.
- ▷ **IsOverlapping** (output_control) ..... integer ⇝ *integer*
  Liegen Teile der Geradenabschnitte übereinander?

Ergebnis

Sind die Parameterwerte korrekt, dann liefert intersection_segments den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

intersection_segment_contour_xld, intersection_contours_xld

Siehe auch

intersection_segment_line, intersection_lines, intersection_line_contour_xld

Modul

Foundation

projection_pl (: : Row, Column, Row1, Column1, Row2, Column2 : RowProj, ColProj)

Berechnen der Projektion von einem Punkt auf eine Gerade.

`projection_pl` berechnet die Projektion von einem Punkt (`Row`,`Column`) auf eine Gerade. Die Gerade wird durch die zwei Punkte (`Row1`,`Column1`) und (`Row2`,`Column2`) beschrieben. Die Koordinaten des projizierten Punktes werden in `RowProj` und `ColProj` zurückgegeben.

Parameter

- ▷ **`Row`** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des Punktes.
- ▷ **`Column`** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des Punktes.
- ▷ **`Row1`** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des ersten Punktes auf der Geraden.
- ▷ **`Column1`** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des ersten Punktes auf der Geraden.
- ▷ **`Row2`** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinate des zweiten Punktes auf der Geraden.
- ▷ **`Column2`** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinate des zweiten Punktes auf der Geraden.
- ▷ **`RowProj`** (output_control) ..... real(-array) ⇝ *real*
  Zeilenkoordinate des Projektionspunktes.
- ▷ **`ColProj`** (output_control) ..... real(-array) ⇝ *real*
  Spaltenkoordinate des Projektionspunktes

Beispiel

```
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
Row1 := 300
Column1 := 200
Row2 := 140
Column2 := 400
Rows := 300
Columns := 170
dev_set_color ('cadet blue')
gen_contour_polygon_xld (Contour, [Row1,Row2], [Column1,Column2])
Offset := 0
for Rows := 40 to 280 by 40
  dev_set_color ('red')
  gen_cross_contour_xld (Point,Rows+Offset, Columns, 6, 0)
  projection_pl (Rows+Offset, Columns, Row1, Column1, Row2, Column2, \
                 RowProj, ColProj)
  dev_set_color ('blue')
  gen_cross_contour_xld (RowP, RowProj, ColProj, 6, 0)
  Offset := Offset+30
endfor
```

Ergebnis

`projection_pl` liefert den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

read_distance_transform_xld (: : FileName : DistanceTransformID)

Lesen einer XLD-Distanztransformation aus einer Datei.

read_distance_transform_xld liest eine XLD-Distanztransformation, die zuvor mit write_distance_transform_xld abgespeichert wurde, aus der Datei FileName und liefert das Handle DistanceTransformID zurück. Die standardmäßige HALCON Dateiendung für die XLD-Distanztransformation ist 'hdtc'.

get_distance_transform_xld_contour kann verwendet werden, um auf die Referenzkontur zuzugreifen, die für die Erzeugung der XLD-Distanztransformation DistanceTransformID verwendet wurde.

Parameter

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Name der Datei.
Dateiendung: .hdtc

▷ **DistanceTransformID** (output_control) . . . . . xld_dist_trans ⇝ *handle*
Handle der XLD-Distanztransformation.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_distance_transform_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

apply_distance_transform_xld, get_distance_transform_xld_contour, get_distance_transform_xld_param

Siehe auch

write_distance_transform_xld, serialize_distance_transform_xld, deserialize_distance_transform_xld, create_distance_transform_xld, set_distance_transform_xld_param, clear_distance_transform_xld

Modul

Foundation

serialize_distance_transform_xld (
: : DistanceTransformID : SerializedItemHandle)

Serialisieren einer XLD-Distanztransformation.

serialize_distance_transform_xld serialisiert eine XLD-Distanztransformation DistanceTransformID. Die serialisierte XLD-Distanztransformation wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_distance_transform_xld wieder deserialisiert werden.

Parameter

▷ **DistanceTransformID** (input_control) . . . . . xld_dist_trans ⇝ *handle*
Handle der XLD-Distanztransformation.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle der serialisierten XLD-Distanztransformation.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_distance_transform_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_distance_transform_xld`

Nachfolger

`clear_distance_transform_xld`

Alternativen

`write_distance_transform_xld`

Siehe auch

`deserialize_distance_transform_xld`, `read_distance_transform_xld`, `apply_distance_transform_xld`, `get_distance_transform_xld_contour`, `get_distance_transform_xld_param`, `set_distance_transform_xld_param`

Modul

Foundation

set_distance_transform_xld_param (: : DistanceTransformID, GenParamName, GenParamValue :)

Setzen neuer Parameter für die XLD-Distanztransformation.

`set_distance_transform_xld_param` setzt neue Parameter für die XLD-Distanztransformation `DistanceTransformID`. Die Namen und Werte dieser Parameter werden in `GenParamName` und `GenParamValue` übergeben. `GenParamName` kann die Parameternamen *'mode'* und *'max_distance'* beinhalten. Die XLD-Distanztransformation der ursprünglichen Templatekontur wird dann mit aktualisierten Parameterwerten neu berechnet.

Parameter

▷ **DistanceTransformID** (input_control) ..... xld_dist_trans ⇝ *handle*
Handle der XLD-Distanztransformation.

▷ **GenParamName** (input_control) ..... attribute.name(-array) ⇝ *string*
Namen der generischen Parameter.
Defaultwert: 'mode'
Werteliste: `GenParamName` ∈ {'mode', 'max_distance'}

▷ **GenParamValue** (input_control) ..... attribute.value(-array) ⇝ *string* / real
Werte der generischen Parameter.
Defaultwert: 'point_to_point'
Wertevorschläge: `GenParamValue` ∈ {'point_to_point', 'point_to_segment', 5.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_distance_transform_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DistanceTransformID

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`get_distance_transform_xld_param`

Nachfolger

`apply_distance_transform_xld`

Siehe auch

`write_distance_transform_xld`, `serialize_distance_transform_xld`, `get_distance_transform_xld_contour`, `read_distance_transform_xld`, `deserialize_distance_transform_xld`, `clear_distance_transform_xld`

Modul

Foundation

write_distance_transform_xld (: : DistanceTransformID, FileName :)

Schreiben einer XLD-Distanztransformation in eine Datei.

`write_distance_transform_xld` schreibt die XLD-Distanztransformation `DistanceTransformID` in die Datei `FileName`. Die standardmäßige HALCON Dateiendung für die XLD-Distanztransformation ist 'hdtc'.

Die gespeicherte XLD-Distanztransformation kann mit `read_distance_transform_xld` wieder eingelesen werden.

Parameter

▷ **DistanceTransformID** (input_control) ..... xld_dist_trans ⇝ *handle*
Handle der XLD-Distanztransformation.

▷ **FileName** (input_control) ..... filename.write ⇝ *string*
Name der Datei.
Dateiendung: `.hdtc`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_distance_transform_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_distance_transform_xld`

Nachfolger

`clear_distance_transform_xld`

Alternativen

`serialize_distance_transform_xld`

Siehe auch

`read_distance_transform_xld`, `deserialize_distance_transform_xld`, `get_distance_transform_xld_contour`, `apply_distance_transform_xld`, `get_distance_transform_xld_param`, `set_distance_transform_xld_param`

Modul

Foundation

26.3 Gitter-Entzerrung

connect_grid_points (Image : ConnectingLines : Row, Column, Sigma, MaxDist :)

Erstellt Verbindungen zwischen den Gitterpunkten des Entzerrungsgitters.

connect_grid_points sucht die Verbindungslinien zwischen den Gitterpunkten (Row,Column) des Entzerrungsgitters. Die Verbindungslinien werden durch Kombination je eines Kanten-, Glättungs- und Linienfilters der Größe σ aus dem Eingabebild Image extrahiert. Das verwendete σ wird dazu wie folgt bestimmt: Wird in Sigma ein einzelner Wert übergeben, so wird dieser Wert benutzt. Wird in Sigma ein Tupel mit drei Werten (*'sigma_min'*, *'sigma_max'*,*'sigma_step'*) übergeben, so überprüft connect_grid_points alle σ im Bereich von *'sigma_min'* bis *'sigma_max'* mit einer Schrittweite von *'sigma_step'* und wählt jenes σ aus, das die größte Anzahl an Verbindungslinien zur Folge hat. Das gleiche passiert, wenn in Sigma ein Tupel mit nur zwei Werten *'sigma_min'* und *'sigma_max'* übergeben wird. In diesem Fall wird dann eine feste Schrittweite von *0.05* verwendet.

Die gefundenen Verbindungslinien werden dann an den Gitterpunkten aufgebrochen und diejenigen Linienstücke ausgewählt, die sowohl an einem Gitterpunkt beginnen, als auch an einem Gitterpunkt enden. Dabei ist zu beachten, dass Kantenfilter rund um Kreuzungspunkte typischerweise keine genauen Resultate liefern und somit die Verbindungslinien kaum die Gitterpunkte direkt treffen werden. Folglich werden tatsächlich diejenigen Linien aufgebrochen und ausgewählt, die in einer Entfernung von einem Gitterpunkt von höchstens MaxDist beginnen, enden oder den Punkt passieren. Die ausgewählten Verbindungslinien werden noch derart modifiziert, dass sie genau in den jeweiligen Gitterpunkten beginnen und enden, und in ConnectingLines als XLD Konturen zurückgegeben.

Darüberhinaus wird für jede Ausgabekontur ihr Typ von Hell-Dunkel-Übergang berechnet und im globalen Attribut *'bright_dark'* abgespeichert. Das Attribut bekommt den Wert *1.0*, wenn an der Verbindungslinie ein Hell-Dunkel-Übergang stattfindet (von links nach rechts, vom Anfangs- zum Endpunkt gesehen), ansonsten *0.0*. Weitere Informationen zu globalen Konturattributen können der Referenz von get_contour_global_attrib_xld entnommen werden.

Achtung

Sowohl für die zuverlässige Bestimmung des Typs von Hell-Dunkel-Übergang als auch für die spätere Entzerrung ist es unabdinglich, dass jede Verbindungslinie mindestens drei Konturpunkte aufweist. Verbindungslinien mit nur zwei Konturpunkten werden daher nicht ausgegeben. Dabei ist zu beachten, dass der Parameter MaxDist einen erheblichen Einfluss auf die Länge der zurückgegebenen Verbindungslinien hat, da alle Konturpunkte einer möglichen Verbindungslinie, die näher als MaxDist an einem Gitterpunkt liegen, durch einen einzigen Konturpunkt ersetzt werden. Ist MaxDist zu groß, dann gehen unter Umständen einige der Verbindungslinien verloren.

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **ConnectingLines** (output_object) ..... xld-array ⇝ *object*
  Ausgabekonturen.
- ▷ **Row** (input_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der Gitterpunkte.
- ▷ **Column** (input_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der Gitterpunkte.
  Restriktion: number(Column) == number(Row)
- ▷ **Sigma** (input_control) ..... number(-array) ⇝ *integer* / real
  Größe der verwendeten Gaußkerne.
  Parameteranzahl: 1 <= Sigma && Sigma <= 3
  Defaultwert: 0.9
  Wertevorschläge: Sigma ∈ {0.7, 0.9, 1.1, 1.3, 1.5}
  Restriktion: 0.7 <= Sigma
- ▷ **MaxDist** (input_control) ..... number ⇝ *real* / integer
  Maximalabstand der Verbindungslinien von den Gitterpunkten.
  Defaultwert: 5.5
  Wertevorschläge: MaxDist ∈ {1.5, 3.5, 5.5, 7.5, 9.5}
  Restriktion: 0.0 <= MaxDist

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `connect_grid_points` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`saddle_points_sub_pix`

Nachfolger

`gen_grid_rectification_map`

Modul

Calibration

`create_rectification_grid` (: : `Width`, `NumSquares`, `GridFile` :)

Erzeugt eine PostScript-Datei, die das Entzerrungsgitter beschreibt.

`create_rectification_grid` erzeugt ein Schachbrettmuster mit `NumSquares` × `NumSquares` abwechselnd schwarzen und weißen Feldern. Dieses Muster hat eine Breite (und Höhe) von `Width` Metern. Um das Schachbrettmuster verläuft ein innerer Rahmen von der *0.3*-fachen Breite eines Feldes, der das Schachbrettmuster fortsetzt. Den Abschluss bildet ein vollständig weißer, äußerer Rahmen der *0.7*-fachen Breite eines Feldes. Ungefähr in der Mitte des Schachbrettmusters befinden sich zwei kreisförmige Marken, eine schwarze auf weißem Feld und eine weiße auf schwarzem Feld. Der Radius jeder Marke entspricht einem Drittel der Seitenlänge des jeweiligen umschließenden Quadrats.

Die kreisförmigen Marken werden von `gen_grid_rectification_map` benutzt, um gegebenenfalls die gefundene Anordnung der Gitterpunkte in die richtige Lage zu drehen. Dabei wird angenommen, dass die schwarze Marke bei korrekter Ausrichtung links neben der weißen liegt. Die Datei `GridFile` enthält die PostScript-Beschreibung des Entzerrungsgitters.

Parameter

▷ **`Width`** (input_control) . . . . . real ⇝ *real*
Breite des Schachbrettmusters in Meter (ohne die beiden Rahmen).
Defaultwert: 0.17
Wertevorschläge: `Width` ∈ {1.2, 0.8, 0.6, 0.4, 0.2, 0.1}
Empfohlene Schrittweite: 0.1
Restriktion: 0.0 < Width

▷ **`NumSquares`** (input_control) . . . . . integer ⇝ *integer*
Anzahl von Feldern in einer Zeile und Spalte.
Defaultwert: 17
Wertevorschläge: `NumSquares` ∈ {11, 13, 15, 17, 19, 21, 23, 25, 27}
Empfohlene Schrittweite: 2
Restriktion: 2 <= NumSquares

▷ **`GridFile`** (input_control) . . . . . filename.write ⇝ *string*
Dateiname der PostScript-Datei.
Defaultwert: 'rectification_grid.ps'
Dateiendung: `.ps`

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich geschrieben werden, dann liefert `find_rectification_grid` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`find_rectification_grid`, `saddle_points_sub_pix`, `connect_grid_points`, `gen_grid_rectification_map`

Modul

Foundation

`find_rectification_grid` (`Image` : `GridRegion` : `MinContrast`, `Radius` :)

Segmentierung der Entzerrungsgitterregion im Bild.

`find_rectification_grid` sucht die Bereiche im Bild `Image`, die das Entzerrungsgitter enthalten, und gibt sie in der Region `GridRegion` zurück. Im Wesentlichen werden dazu Bildbereiche mit einem Kontrast von wenigstens `MinContrast` extrahiert und die Löcher, die in diesen Bereichen enthalten sind, aufgefüllt. Anschließend werden diese Bereiche einem Opening mit dem Radius `Radius` unterzogen, um kleinere Flächen mit hohem Kontrast zu eliminieren.

Eine sorgfältige Einschränkung der Eingaberegion bei der Gitter-Entzerrung auf die Bildbereiche, die das Entzerrungsgitter enthalten, ist in zweierlei Hinsicht nützlich: Zum Einen kann damit die Rechenzeit eingeschränkt werden, zum Anderen verhindert man so, dass `saddle_points_sub_pix` und `connect_grid_points` falsche Gitterpunkte und Verbindungslinien finden.

Parameter

▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
Eingabebild.

▷ **`GridRegion`** (output_object) ..... region ⇝ *object*
Ausgaberegion die das Entzerrungsgitter enthält.

▷ **`MinContrast`** (input_control) ..... number ⇝ *real* / integer
Minimaler Kontrast.
Defaultwert: 8.0
Wertevorschläge: `MinContrast` ∈ {2.0, 4.0, 8.0, 16.0, 32.0}
Restriktion: MinContrast >= 0

▷ **`Radius`** (input_control) ..... real ⇝ *real* / integer
Radius der Kreismaske.
Defaultwert: 7.5
Wertevorschläge: `Radius` ∈ {1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5}
Restriktion: Radius >= 0.5

Beispiel

```
find_rectification_grid (Image, GridRegion, 8, 10)
dilation_circle (GridRegion, GridRegionDilated, 5.5)
reduce_domain (Image, GridRegionDilated, ImageReduced)
saddle_points_sub_pix (ImageReduced, 'facet', 1.5, 5, Row, Column)
connect_grid_points (ImageReduced, ConnectingLines, Row, Column, 1.1, 5.5)
gen_grid_rectification_map (ImageReduced, ConnectingLines, Map, Meshes, 20, \
                            'auto', Row, Column, 'bilinear')
map_image (Image, Map, ImageMapped)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `find_rectification_grid` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`dilation_circle`, `reduce_domain`

Modul

Calibration

gen_arbitrary_distortion_map (: Map : GridSpacing, Row, Column, GridWidth, ImageWidth, ImageHeight, MapType :)

Berechnet die Abbildung von einem beliebig verzerrten Bild ins entzerrte Bild.

`gen_arbitrary_distortion_map` berechnet die Abbildung `Map` von einem beliebig verzerrten ins entzerrte Bild. Dazu beschreiben die Koordinaten `Row` und `Column` ein (verzerrtes) Punktgitter, das im entzerrten Bild auf ein regelmäßiges Gitter abgebildet wird. Die Koordinaten der Gitterpunkte sind, entsprechend ihrer Anordnung im Punktgitter, zeilenweise von oben nach unten zu übergeben. `GridWidth` ist die Breite des Punktgitters in Gitterpunkten. `GridSpacing` ist der Abstand zweier benachbarter Gitterpunkte im entzerrten Bild. Da Gitterpunkte immer in die Mitte der entsprechenden Pixel projiziert werden, belegt eine Gitterzelle $(\texttt{GridSpacing} + 1) \times (\texttt{GridSpacing} + 1)$ Pixel. Allerdings überlappen sich benachbarte Gitterzellen auch auf einem Pixel Breite.

Darüber hinaus müssen die Breite `ImageWidth` und Höhe `ImageHeight` der zu entzerrenden Bilder übergeben werden, um die Abbildung `Map` zu berechnen.

Der Parameter `MapType` gibt an, in welchem Typ die Abbildungsvorschrift gespeichert werden soll. Wurde die bilineare Interpolation (*'bilinear'*) gewählt, so besteht `Map` aus einem Bild mit fünf Kanälen. Der erste Kanal enthält für jedes Pixel des Ergebnisbildes die linearen Koordinaten desjenigen Pixels des Eingabebildes, welches links oberhalb von den transformierten Koordinaten liegt. Die übrigen vier Kanäle enthalten die Gewichte der einzelnen vier Nachbarpixel der transformierten Koordinaten für die bilineare Interpolation in folgender Reihenfolge:

2	3
4	5

Der zweite Kanal z.B. enthält die Gewichte der Pixel, die links oberhalb der transformierten Koordinaten liegen. Wurde *'coord_map_sub_pix'* gewählt, besteht `Map` aus einem Vektorfeld, welches für jedes Pixel des Ergebnisbildes die subpixelgenauen Koordinaten im Eingabebild enthält.

Aus dem Obigen geht hervor, dass zur Interpolation des Grauwertes eines Pixels im entzerrten Bild immer alle vier benachbarte Pixel im verzerrten Bild benötigt werden. Liegt mindestens eines dieser benachbarten Pixel außerhalb des verzerrten Bildes, so kann der Grauwert nicht berechnet werden. In der resultierenden `Map` wird der Definitionsbereich entsprechend eingeschränkt.

Im Gegensatz zu `gen_grid_rectification_map` wird `gen_arbitrary_distortion_map` verwendet, falls die Koordinaten (`Row`,`Column`) der Gitterpunkte im verzerrten Bild schon bekannt sind bzw. die relevanten Bildinhalte schon reguläre, gitterförmige Strukturen aufweisen, aus denen die Koordinaten gewonnen werden können.

Wenn die erzeugte Abbildung in einem anderen Programm wiederverwendet werden soll, kann man sie mit dem Operator `write_image` als Mehrkanalbild abspeichern (Format: *'tiff'*).

Parameter

▷ **Map** (output_object) ..... multichannel-image ⇝ *object :* int4 / uint2 / vector_field
Bild mit Transformationsdaten.

▷ **GridSpacing** (input_control) ..... integer ⇝ *integer*
Abstand der Gitterpunkte im entzerrten Bild.
Restriktion: GridSpacing > 0

▷ **Row** (input_control) ..... point.y-array ⇝ *real*
Zeilenkoordinaten der Gitterpunkte im verzerrten Bild.

▷ **Column** (input_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinaten der Gitterpunkte im verzerrten Bild.
Restriktion: number(Row) == number(Column)

▷ **GridWidth** (input_control) . . . . . integer ⇝ *integer*
Breite des Punktgitters (Anzahl der Gitterpunkte).

▷ **ImageWidth** (input_control) . . . . . extent.x ⇝ *integer*
Breite der zu entzerrenden Bilder.
Restriktion: ImageWidth > 0

▷ **ImageHeight** (input_control) . . . . . extent.y ⇝ *integer*
Höhe der zu entzerrenden Bilder.
Restriktion: ImageHeight > 0

▷ **MapType** (input_control) . . . . . string ⇝ *string*
Typ der Transformationsdaten.
Defaultwert: 'bilinear'
Werteliste: `MapType` ∈ {'bilinear', 'coord_map_sub_pix'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_arbitrary_distortion_map` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`map_image`

Siehe auch

`create_rectification_grid`, `find_rectification_grid`, `connect_grid_points`, `gen_grid_rectification_map`

Modul

Calibration

gen_grid_rectification_map (Image, ConnectingLines : Map, Meshes : GridSpacing, Rotation, Row, Column, MapType :)

Berechnet die Abbildung vom verzerrten ins entzerrte Bild, aufbauend auf den Punkten in einem regulären Gitter.

`gen_grid_rectification_map` berechnet die Abbildung der (typischerweise mit `saddle_points_sub_pix`) gefundenen Gitterpunkte (`Row`,`Column`) im verzerrten Bild `Image` in die entsprechenden Gitterpunkte im idealen regelmäßigen Punktgitter. Dazu werden aus den Gitterpunkten (`Row`,`Column`) und den (von `connect_grid_points` gefundenen) Verbindungslinien `ConnectingLines` alle Pfade gebildet, die von ihrem Ausgangspunkt über genau vier verschiedene Verbindungslinien zum Ausgangspunkt zurück führen. Sofern die Gitterpunkte (`Row`, `Column`) und Verbindungslinien `ConnectingLines` sinnvolle Eingaben waren, entspricht jede solche 'Masche' genau einem Feld im Entzerrungsgitter. Danach werden die gefundenen Maschen zum gesuchten Punktgitter zusammengesetzt. Das Punktgitter wird, je nachdem, welchen Wert `Rotation` hat, um *0*, *90*, *180* oder *270* Grad gedreht. Dabei ist zu beachten, dass das gefundene Punktgitter nicht notwendigerweise eine korrekte Ausrichtung aufweisen muss. Wird in `Rotation` *'auto'* übergeben, so wird das Punktgitter so gedreht, dass im entzerrten Bild die schwarze der beiden kreisförmigen Marken im Entzerrungsgitter links neben der weißen zu liegen kommt. Abschließend wird die Abbildung `Map` vom verzerrten ins entzerrte Bild durch Interpolation zwischen den Gitterpunkten berechnet. Jede Gitterzelle, für die die Koordinaten (`Row`,`Column`) aller vier Eckpunkte im verzerrten Bild bekannt sind, wird durch Interpolation auf ein Quadrat von `GridSpacing` × `GridSpacing`

Pixeln abgebildet.

Der Parameter `MapType` gibt an, in welchem Typ die Abbildungsvorschrift gespeichert werden soll. Wurde die bilineare Interpolation (*'bilinear'*) gewählt, so besteht `Map` aus einem Bild mit fünf Kanälen. Der erste Kanal enthält für jedes Pixel des Ergebnisbildes die linearen Koordinaten desjenigen Pixels des Eingabebildes, welches links

oberhalb von den transformierten Koordinaten liegt. Die übrigen vier Kanäle enthalten die Gewichte der einzelnen vier Nachbarpixel der transformierten Koordinaten für die bilineare Interpolation in folgender Reihenfolge:

2	3
4	5

Der zweite Kanal z.B. enthält die Gewichte der Pixel, die links oberhalb der transformierten Koordinaten liegen. Wurde *'coord_map_sub_pix'* gewählt, besteht `Map` aus einem Vektorfeld, welches für jedes Pixel des Ergebnisbildes die subpixelgenauen Koordinaten im Eingabebild enthält.

`gen_grid_rectification_map` liefert auch die gefundenen Maschen, als XLD-Konturen, in `Meshes` zurück. Im Gegensatz zu `gen_arbitrary_distortion_map` werden `gen_grid_rectification_map` und die Vorgängerfunktionen verwendet, falls die Koordinaten (`Row`,`Column`) der Gitterpunkte im verzerrten Bild weder bekannt sind noch aus den Bildinhalten gewonnen werden können.

Wenn die erzeugte Abbildung in einem anderen Programm wiederverwendet werden soll, kann man sie mit dem Operator `write_image` als Mehrkanalbild abspeichern (Format: *'tiff'*).

Achtung

Jede Eingabekontur `ConnectingLines` muss, so wie bei `connect_grid_points` beschrieben, das globale Attribut *'bright_dark'* besitzen!

Parameter

- ▷ **`Image`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild.
- ▷ **`ConnectingLines`** (input_object) ..... xld-array ⇝ *object*
  Eingabekonturen.
- ▷ **`Map`** (output_object) ..... multichannel-image ⇝ *object :* int4 / uint2 / vector_field
  Bild mit Transformationsdaten.
- ▷ **`Meshes`** (output_object) ..... xld-array ⇝ *object*
  Ausgabekonturen.
- ▷ **`GridSpacing`** (input_control) ..... integer ⇝ *integer*
  Abstand der Gitterpunkte im entzerrten Bild.
  Restriktion: GridSpacing > 0
- ▷ **`Rotation`** (input_control) ..... string ⇝ *string* / integer
  Drehung, die am Punktgitter vorgenommen wird.
  Defaultwert: 'auto'
  Werteliste: `Rotation` ∈ {'auto', 0, 90, 180, 270}
- ▷ **`Row`** (input_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der Gitterpunkte.
- ▷ **`Column`** (input_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der Gitterpunkte.
  Restriktion: number(Column) == number(Row)
- ▷ **`MapType`** (input_control) ..... string ⇝ *string*
  Typ der Transformationsdaten.
  Defaultwert: 'bilinear'
  Werteliste: `MapType` ∈ {'bilinear', 'coord_map_sub_pix'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_grid_rectification_map` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

connect_grid_points

Nachfolger

map_image

Siehe auch

gen_arbitrary_distortion_map

Modul

Calibration

26.4 Hintergrundschätzer

close_bg_esti (: : BgEstiHandle :)

Löscht Hintergrundschätzer Datensatz.

close_bg_esti gibt den Speicherplatz des Hintergrundschätzer Datensatzes frei.

Parameter

▷ **BgEstiHandle** (input_control) ..... bg_estimation ⇝ *handle*
ID des BgEsti-Datensatzes.

Beispiel

```
* Read image for initialization:
read_image (InitImage, 'xing/init')
* Initialize BgEsti dataset with
* fixed gains and threshold adaption:
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, \
               'on',7.0,10,3.25,15.0,BgEstiHandle)
* Read the next image in sequence:
read_image(Image0, 'xing/xing001')
* Estimate the background:
run_bg_esti(Image0,ForegroundRegion1,BgEstiHandle)
* Display the foreground region:
dev_display (ForegroundRegion1)
* Read the next image in sequence:
read_image(Image1, 'xing/xing002')
* Estimate the background:
run_bg_esti(Image1,ForegroundRegion2,BgEstiHandle)
* Display the foreground region:
dev_display (ForegroundRegion2)
* etc.
* - End of background estimation -
* Close the dataset:
close_bg_esti(BgEstiHandle)
```

Ergebnis

close_bg_esti liefert den Wert TRUE, falls die Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- BgEstiHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

run_bg_esti

Siehe auch

create_bg_esti

Modul

Foundation

create_bg_esti (InitializeImage : : Syspar1, Syspar2, GainMode, Gain1, Gain2, AdaptMode, MinDiff, StatNum, ConfidenceC, TimeC : BgEstiHandle)

Erzeugt Datensatz für Hintergrundschätzer und belegt diesen mit Parametern.

create_bg_esti legt einen neuen Datensatz für die Hintergrundschätzung an und belegt diesen mit den übergebenen Parametern. Zu diesem Datensatz gehört auch das geschätzte Hintergrundbild. Der erzeugte Datensatz wird automatisch der aktuelle Datensatz.

Mit InitializeImage wird eine Anfangsvorhersage für das Hintergrundbild festgelegt. Es ist sinnvoll eine Aufnahme der Szene ohne bewegte Objekte zu übergeben, so dass eine geringe Adaption für den Vordergrund gewählt werden kann. Falls keine Aufnahme der leeren Szene zur Verfügung steht, kann z.B. ein Grauwertbild übergeben werden. Es muss dann allerdings eine schnellere Adaption für den Vordergrund gewählt werden, da alle Bildbereiche zunächst als Vordergrund erkannt werden. Das Initialisierungsbild muss ein Bild vom Typ *'byte'* oder *'real'* sein. Ferner werden nur einkanalige Bilder verarbeitet, daher muss für jeden Kanal ein eigener Datensatz aufgebaut werden. Mit dem InitializeImage wird die Größe und der aktuelle Ausschnitt (Region) für alle Hintergrundschätzungen (run_bg_esti), die mit diesem Datensatz durch geführt werden, festgelegt.

Syspar1 und Syspar2 sind die Parameter der Systemmatrix. Die Systemmatrix beschreibt das System der Grauwertveränderungen entsprechend der Kalmanfilter-Theorie. Im Hintergrundschätzer ist für jedes Pixel ein solches System realisiert.

Mit GainMode wird bestimmt, ob ein festes Kalman-Gain für die Schätzung verwendet werden soll oder ob das Gain in Abhängigkeit der Grauwertdifferenz zwischen Vorhersage und Messwert verwendet werden soll. Wird GainMode = *'fixed'* gewählt, so ist Gain1 der Kalmangain, der bei der Schätzung für Pixel, die Vordergrund detektiert wurden, verwendet wird und Gain2 ist der Kalmangain, der bei der Schätzung für Pixel, die als Hintergrund detektiert wurden, verwendet wird. Gain1 ist sinnvollerweise kleiner als Gain2, denn die vollständige Adaption des Vordergrundes sollte langsamer als die Adaption des Hintergrundes erreicht sein. Feste Gains sind sinnvoll, wenn mit dem Hintergrundschätzer Regionen mit Bewegung gefunden werden sollen und somit Gain1 sehr klein oder *0.0* gewählt wurde. Gain1 und Gain2 sollten hier zwischen *0.0* und *1.0* gewählt werden.

Wird GainMode = *'frame'* gewählt, so wird für die Vordergrundschätzung, sowie für die Hintergrundschätzung eine Tabelle mit Kalmangains für alle 256 möglichen Grauwertdifferenzen bestimmt. Mit Gain1 und Gain2 wird dann die Anzahl der Frames angegeben, die benötigt werden, um eine Abweichung zwischen dem vorhergesagten und dem gemessenen Wert zu adaptieren. Es ist einleuchtend, dass bei gleicher Adaptionszeit (Anzahl der Frames) bei einer großen Differenz ein größeres Kalmangain benötigt wird als bei einer kleinen Differenz. Gain1 ist hier sinnvollerweise größer als Gain2, denn die Adaptionszeit für die Vordergrundadaption sollte länger sein, als die der Hintergrundadaption. Unterschiedliche Gains für verschiedene Grauwertdifferenzen sind sinnvoll, wenn der Hintergrundschätzer zur Aufnahme der 'leeren Szene' verwendet wird, d.h. wenn sich im Aufnahmebereich ständig Objekte bewegen und ein Bild des reinen Hintergrundes erstellt werden muss. Hierzu darf dann die Adaptionszeit für den Vordergrund (Gain1) nicht zu groß gewählt werden. Gain1 und Gain2 sollten hier größer als *1.0* gewählt werden.

Mit AdaptMode wird festgelegt, ob der Schwellenwert, der auf die Grauwertdifferenz zwischen Messwert und Schätzwert angewendet wird, fest ist oder entsprechend der Streuung der Grauwerte der als Hintergrund detektieten Pixel verändert wird.

Wenn AdaptMode = *'off'* gewählt wird, wird mit MinDiff die absolute Schwelle angegeben.Die Parameter StatNum, ConfidenceC und TimeC sind hier ohne Bedeutung.

Wenn AdaptMode = *'on'* gewählt wird, wird mit MinDiff eine Basisschwelle angegeben, auf die, entsprechend der statistischen Auswertung der Grauwertverteilung über die Zeit in jedem Bildpunkt, ein Offset hinzu addiert wird. Mit StatNum gibt man die Anzahl der statistischen Datensätze an, d.h. wie viele der vergangenen Frames zur Berechnung der Varianz verwendet werden sollen (FIR-Filter). Mit ConfidenceC gibt man die Konfidenzkonstante an, die zur Bestimmung des Konfidenzintervalls dient. Mit dem Konfidenzintervall werden die Werte für die Hintergrundstatistik nachgeführt, wenn der Bereich durch ein Objekt im Vordergrund verdeckt ist, d.h. das Pixel als Vordergrund detektiert ist. Entsprechend der Studentsche t-Verteilung ist die Konfidenzkonstante *4.30* (*3.25*, *2.82*, *2.26*) für ein *99,8%* (*99,0%*, *98,0%*, *95,0%*) Konfidenzintervall. Mit TimeC gibt man eine Zeitkonstante für die e-Funktion an, mit der die Schwelle im Fall der Vordergrunddetektion angehoben werden soll. Dies bedeutet, dass die Schwelle in den Bereichen, in denen Bewegung im Vordergrund detektiert wurde, angehoben wird, damit bei anschließendem wieder freigegebenem Blick auf den Hintergrund die Toleranz (Schwelle) für Beleuchtungsänderungen gestiegen ist. Dies ist nötig, da man in der Zeit, in der der Hintergrund verdeckt ist keine Aussage über die Beleuchtungsänderungen im Hintergrund, machen kann und somit das geschätzte Hintergrundbild auch nicht adaptieren kann.

Achtung

Wenn für GainMode = *'frame'* gewählt wurde, kann bei der Wahl von großen Werten für Gain1 oder Gain2 die Laufzeit sehr groß werden, da die Werte für die Gaintabelle mit einem eindimensionalen Optimierer (binäre Suche) bestimmt werden.

Parameter

▷ **InitializeImage** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / real
Initialisierungsbild.

▷ **Syspar1** (input_control) . . . . . real ⇝ *real*
1. Parameter der Systemmatrix.
Defaultwert: 0.7
Wertevorschläge: Syspar1 ∈ {0.65, 0.7, 0.75}
Typischer Wertebereich: 0.05 ≤ Syspar1 ≤ 1.0
Empfohlene Schrittweite: 0.05

▷ **Syspar2** (input_control) . . . . . real ⇝ *real*
2. Parameter der Systemmatrix.
Defaultwert: 0.7
Wertevorschläge: Syspar2 ∈ {0.65, 0.7, 0.75}
Typischer Wertebereich: 0.05 ≤ Syspar2 ≤ 1.0
Empfohlene Schrittweite: 0.05

▷ **GainMode** (input_control) . . . . . string ⇝ *string*
Art der Gains.
Defaultwert: 'fixed'
Werteliste: GainMode ∈ {'fixed', 'frame'}

▷ **Gain1** (input_control) . . . . . real ⇝ *real*
Kalmangain / Adaptionszeit für Vordergrund.
Defaultwert: 0.002
Wertevorschläge: Gain1 ∈ {10.0, 20.0, 50.0, 0.1, 0.05, 0.01, 0.005, 0.001}
Restriktion: 0.0 <= Gain1

▷ **Gain2** (input_control) . . . . . real ⇝ *real*
Kalmangain / Adaptionszeit für Hintergrund.
Defaultwert: 0.02
Wertevorschläge: Gain2 ∈ {2.0, 4.0, 8.0, 0.5, 0.1, 0.05, 0.01}
Restriktion: 0.0 <= Gain2

▷ **AdaptMode** (input_control) . . . . . string ⇝ *string*
Adaption der Schwelle.
Defaultwert: 'on'
Werteliste: AdaptMode ∈ {'on', 'off'}

▷ **MinDiff** (input_control) . . . . . real ⇝ *real*
Schwelle, für Vordergrund / Hintergrund.
Defaultwert: 7.0
Wertevorschläge: MinDiff ∈ {3.0, 5.0, 7.0, 9.0, 11.0}
Empfohlene Schrittweite: 0.2

▷ **StatNum** (input_control) ..... integer ⇝ *integer*
Anzahl statistischer Datensätze.
Defaultwert: 10
Wertevorschläge: StatNum ∈ {5, 10, 20, 30}
Typischer Wertebereich: 1 ≤ StatNum
Empfohlene Schrittweite: 5

▷ **ConfidenceC** (input_control) ..... real ⇝ *real*
Konfidenzkonstante.
Defaultwert: 3.25
Wertevorschläge: ConfidenceC ∈ {4.30, 3.25, 2.82, 2.62}
Empfohlene Schrittweite: 0.01
Restriktion: 0.0 < ConfidenceC

▷ **TimeC** (input_control) ..... real ⇝ *real*
Abklingkonstante.
Defaultwert: 15.0
Wertevorschläge: TimeC ∈ {10.0, 15.0, 20.0}
Empfohlene Schrittweite: 5.0
Restriktion: 0.0 < TimeC

▷ **BgEstiHandle** (output_control) ..... bg_estimation ⇝ *handle*
ID des BgEsti-Datensatzes.

Beispiel

```
* read Init-Image:
read_image (InitImage, 'xing/init')
* initialize 1. BgEsti-Dataset with
* fixed gains and threshold adaption:
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, \
               'on',7.0,10,3.25,15.0,BgEstiHandle1)
* initialize 2. BgEsti-Dataset with
* frame orientated gains and fixed threshold
create_bg_esti(InitImage,0.7,0.7,'frame',30.0,4.0, \
               'off',9.0,10,3.25,15.0,BgEstiHandle2)
```

Ergebnis

create_bg_esti liefert den Wert TRUE, falls die Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

run_bg_esti

Siehe auch

set_bg_esti_params, close_bg_esti

Modul

Foundation

get_bg_esti_params (: : BgEstiHandle : Syspar1, Syspar2, GainMode, Gain1, Gain2, AdaptMode, MinDiff, StatNum, ConfidenceC, TimeC)

Gibt die Parameter des Datensatzes aus.

Mit get_bg_esti_params werden die Parameter des Datensatz ausgegeben. Die zurückgegebenen Parameter sind identisch mit denen für create_bg_esti und set_bg_esti_params (Erklärung siehe dort).

Parameter

- ▷ **BgEstiHandle** (input_control) ..... bg_estimation ⇝ *handle*
  ID des BgEsti-Datensatzes.
- ▷ **Syspar1** (output_control) ..... real ⇝ *real*
  1. Parameter der Systemmatrix.
- ▷ **Syspar2** (output_control) ..... real ⇝ *real*
  2. Parameter der Systemmatrix.
- ▷ **GainMode** (output_control) ..... string ⇝ *string*
  Art der Gains.
- ▷ **Gain1** (output_control) ..... real ⇝ *real*
  Kalmangain / Adaptionszeit für Vordergrund.
- ▷ **Gain2** (output_control) ..... real ⇝ *real*
  Kalmangain / Adaptionszeit für Hintergrund.
- ▷ **AdaptMode** (output_control) ..... string ⇝ *string*
  Adaption der Schwelle.
- ▷ **MinDiff** (output_control) ..... real ⇝ *real*
  Schwelle, für Vordergrund / Hintergrund.
- ▷ **StatNum** (output_control) ..... integer ⇝ *integer*
  Anzahl statistischer Datensätze.
- ▷ **ConfidenceC** (output_control) ..... real ⇝ *real*
  Konfidenzkonstante.
- ▷ **TimeC** (output_control) ..... real ⇝ *real*
  Abklingkonstante.

Beispiel

```
* Read image for initialization:
read_image(InitImage,'xing/init')
* Initialize BgEsti dataset with
* fixed gains and threshold adaption:
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, \
               'on',7.0,10,3.25,15.0,BgEstiHandle)
* Read the next image in sequence:
read_image(Image0,'xing/xing000')
* Estimate the background:
run_bg_esti(Image0,ForegroundRegion1,BgEstiHandle)
* Display the foreground region:
dev_display (ForegroundRegion1)
* Read the next image in sequence:
read_image(Image1,'xing/xing001')
* Estimate the background:
run_bg_esti(Image1,ForegroundRegion2,BgEstiHandle)
* Display the foreground region:
dev_display (ForegroundRegion2)
* etc.
* Change only the gain parameter in dataset:
get_bg_esti_params(BgEstiHandle, Syspar1, Syspar2, \
                   GainMode, Gain1, Gain2, AdaptMode,\
                   MinDiff, StatNum, ConfidenceC, TimeC)
set_bg_esti_params(BgEstiHandle, Syspar1, Syspar2, \
                   GainMode, 0.004, 0.08, AdaptMode,\
                   MinDiff, StatNum, ConfidenceC, TimeC)
* Read the next image in sequence:
read_image(Image3,'xing/xing003')
* Estimate the background:
```

```
run_bg_esti(Image3,ForegroundRegion3,BgEstiHandle)
* Display the foreground region:
dev_display(ForegroundRegion3)
* etc
```

Ergebnis

`get_bg_esti_params` liefert den Wert TRUE, falls die Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_bg_esti`

Nachfolger

`run_bg_esti`

Siehe auch

`set_bg_esti_params`

Modul

Foundation

give_bg_esti (: BackgroundImage : BgEstiHandle :)

Liefert geschätztes Hintergrundbild.

`give_bg_esti` liefert das im aktuellen BgEsti-Datensatz gespeicherte Hintergrundbild. Das Hintergrundbildes ist vom gleichen Typ und gleicher Größe, wie das bei `create_bg_esti` angegebene Initialisierungsbild.

Parameter

▷ **BackgroundImage** (output_object) ..... image ⇝ *object :* byte / real
geschätztes Hintergrundbild aus aktuellem BgEsti-Datensatz.

▷ **BgEstiHandle** (input_control) ..... bg_estimation ⇝ *handle*
ID des BgEsti-Datensatzes.

Beispiel

```
* Read image for initialization:
read_image(InitImage,'xing/init')
* Initialize BgEsti dataset with
* fixed gains and threshold adaption:
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, \
               'on',7.0,10,3.25,15.0,BgEstiHandle)
* Read the next image in sequence:
read_image(Image0,'xing/xing000')
* Estimate the background:
run_bg_esti(Image0,ForegroundRegion1,BgEstiHandle)
* Give the background image from the aktive dataset:
give_bg_esti(BgImage,BgEstiHandle)
* Display the background image:
dev_display(BgImage)
```

Ergebnis

`give_bg_esti` liefert den Wert TRUE, falls die Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

run_bg_esti

Nachfolger

run_bg_esti, create_bg_esti, update_bg_esti

Siehe auch

run_bg_esti, update_bg_esti, create_bg_esti

Modul

Foundation

run_bg_esti (PresentImage : ForegroundRegion : BgEstiHandle :)

Schätzt den Hintergrund und liefert Vordergrund-Region.

run_bg_esti adaptiert mit einem Kalman-Filter das im angegebenen Datensatz liegende Hintergrundbild und liefert eine Region des Vordergrundes (Bereich bewegter Objekte) zurück.

Es wird für jedes Pixel anhand der Daten im aktuellen Datensatz eine Vorhersage des Grauwertes bestimmt und diese dann mit dem Wert im aktuellen Bild (PresentImage) verglichen. Mit dem Schwellenwert (fest oder adaptiv, siehe create_bg_esti) werden die Pixel in Vordergrund oder Hintergrund klassifiziert.

Es werden nur einkanalige Bilder verarbeitet, daher muss der Hintergrund für jeden Kanal einzeln adaptiert werden. Hierzu müssen mehrere Datensätze mit create_bg_esti aufgebaut werden.

Empfehlenswert ist es, den Hintergrundschätzer nur auf der Hälfte oder sogar nur auf einem Viertel der Originalbildgröße anzuwenden. Hierzu muss das Originalbild zunächst mit zoom_image_factor verkleinert werden (dies gilt auch für das Initialisierungsbild bei create_bg_esti). Der Vorteil ist auf der einen Seite die reduzierte Laufzeit und auf der anderen Seite ist das Bild durch die Verkleinerung Tiefpass-gefiltert. Dadurch sind hochfrequente Störungen, wie Kamerarauschen herausgefiltert und stören somit auch nicht bei der Adaption des Hintergrundbildes. Es ist somit möglich die Schwelle (siehe create_bg_esti) geringer zu wählen. Die von run_bg_esti zurückgelieferte ForegroundRegion muss dann natürlich für die Auswertung (Bereiche mit Bewegung) mit zoom_region wieder um den gleichen Faktor vergrößert werden.

Achtung

Das übergebene Bild (PresentImage) muss von gleichem Typ und gleicher Größe sein, wie das im aktuellen Datensatz abgelegte Hintergrundbild (festgelegt mit create_bg_esti).

Parameter

- ▷ **PresentImage** (input_object) ..... singlechannelimage ⇝ *object :* byte / real
  Aktuelles Bild.
- ▷ **ForegroundRegion** (output_object) ..... region ⇝ *object*
  Region der als Vordergrund detektierten Bereiche.
- ▷ **BgEstiHandle** (input_control) ..... bg_estimation ⇝ *handle*
  ID des BgEsti-Datensatzes.

Beispiel

```
* Read image for initialization:
read_image(InitImage,'xing/init')
* Initialize BgEsti dataset with
* fixed gains and threshold adaption:
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, \
               'on',7.0,10,3.25,15.0,BgEstiHandle)
* Read the next image in sequence:
read_image(Image0,'xing/xing000')
* Estimate the background:
```

```
run_bg_esti(Image0,ForegroundRegion1,BgEstiHandle)
* Display the foreground region:
dev_display (ForegroundRegion1)
* Read the next image in sequence:
read_image(Image1,'xing/xing001')
* Estimate the background:
run_bg_esti(Image1,ForegroundRegion2,BgEstiHandle)
* Display the foreground region:
dev_display (ForegroundRegion2)
* etc.
```

Ergebnis

`run_bg_esti` liefert den Wert TRUE, falls die Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_bg_esti`, `update_bg_esti`

Nachfolger

`give_bg_esti`, `update_bg_esti`

Siehe auch

`set_bg_esti_params`, `create_bg_esti`, `update_bg_esti`, `give_bg_esti`

Modul

Foundation

`set_bg_esti_params` `( : : BgEstiHandle, Syspar1, Syspar2, GainMode, Gain1, Gain2, AdaptMode, MinDiff, StatNum, ConfidenceC, TimeC : )`

Ändert im Datensatz die Parameter.

Mit `set_bg_esti_params` werden die Parameter im Datensatz verändert. Die übergebenen Parameter sind identisch mit denen für `create_bg_esti` (Erklärung siehe dort).

Das Bildformat kann jedoch nicht geändert werden, hierfür muss ein neuer Datensatz angelegt werden, bei dem das Initialisierungsbild das gewünschte Format hat.

Soll das Hintergrundbild komplett ausgetauscht werden, so muss `update_bg_esti` verwendet werden. Es muss dann sowohl für das Eingabebild und auch für die Up-Date-Region das aktuelle Bild angegeben werden.

Achtung

Wenn `GainMode` = *'frame'* gewählt wurde, kann bei der Wahl von großen Werten für `Gain1` oder `Gain2` die Laufzeit sehr groß werden, da die Werte für die Gaintabelle mit einem eindimensionalen Optimierer (binäre Suche) bestimmt werden.

Parameter

▷ **`BgEstiHandle`** (input_control) . . . . . bg_estimation ⇝ *handle*
ID des BgEsti-Datensatzes.

▷ **`Syspar1`** (input_control) . . . . . real ⇝ *real*
1. Parameter der Systemmatrix.
Defaultwert: 0.7
Wertevorschläge: `Syspar1` ∈ {0.65, 0.7, 0.75}
Typischer Wertebereich: 0.05 ≤ `Syspar1` ≤ 1.0
Empfohlene Schrittweite: 0.05

▷ **Syspar2** (input_control) . . . . . real ⇝ *real*
2. Parameter der Systemmatrix.
Defaultwert: 0.7
Wertevorschläge: Syspar2 ∈ {0.65, 0.7, 0.75}
Typischer Wertebereich: 0.05 ≤ Syspar2 ≤ 1.0
Empfohlene Schrittweite: 0.05

▷ **GainMode** (input_control) . . . . . string ⇝ *string*
Art der Gains.
Defaultwert: 'fixed'
Werteliste: GainMode ∈ {'fixed', 'frame'}

▷ **Gain1** (input_control) . . . . . real ⇝ *real*
Kalmangain / Adaptionszeit für Vordergrund.
Defaultwert: 0.002
Wertevorschläge: Gain1 ∈ {10.0, 20.0, 50.0, 0.1, 0.05, 0.01, 0.005, 0.001}
Restriktion: 0.0 <= Gain1

▷ **Gain2** (input_control) . . . . . real ⇝ *real*
Kalmangain / Adaptionszeit für Hintergrund.
Defaultwert: 0.02
Wertevorschläge: Gain2 ∈ {2.0, 4.0, 8.0, 0.5, 0.1, 0.05, 0.01}
Restriktion: 0.0 <= Gain2

▷ **AdaptMode** (input_control) . . . . . string ⇝ *string*
Adaption der Schwelle.
Defaultwert: 'on'
Werteliste: AdaptMode ∈ {'on', 'off'}

▷ **MinDiff** (input_control) . . . . . real ⇝ *real*
Schwelle, für Vordergrund / Hintergrund.
Defaultwert: 7.0
Wertevorschläge: MinDiff ∈ {3.0, 5.0, 7.0, 9.0, 11.0}
Empfohlene Schrittweite: 0.2

▷ **StatNum** (input_control) . . . . . integer ⇝ *integer*
Anzahl statistischer Datensätze.
Defaultwert: 10
Wertevorschläge: StatNum ∈ {5, 10, 20, 30}
Typischer Wertebereich: 1 ≤ StatNum
Empfohlene Schrittweite: 5

▷ **ConfidenceC** (input_control) . . . . . real ⇝ *real*
Konfidenzkonstante.
Defaultwert: 3.25
Wertevorschläge: ConfidenceC ∈ {4.30, 3.25, 2.82, 2.62}
Empfohlene Schrittweite: 0.01
Restriktion: 0.0 < ConfidenceC

▷ **TimeC** (input_control) . . . . . real ⇝ *real*
Abklingkonstante.
Defaultwert: 15.0
Wertevorschläge: TimeC ∈ {10.0, 15.0, 20.0}
Empfohlene Schrittweite: 5.0
Restriktion: 0.0 < TimeC

Beispiel

```
* Read image for initialization:
read_image(InitImage,'xing/init')
* Initialize BgEsti dataset with
* fixed gains and threshold adaption:
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, \
               'on',7.0,10,3.25,15.0,BgEstiHandle)
* Read the next image in sequence:
read_image(Image0,'xing/xing000')
```

```
* Estimate the background:
run_bg_esti(Image0,ForegroundRegion1,BgEstiHandle)
* Display the foreground region:
dev_display (ForegroundRegion1)
* Read the next image in sequence:
read_image(Image1,'xing/xing001')
* Estimate the background:
run_bg_esti(Image1,ForegroundRegion2,BgEstiHandle)
* Display the foreground region:
dev_display (ForegroundRegion2)
* etc.
* Change parameter in dataset:
set_bg_esti_params(BgEstiHandle,0.7,0.7,'fixed', \
                   0.004,0.08,'on',9.0,10,3.25,20.0)
* Read the next image in sequence:
read_image(Image2,'xing/xing002')
* Estimate the background:
run_bg_esti(Image2,ForegroundRegion3,BgEstiHandle)
* Display the foreground region:
dev_display(ForegroundRegion3)
* etc.
```

Ergebnis

`set_bg_esti_params` liefert den Wert TRUE, falls die Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_bg_esti`

Nachfolger

`run_bg_esti`

Siehe auch

`update_bg_esti`

Modul

Foundation

update_bg_esti (PresentImage, UpDateRegion : : BgEstiHandle :)

Verändert geschätztes Hintergrundbild.

`update_bg_esti` überschreibt das im BgEsti-Datensatz gespeicherte Hintergrundbild in den durch `UpDateRegion` festgelegten Bereichen durch die Grauwerte aus `PresentImage`. Dies kann zur „harten“ Adaption verwendet werden, d.h. Bildbereiche, in denen sich der Hintergrund stark verändert hat, können hiermit schnell adaptiert werden.

Achtung

Das übergebene Bild (`PresentImage`) muss vom gleichen Typ und gleicher Größe sein, wie das im aktuellen Datensatz abgelegte Hintergrundbild (festgelegt mit `create_bg_esti`).

Parameter

▷ **PresentImage** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / real
Aktuelles Bild.

▷ **UpDateRegion** (input_object) . . . . . region ⇝ *object*
Region, die zu verändernde Bereiche beschreibt.

▷ **BgEstiHandle** (input_control) . . . . . bg_estimation ⇝ *handle*
ID des BgEsti-Datensatzes.

Beispiel

```
* read Init-Image:
read_image(InitImage,'xing/init')
* initialize BgEsti-Dataset with
* fixed gains and threshold adaption
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, \
               'on',7,10,3.25,15.0,BgEstiHandle)
* read the next image in sequence:
read_image(Image0,'xing/xing000')
* estimate the Background:
run_bg_esti(Image0,Region1,BgEstiHandle)
* use the Region and the information of a knowledge base
* to calculate the UpDateRegion
update_bg_esti(Image0,UpdateRegion,BgEstiHandle)
* then read the next image in sequence:
read_image(Image1,'xing/xing001')
* estimate the Background:
run_bg_esti(Image1,Region2,BgEstiHandle)
* etc.
```

Ergebnis

`update_bg_esti` liefert den Wert TRUE, falls die Parameter korrekt sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`run_bg_esti`

Nachfolger

`run_bg_esti`

Siehe auch

`run_bg_esti`, `give_bg_esti`

Modul

Foundation

26.5 Hough

hough_circle_trans (Region : HoughImage : Radius :)

Liefert die Hough-Transformation für Kreise mit dem angegebenen Radius.

`hough_circle_trans` berechnet für die in `Region` übergebenen Regionen die Hough-Transformation für Kreise mit einem bestimmten `Radius`. Dabei werden im Parameterraum (bzw. Hough- oder Akkumulatorraum) die Mittelpunkte aller möglichen Kreise für jeden Punkt im Bildraum akkumuliert. Kreise, die durch viele Punkte der Eingabe-Region gestützt werden, erzeugen so im Ergebnisbild (`HoughImage`) ein Maximum an der den Kreismittelpunkt beschreibenden Stelle. Aus den Koordinaten dieser Maxima ergeben sich durch Subtraktion des `Radius` die Mittelpunkte der Kreise im Bildraum. Werden mehrere Radien übergeben, sind alle Hough-Bilder um den maximalen Radius verschoben.

Parameter

▷ **Region** (input_object) . . . . . region ⇝ *object*
Binäres Kantenbild, in dem Kreise gefunden werden sollen.

▷ **HoughImage** (output_object) . . . . . image(-array) ⇝ *object* : int2
Hough-Transformierte für Kreise mit dem angegebenen Radius.
Parameteranzahl: HoughImage == Radius

▷ **Radius** (input_control) . . . . . integer(-array) ⇝ *integer*
Radius des Kreises, nach dem im Bild gesucht werden soll.
Parameteranzahl: 1 <= Radius <= 500
Defaultwert: 12
Typischer Wertebereich: 3 ≤ Radius (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Ergebnis

`hough_circle_trans` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

hough_circles (RegionIn : RegionOut : Radius, Percent, Mode :)

Mittelpunkt von Kreisen für einem bestimmten Radius.

`hough_circle_trans` findet den Mittelpunkt von Kreisen in Regionen mit Hilfe der Hough-Transformation für Kreise mit einem bestimmten Radius.

Parameter

▷ **RegionIn** (input_object) . . . . . region ⇝ *object*
Binäres Kantenbild, in dem Kreise gefunden werden sollen.

▷ **RegionOut** (output_object) . . . . . region(-array) ⇝ *object*
Mittelpunkte der Kreise, die mit `Percent` Prozent im Kantenbild enthalten sind.
Parameteranzahl: RegionOut == Radius * Percent * Mode

▷ **Radius** (input_control) . . . . . integer(-array) ⇝ *integer*
Radius des Kreises, nach dem im Bild gesucht werden soll.
Parameteranzahl: 1 <= Radius <= 500
Defaultwert: 12
Typischer Wertebereich: 2 ≤ `Radius` ≤ 500 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

▷ **Percent** (input_control) . . . . . integer(-array) ⇝ *integer*
Gibt an, wie viel Prozent des (idealen) Kreises im Kantenbild `RegionIn` enthalten sein müssen (ca.-Angabe).
Parameteranzahl: 1 <= Percent <= 100
Defaultwert: 60
Typischer Wertebereich: 10 ≤ `Percent` ≤ 100 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 5

▷ **Mode** (input_control) ..... integer(-array) ⇝ *integer*
Der Modus definiert die Lage der gesuchten Kreise:
0 - Radius entspricht dem äußerem Rand der gesetzten Pixel.
1 - Radius entspricht den Mittelpunkten der Pixel der Kreislinie.
2 - 0 und 1 zusammen (etwas unschärfer, dadurch aber auch sicherer gegenüber etwas anders gesetzten Kreisen, 50 % höherer Rechenaufwand als 0 und 1).
Parameteranzahl: 1 <= Mode <= 3
Werteliste: `Mode` ∈ {0, 1, 2}

Ergebnis

`hough_circles` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

hough_line_trans (Region : HoughImage : AngleResolution :)

Liefert die Hough-Transformation für Linien in Regionen.

`hough_line_trans` berechnet für die in `Region` übergebenen Regionen die Hough-Transformation für Linien. Dabei werden im Parameterraum (bzw. Hough- oder Akkumulatorraum) Winkel und Länge der Normalenvektoren der Linien eingetragen, d.h. die Parametrisierung der Linien erfolgt in der Hesseschen Normalform.

Das Ergebnis wird in ein neu erzeugtes Int2-Image (`HoughImage`) eingetragen, bei dem die X-Achse dem Winkel des Normalenvektors zur X-Achse (im Ausgangsbild) und die Y-Achse dem Abstand der Linie vom Ursprung entspricht.

Der Winkel läuft von -90 Grad bis 180 Grad, und wird mit einer Auflösung von $1/$`AngleResolution` eingetragen, das bedeutet, dass ein Pixel in X-Richtung $1/$`AngleResolution` Grad entspricht und dass das `HoughImage` $270 *$ `AngleResolution` $+ 1$ Pixel breit ist. Die Höhe von `HoughImage` entspricht dem Abstand der unteren rechten Ecke des umschließenden Rechtecks der Eingabe-Region vom Ursprung.

Die Maxima im Ergebnisbild entsprechen den Parameterwerten der Linien im Ausgangsbild.

Parameter

▷ **Region** (input_object) ..... region ⇝ *object*
Binäres Kantenbild, in dem Linien gefunden werden sollen.

▷ **HoughImage** (output_object) ..... image ⇝ *object :* int2
Hough-Transformierte für Linien.

▷ **AngleResolution** (input_control) ..... integer ⇝ *integer*
Einstellen der Auflösung im Winkelbereich.
Defaultwert: 4
Werteliste: `AngleResolution` ∈ {1, 2, 4, 8}

Ergebnis

`hough_line_trans` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, skeleton

Nachfolger

threshold, local_max

Siehe auch

hough_circle_trans, gen_region_hline

Modul

Foundation

hough_line_trans_dir (ImageDir : HoughImage : DirectionUncertainty, AngleResolution :)

Berechnung der Hough-Transformation für Linien unter Berücksichtigung der lokalen Gradientenrichtung.

hough_line_trans_dir berechnet für die durch den Definitionsbereich von ImageDir übergebene Region die Hough-Transformation für Linien. Dabei werden im Parameterraum (bzw. Hough- oder Akkumulatorraum) Winkel und Länge der Normalenvektoren der Linien eingetragen.

Im Unterschied zu hough_line_trans wird hier jedoch zusätzlich die in ImageDir übergebene Kantenrichtung berücksichtigt, die z.B. mit Hilfe von sobel_dir oder edges_image bestimmt werden kann. Dies führt zu einer effizienteren Berechnung und zu einer Reduktion des Rauschens im Hough-Raum.

Der Parameter DirectionUncertainty beschreibt hierbei, wie stark die Kantenrichtung der einzelnen Punkte innerhalb einer Linie variieren darf. Ist z.B. DirectionUncertainty = *10*, werden einer waagrechten Linie (d.h. Kantenrichtung = 0 Grad) nur Punkte mit einer Kantenrichtung zwischen -10 und +10 Grad zugeordnet. Je größer DirectionUncertainty gewählt wird desto langsamer ist die Berechnung; für DirectionUncertainty = *180* entspricht das Verhalten von hough_line_trans_dir dem von hough_line_trans, d.h. die Kantenrichtung wird ignoriert. DirectionUncertainty sollte mindestens so groß gewählt werden wie die Winkelschrittweite, in der die Kantenrichtungen in ImageDir gespeichert werden; die minimale Schrittweite beträgt 2 Grad (vorgegeben vom Bildtyp 'direction').

Das Ergebnis wird in ein neu erzeugtes UINT2-Image (HoughImage) eingetragen, bei dem die x-Achse (Spalten) dem Winkel des Normalenvektors zur x-Achse des Ausgangsbildes und die y-Achse (Zeilen) dem Abstand der Linie vom Ursprung entspricht.

Der Winkel läuft von -90 Grad bis 180 Grad, und wird mit einer Auflösung von $1/$AngleResolution eingetragen. Das bedeutet, dass ein Pixel in x-Richtung $1/$AngleResolution Grad entspricht und dass das HoughImage $270 *$ AngleResolution $+ 1$ Pixel breit ist. Die Höhe von HoughImage entspricht dem Abstand der unteren rechten Ecke des umschließenden Rechtecks der Eingabe-Region vom Ursprung.

Die Maxima im Ergebnisbild entsprechen den Parameterwerten der Linien im Ausgangsbild.

Parameter

▷ **ImageDir** (input_object) ..... singlechannelimage ⇝ *object :* direction
Kantenrichtungsbild mit auf die Kanten eingeschränktem Definitionsbereich.

▷ **HoughImage** (output_object) ..... image ⇝ *object :* uint2
Hough-Transformation.

▷ **DirectionUncertainty** (input_control) ..... angle.deg ⇝ *integer*
Kantenungenauigkeit (in Grad).
Defaultwert: 2
Typischer Wertebereich: $2 \leq$ DirectionUncertainty ≤ 180
Minimale Schrittweite: 2

▷ **AngleResolution** (input_control) ..... integer ⇝ *integer*
Auflösung im Winkelbereich (in 1/Grad).
Defaultwert: 4
Werteliste: AngleResolution $\in \{1, 2, 4, 8\}$

Ergebnis

`hough_line_trans_dir` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`edges_image`, `sobel_dir`, `threshold`, `hysteresis_threshold`, `nonmax_suppression_dir`, `reduce_domain`

Nachfolger

`binomial_filter`, `gauss_filter`, `threshold`, `local_max`, `plateaus_center`

Siehe auch

`hough_line_trans`, `hough_lines`, `hough_lines_dir`

Modul

Foundation

hough_lines (RegionIn : : AngleResolution, Threshold, AngleGap, DistGap : Angle, Dist)

Findet Linien in Kantenbildern mit Hilfe der Hough-Transformation und liefert sie in Hessescher Normalform zurück.

Mit Hilfe von `hough_lines` können in einer Region linienhafte Strukturen selektiert werden. Dabei müssen die einzelnen Punkte einer Linie nicht zusammenhängen. Das Verfahren basiert auf der Hough-Transformation. Linien werden in Hessescher Normalform, d.h durch Richtung und Länge ihres Normalenvektors repräsentiert.

Mit `AngleResolution` wird die Genauigkeit bei der Bestimmung der Winkel definiert. Sie beträgt $1/$`AngleResolution` Grad. Der Parameter `Threshold` bestimmt, durch wie viel Punkte der Ausgangsregion eine Linienhypothese wenigstens gestützt werden muss, um in die Ausgabe zu gelangen. `AngleGap` und `DistGap` definieren eine Umgebung um die Punkte im Hough-Bild für die Bestimmung der lokalen Maxima. Zurückgeliefert werden die Linien in Hessescher Normalform.

Parameter

▷ **RegionIn** (input_object) . . . . . region ⇝ *object*
Binäres Kantenbild, in dem Linien gefunden werden sollen.

▷ **AngleResolution** (input_control) . . . . . integer ⇝ *integer*
Einstellen der Auflösung im Winkelbereich.
Defaultwert: 4
Werteliste: `AngleResolution` ∈ {1, 2, 4, 8}

▷ **Threshold** (input_control) . . . . . integer ⇝ *integer*
Schwellenwert im Hough-Bild.
Defaultwert: 100
Typischer Wertebereich: 2 ≤ Threshold

▷ **AngleGap** (input_control) . . . . . integer ⇝ *integer*
Minimaler Abstand zwischen zwei Maxima im Hough-Bild (in Angle-Richtung).
Defaultwert: 5
Typischer Wertebereich: 0 ≤ AngleGap

▷ **DistGap** (input_control) . . . . . integer ⇝ *integer*
Minimaler Abstand zwischen zwei Maxima im Hough-Bild (in Distance-Richtung).
Defaultwert: 5
Typischer Wertebereich: 0 ≤ DistGap

▷ **Angle** (output_control) . . . . . hesseline.angle.rad-array ⇝ *real*
Winkel (im Bogenmaß) der Normalen-Vektoren der gefundenen Linien.
Typischer Wertebereich: -1.5707963 ≤ Angle ≤ 3.1415927

▷ **Dist** (output_control) . . . . . hesseline.distance-array ⇝ *real*
Abstände der gefundenen Linien vom Koordinatenursprung.
Parameteranzahl: Dist == Angle
Typischer Wertebereich: 0 ≤ Dist

Ergebnis

hough_lines liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

threshold, skeleton

Nachfolger

select_matching_lines

Siehe auch

hough_line_trans, gen_region_hline, hough_circles

Modul

Foundation

hough_lines_dir (ImageDir : HoughImage, Lines : DirectionUncertainty, AngleResolution, Smoothing, FilterSize, Threshold, AngleGap, DistGap, GenLines : Angle, Dist)

Finden von Linien in Kantenbildern mit Hilfe der Hough-Transformation unter Berücksichtigung der lokalen Gradientenrichtung und Ausgabe in Hessescher Normalform.

Mit Hilfe von hough_lines_dir können in einer Region linienhafte Strukturen selektiert werden. Dabei müssen die einzelnen Punkte einer Linie nicht zusammenhängen. Das Verfahren basiert auf der Hough-Transformation. Die Region muss als Definitionsbereich in ImageDir übergeben werden. Linien werden in Hessescher Normalform, d.h durch Richtung und Länge ihres Normalenvektors repräsentiert.

Im Unterschied zu hough_lines wird hier jedoch zusätzlich die in ImageDir übergebene Kantenrichtung berücksichtigt, die z.B. mit Hilfe von sobel_dir oder edges_image bestimmt werden kann. Dies führt zu einer effizienteren Berechnung und zu einer Reduktion des Rauschens im Hough-Raum.

Der Parameter DirectionUncertainty beschreibt hierbei, wie stark die Kantenrichtung der einzelnen Punkte innerhalb einer Linie variieren darf. Ist z.B. DirectionUncertainty = *10*, werden einer waagrechten Linie (d.h. Kantenrichtung = 0 Grad) nur Punkte mit einer Kantenrichtung zwischen -10 und +10 Grad zugeordnet. Je größer DirectionUncertainty gewählt wird desto langsamer ist die Berechnung; für DirectionUncertainty = *180* entspricht das Verhalten von hough_lines_dir dem von hough_lines, d.h. die Kantenrichtung wird ignoriert. DirectionUncertainty sollte mindestens so groß gewählt werden wie die Winkelschrittweite, in der die Kantenrichtungen in ImageDir gespeichert werden; die minimale Schrittweite beträgt 2 Grad (vorgegeben vom Bildtyp 'direction').

Mit AngleResolution wird die Genauigkeit bei der Bestimmung der Winkel definiert. Sie beträgt 1/AngleResolution Grad. Eine Glättung des Hough-Raumes nach der Transformation erhöht die Stabilität des Verfahrens. Die Art der Glättung kann mit Smoothing festgelegt werden. Der Glättungsgrad kann mit FilterSize analog zu gauss_filter oder mean_image angegeben werden. Der Parameter Threshold bestimmt, durch wie viel Punkte der Ausgangsregion eine Linienhypothese wenigstens gestützt werden muss, um

in die Ausgabe zu gelangen. AngleGap und DistGap definieren eine Umgebung um die Punkte im Hough-Bild für die Bestimmung der lokalen Maxima: AngleGap beschreibt den minimalen Abstand zweier Maxima im Hough-Bild in Richtung des Winkels und DistGap entsprechend Richtung der Distanz. Dadurch werden Maxima, die zwar die vorgegebene Schwelle Threshold überschreiten, jedoch sehr nahe an einem noch höheren Maximum liegen, eliminiert. Dies kann vor allem bei der gleichzeitigen Suche nach langen und kurzen Linien hilfreich sein. Neben dem ungeglätteten Hough-Bild HoughImage werden die Linien in Hessescher Normalform (Angle, Dist) zurückgeliefert. Mit Hilfe des Parameters GenLines kann darüber hinaus festgelegt werden, ob zusätzlich diejenigen Regionen in ImageDir zurückgeliefert werden sollen, die zur Bildung der lokalen Maxima beigetragen haben. Für GenLines = *'true'* werden diese dann in Lines gespeichert.

Parameter

- ▷ **ImageDir** (input_object) ..... singlechannelimage ⇝ *object :* direction
  Kantenrichtungsbild mit auf die Kanten eingeschränktem Definitionsbereich.
- ▷ **HoughImage** (output_object) ..... image ⇝ *object :* uint2
  Hough-Transformation.
- ▷ **Lines** (output_object) ..... region-array ⇝ *object*
  Regionen des Eingabebildes, die zur Bildung der lokalen Maxima beigetragen haben.
- ▷ **DirectionUncertainty** (input_control) ..... angle.deg ⇝ *integer*
  Kantenungenauigkeit (in Grad).
  Defaultwert: 2
  Typischer Wertebereich: 2 ≤ DirectionUncertainty ≤ 180
  Minimale Schrittweite: 2
- ▷ **AngleResolution** (input_control) ..... integer ⇝ *integer*
  Auflösung im Winkelbereich (in 1/Grad).
  Defaultwert: 4
  Werteliste: AngleResolution ∈ {1, 2, 4, 8}
- ▷ **Smoothing** (input_control) ..... string ⇝ *string*
  Glättungsfilter für Hough-Bild.
  Defaultwert: 'mean'
  Werteliste: Smoothing ∈ {'none', 'mean', 'gauss'}
- ▷ **FilterSize** (input_control) ..... integer ⇝ *integer*
  Gewünschte Größe des Glättungsfilters.
  Defaultwert: 5
  Werteliste: FilterSize ∈ {3, 5, 7, 9, 11}
- ▷ **Threshold** (input_control) ..... integer ⇝ *integer*
  Schwellenwert im Hough-Bild.
  Defaultwert: 100
  Typischer Wertebereich: 1 ≤ Threshold
- ▷ **AngleGap** (input_control) ..... integer ⇝ *integer*
  Minimaler Abstand zwischen zwei Maxima im Hough-Bild (in Richtung des Winkels).
  Defaultwert: 5
  Typischer Wertebereich: 0 ≤ AngleGap
- ▷ **DistGap** (input_control) ..... integer ⇝ *integer*
  Minimaler Abstand zwischen zwei Maxima im Hough-Bild (in Richtung der Distance).
  Defaultwert: 5
  Typischer Wertebereich: 0 ≤ DistGap
- ▷ **GenLines** (input_control) ..... string ⇝ *string*
  Generiert Geraden-Regionen, wenn *'true'*.
  Defaultwert: 'true'
  Werteliste: GenLines ∈ {'true', 'false'}
- ▷ **Angle** (output_control) ..... hesseline.angle.rad-array ⇝ *real*
  Winkel (im Bogenmaß) der Normalenvektoren der gefundenen Linien.
  Typischer Wertebereich: -1.5707963 ≤ Angle ≤ 3.1415927
- ▷ **Dist** (output_control) ..... hesseline.distance-array ⇝ *real*
  Abstände der gefundenen Linien vom Koordinatenursprung.
  Parameteranzahl: Dist == Angle
  Typischer Wertebereich: 0 ≤ Dist

Ergebnis

hough_lines_dir liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

edges_image, sobel_dir, threshold, nonmax_suppression_dir, reduce_domain, skeleton

Nachfolger

gen_region_hline, select_matching_lines

Siehe auch

hough_line_trans_dir, hough_line_trans, gen_region_hline, hough_circles

Modul

Foundation

select_matching_lines (RegionIn : RegionLines : AngleIn, DistIn, LineWidth, Thresh : AngleOut, DistOut)

Wählt aus einer Menge von Linien (Hessesche Normalform) die Linien aus, die am besten in eine Eingaberegion passen.

Mit select_matching_lines können aus einer Menge von Linien, die in Hessescher Normalform vorliegen zu einer ebenfalls als Parameter übergebenen Region (RegionIn) die Linien ausgewählt werden, die am besten in die Region hineinpassen. Mit LineWidth kann die Breite der Linien angegeben werden. Die selektierten Linien werden sowohl in Hessescher Normalform als auch als Regionen zurückgeliefert.

Die Auswahl der Linien erfolgt iterativ: In einer Schleife wird als erstes die Linie aus der Menge der Eingabelinien ausgewählt, die die größte Überschneidung mit der Eingaberegion besitzt. Diese Linie wird in die Ausgabemenge übernommen, alle Punkte, die zu dieser Linie gehören, werden im Folgenden nicht mehr für die Bestimmung von Überschneidungen berücksichtigt. Unterschreitet, die maximale Überschneidung zwischen der Region und den Linien einen Schwellenwert (Thresh), wird die Schleife verlassen. Die ausgewählten Linien werden sowohl als Regionen, als auch in Hessescher Normalform zurückgegeben.

Parameter

▷ **RegionIn** (input_object) . . . . . region ⇝ *object*
Region, in denen die Linien gematcht werden sollen.

▷ **RegionLines** (output_object) . . . . . region(-array) ⇝ *object*
Regionen-Array, mit den gematchten Linien.

▷ **AngleIn** (input_control) . . . . . hesseline.angle.rad(-array) ⇝ *real*
Winkel (im Bogenmaß) der Normalen-Vektoren der Eingabe-Linien.
Typischer Wertebereich: -1.5707963 ≤ AngleIn ≤ 3.1415927

▷ **DistIn** (input_control) . . . . . hesseline.distance(-array) ⇝ *real*
Abstände der Eingabe-Linien vom Koordinatenursprung.
Parameteranzahl: DistIn == AngleIn

▷ **LineWidth** (input_control) . . . . . integer ⇝ *integer*
Breite der Linien.
Defaultwert: 7
Typischer Wertebereich: 1 ≤ LineWidth

▷ **Thresh** (input_control) . . . . . integer ⇝ *integer*
Schwellenwert für die Anzahl der Linienpunkte in der Region.
Defaultwert: 100
Typischer Wertebereich: 1 ≤ Thresh

▷ **AngleOut** (output_control) . . . . . hesseline.angle.rad(-array) ⇝ *real*
Winkel (im Bogenmaß) der Normalen-Vektoren der selektierten Linien.
Parameteranzahl: AngleOut <= AngleIn
Typischer Wertebereich: -1.5707963 ≤ `AngleOut` ≤ 3.1415927

▷ **DistOut** (output_control) . . . . . hesseline.distance(-array) ⇝ *real*
Abstände der selektierten Linien vom Koordinatenursprung.
Parameteranzahl: DistOut == AngleOut
Typischer Wertebereich: 0 ≤ DistOut

Ergebnis

`select_matching_lines` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hough_lines`

Modul

Foundation

26.6 Interpolation

clear_scattered_data_interpolator (
: : ScatteredDataInterpolatorHandle :)

Löschen eines Interpolators für verstreute Daten.

`clear_scattered_data_interpolator` löscht den Interpolator für verstreute Daten, der durch `ScatteredDataInterpolatorHandle` angegeben wird, und gibt sämtlichen für den Interpolator benötigten Speicher frei. Der Interpolator kann nach dem Aufruf von `clear_scattered_data_interpolator` nicht mehr verwendet werden. Das Handle `ScatteredDataInterpolatorHandle` ist nach dem Aufruf ungültig.

Parameter

▷ **ScatteredDataInterpolatorHandle** (input_control) . . . . . scattered_data_interpolator(-array) ⇝ *handle*
Handle des Interpolators für verstreute Daten

Ergebnis

Falls `ScatteredDataInterpolatorHandle` korrekt ist, dann liefert `clear_scattered_data_interpolator` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- ScatteredDataInterpolatorHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_scattered_data_interpolator`, `interpolate_scattered_data`

Modul

Foundation

`create_scattered_data_interpolator` (: : `Method`, `Rows`, `Columns`, `Values`, `GenParamName`, `GenParamValue` : `ScatteredDataInterpolatorHandle`)

Erzeugung eines Interpolators zur Interpolation von verstreuten Datenpunkten.

`create_scattered_data_interpolator` erzeugt den Interpolator `ScatteredDataInterpolatorHandle` anhand der verstreuten Datenpunkte `Rows`, `Columns` und der dazugehörigen Messwerte `Values` für die Interpolation der Messwerte an unbekannten Datenpunkten. Mit dem Parameter `Method` wird der Algorithmus festgelegt, der für die Interpolation mit `interpolate_scattered_data` verwendet werden soll. Bisher werden nur die *'thin_plate_splines'* unterstützt. Diese Methode interpoliert die Datenpunkte auf globaler Ebene, was bedeutet, dass alle Punkte für die Interpolation verwendet werden, egal wie weit sie von einander weg liegen. Der Einfluss von entfernten Punkten korreliert mit $r^2 \cdot \log(r)$ wobei r die Entfernung zweier Punkte beschreibt. Im Vergleich zu `interpolate_scattered_data_image`, unterstützt `create_scattered_data_interpolator` auch subpixelgenaue Koordinaten für `Rows` und `Columns`. Nach der Erzeugung des Interpolators wird `interpolate_scattered_data` für die Interpolation der unbekannten Messwerte aufgerufen. Durch die Aufteilung der Erzeugung (`create_scattered_data_interpolator`) und der Auswertung (`interpolate_scattered_data`) des Interpolators, wird die Interpolation an unterschiedlichen Datenpunkten in aufeinanderfolgenden Messungen effizienter, da der Interpolator nur einmal erzeugt werden muss.

Die folgenden Parameter können mit `GenParamName` und `GenParamValue` gesetzt werden:

'alpha': Der Parameter *'alpha'* ist ein Glättungsfaktor. Für *'alpha'* $= 0$ werden alle übergebenen Datenpunkte `Rows`, `Columns` mit `Values` exakt interpoliert. Für sehr große Werte von *'alpha'* glättet die Interpolation die Punkte so, dass alle interpolierten Punkte auf einer gemeinsamen Ebene liegen. **Defaultwert** ist *0*. Beschränkungen: *'alpha'* ≥ 0

Parameter

▷ **`Method`** (input_control) . . . . . string ⇝ *string*
Methode für die Interpolation.
Defaultwert: 'thin_plate_splines'
Wertevorschläge: `Method` ∈ {'thin_plate_splines'}

▷ **`Rows`** (input_control) . . . . . point.y-array ⇝ *real* / integer
Zeilenkoordinaten der Stützstellen.

▷ **`Columns`** (input_control) . . . . . point.x-array ⇝ *real* / integer
Spaltenkoordinaten der Stützstellen.

▷ **`Values`** (input_control) . . . . . number-array ⇝ *real* / integer
Messwerte an den Stützstellen.

▷ **`GenParamName`** (input_control) . . . . . attribute.name-array ⇝ *string*
Namen der generischen Parameter, die angepasst werden können.
Defaultwert: []
Wertevorschläge: `GenParamName` ∈ {'alpha'}

▷ **`GenParamValue`** (input_control) . . . . . attribute.value-array ⇝ *real* / string / integer
Werte der generischen Parameter, die angepasst werden können.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {0, 1.0, 10.0, 100.0}

▷ **`ScatteredDataInterpolatorHandle`** (output_control) . . . . . scattered_data_interpolator ⇝ *handle*
Handle des Interpolators für verstreute Daten

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_scattered_data_interpolator` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`interpolate_scattered_data`

Modul

Foundation

interpolate_scattered_data (: : ScatteredDataInterpolatorHandle, Row, Column : ValueInterpolated)

Interpolation von verstreuten Datenpunkten anhand eines Interpolators.

`interpolate_scattered_data` interpoliert die Messwerte der verstreuten Datenpunkte (`Row`,`Column`) anhand des Interpolators `ScatteredDataInterpolatorHandle` und gibt das Ergebnis in `ValueInterpolated` zurück. Im Vergleich zu `interpolate_scattered_data_image`, unterstützt `interpolate_scattered_data` auch subpixelgenaue Koordinaten für `Row` und `Column`. Der Interpolator muss zuvor mit `create_scattered_data_interpolator` erzeugt werden.

Parameter

- ▷ **ScatteredDataInterpolatorHandle** (input_control) ..... scattered_data_interpolator ⇝ *handle*
  Handle des Interpolators für verstreute Daten
- ▷ **Row** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenkoordinaten der zu interpolierenden Datenpunkte.
- ▷ **Column** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenkoordinaten der zu interpolierenden Datenpunkte.
- ▷ **ValueInterpolated** (output_control) ..... number(-array) ⇝ *real* / integer
  Messwerte der interpolierten Punkte.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `interpolate_scattered_data` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_scattered_data_interpolator`

Nachfolger

`clear_scattered_data_interpolator`, `gen_object_model_3d_from_points`

Modul

Foundation

interpolate_scattered_data_image (Image, RegionInterpolate : ImageInterpolated : Method, GenParamName, GenParamValue :)

Interpolation eines Bildes.

interpolate_scattered_data_image interpoliert das Bild Image innerhalb der Region RegionInterpolate und gibt das Ergebnis in ImageInterpolated zurück. Die Differenz zwischen der Domäne des Bildes Image und der Region RegionInterpolate definiert die gültigen Datenpunkte, die für die Interpolation verwendet werden, wohingegen die Region RegionInterpolate die Punkte definiert, an denen die Grauwerte des Bildes Image bestimmt werden sollen. Mit dem Parameter Method wird der Interpolationsalgorithmus festgelegt. Bisher werden nur die *'thin_plate_splines'* unterstützt. Diese Methode interpoliert die Datenpunkte auf globaler Ebene, was bedeutet, dass alle Punkte für die Interpolation verwendet werden, egal wie weit sie von einander weg liegen. Der Einfluss von entfernten Punkten korreliert mit $r^2 \cdot \log(r)$ wobei r die Entfernung zweier Punkte beschreibt. Wird das selbe Bild Image and unterschiedlichen Regionen RegionInterpolate interpoliert, kann der Operator create_scattered_data_interpolator effizienter sein.

Die folgenden Parameter können mit GenParamName und GenParamValue gesetzt werden:

'alpha': Der Parameter *'alpha'* ist ein Glättungsfaktor. Für *'alpha'* $= 0$ werden alle Punkte des Bildes Image exakt interpoliert. Für sehr große Werte von *'alpha'* glättet die Interpolation die Bildpunkte so, dass alle interpolierten Punkte des Ausgabebildes ImageInterpolated auf einer gemeinsamen Ebene liegen. **Defaultwert** ist *0*. Beschränkungen: *'alpha'* ≥ 0

interpolate_scattered_data_image arbeitet am besten mit wenigen Datenpunkten, z.B. weniger als 1000. Zum Rekonstruieren von verlorenen Bilddaten in der Region RegionInterpolate, sollte die Anzahl der Datenpunkte entsprechend angepasst werden, z.B. indem nur die Randpixel der zu rekonstruierenden Regionen verwendet werden (siehe Beispielprogramm unten).

Parameter

- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2 / real
  Bild, das interpoliert werden soll.
- ▷ **RegionInterpolate** (input_object) ..... region ⇝ *object*
  Region auf der interpoliert werden soll.
- ▷ **ImageInterpolated** (output_object) ..... singlechannelimage ⇝ *object :* real
  Interpoliertes Ausgabebild.
- ▷ **Method** (input_control) ..... string ⇝ *string*
  Methode für die Interpolation.
  Defaultwert: 'thin_plate_splines'
  Wertevorschläge: Method ∈ {'thin_plate_splines'}
- ▷ **GenParamName** (input_control) ..... attribute.name-array ⇝ *string*
  Namen der generischen Parameter, die angepasst werden können.
  Defaultwert: []
  Wertevorschläge: GenParamName ∈ {'alpha'}
- ▷ **GenParamValue** (input_control) ..... attribute.value-array ⇝ *string* / integer / real
  Werte der generischen Parameter, die angepasst werden können.
  Defaultwert: []
  Wertevorschläge: GenParamValue ∈ {0, 1.0, 10.0, 100.0}

Beispiel

```
* This example program shows how to use the scattered data interpolator
* to fill holes in an image
gen_image_surface_second_order (ImageData, 'real', 1, 1, 0, 0, 0, 1, \
                                24, 24, 48, 48)
gen_circle (Circle, 12, 12, 6)
difference (ImageData, Circle, Region)
reduce_domain (ImageData, Region, ImageReduced)
dev_clear_window ()
```

```
dev_display (ImageReduced)
stop()
*
* Select only border pixels for the interpolation
dilation_circle (Circle, CircleDilation, 1.5)
intersection (CircleDilation, Region, RegionBorderData)
dev_clear_window ()
dev_display (ImageReduced)
dev_display (RegionBorderData)
stop()
*
* Interpolate pixels
reduce_domain (ImageData, RegionBorderData, ImageReducedBorder)
interpolate_scattered_data_image (ImageReducedBorder, Circle, \
                                  ImageInterpolated, \
                                  'thin_plate_splines', [], [])
paint_gray (ImageInterpolated, ImageData, ImageFilled)
dev_clear_window ()
dev_display (ImageFilled)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `interpolate_scattered_data_image` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`interpolate_scattered_data_points_to_image`

Modul

Foundation

interpolate_scattered_data_points_to_image (
: ImageInterpolated : Method, Rows, Columns, Values, Width, Height, GenParamName, GenParamValue :)

Generierung eines Bildes durch die Interpolation von verstreuten Datenpunkten.

`interpolate_scattered_data_points_to_image` interpoliert die verstreuten Datenpunkte, die durch `Rows`, `Columns` und die dazugehörigen Messwerte `Values` gegeben sind und gibt das Ergebnis in `ImageInterpolated` zurück. Die Größe des Ausgabebildes wird durch dessen Breite `Width` und dessen Höhe `Height` mit der linken oberen Ecke bei (0,0) festgelegt. Um auch an negativen Koordinaten (`Rows`,`Columns`) zu interpolieren, können einfach alle Eingabepunkte entsprechend verschoben werden. Mit dem Parameter `Method` wird der Interpolationsalgorithmus festgelegt. Bisher werden nur die *'thin_plate_splines'* unterstützt. Diese Methode interpoliert die Datenpunkte auf globaler Ebene, was bedeutet, dass alle Punkte für die Interpolation verwendet werden, egal wie weit sie von einander weg liegen. Der Einfluss von entfernten Punkten korreliert mit $r^2 \cdot \log(r)$ wobei r die Entfernung zweier Punkte beschreibt. Im Vergleich zu `interpolate_scattered_data_image`, unterstützt `interpolate_scattered_data_points_to_image` auch subpixelgenaue Koordinaten für `Rows` und `Columns`. Werden die selben Datenpunkte (`Rows`,`Columns`,`Values`) für die Interpolation von verschiedenen Ausgabebildern verwendet, kann der Operator `create_scattered_data_interpolator` effizienter sein.

Die folgenden Parameter können mit `GenParamName` und `GenParamValue` gesetzt werden:

'alpha': Der Parameter *'alpha'* ist ein Glättungsfaktor. Für *'alpha'* $= 0$ werden alle übergebenen Datenpunkte `Rows`, `Columns` mit `Values` exakt interpoliert. Für sehr große Werte von *'alpha'* glättet die Interpolation

die Punkte so, dass alle interpolierten Punkte des Ausgabebildes `ImageInterpolated` auf einer gemeinsamen Ebene liegen. **Defaultwert** ist *0*. Beschränkungen: *'alpha'* $\geq$ *0*

Parameter

- ▷ **`ImageInterpolated`** (output_object) ..... singlechannelimage ⇝ *object :* real
  Interpoliertes Ausgabebild.
- ▷ **`Method`** (input_control) ..... string ⇝ *string*
  Methode für die Interpolation.
  Defaultwert: 'thin_plate_splines'
  Wertevorschläge: `Method` ∈ {'thin_plate_splines'}
- ▷ **`Rows`** (input_control) ..... point.y-array ⇝ *real* / integer
  Zeilenkoordinaten der Stützstellen.
- ▷ **`Columns`** (input_control) ..... point.x-array ⇝ *real* / integer
  Spaltenkoordinaten der Stützstellen.
- ▷ **`Values`** (input_control) ..... number-array ⇝ *real* / integer
  Messwerte an den Stützstellen.
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Breite des Ausgabebildes.
  Defaultwert: 640
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Ausgabebildes.
  Defaultwert: 480
- ▷ **`GenParamName`** (input_control) ..... attribute.name-array ⇝ *string*
  Namen der generischen Parameter, die angepasst werden können.
  Defaultwert: []
  Wertevorschläge: `GenParamName` ∈ {'alpha'}
- ▷ **`GenParamValue`** (input_control) ..... attribute.value-array ⇝ *real* / string / integer
  Werte der generischen Parameter, die angepasst werden können.
  Defaultwert: []
  Wertevorschläge: `GenParamValue` ∈ {0, 1.0, 10.0, 100.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `interpolate_scattered_data_points_to_image` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`interpolate_scattered_data_image`

Modul

Foundation

26.7 Linien

`line_orientation` (: : `RowBegin`, `ColBegin`, `RowEnd`, `ColEnd` : `Phi`)

Berechnung der Orientierung von Linien.

`line_orientation` liefert die Orientierung ($-pi/2 <$ `Phi` $\leq pi/2$) der übergebenen Linien. Falls mehr als eine Linie behandelt werden soll, können die Zeilen- und Spaltenindizes als Tupel übergeben werden. In diesem Fall ist `Phi` natürlich ebenfalls ein Tupel und enthält die entsprechenden Orientierungen.

Parameter

▷ **RowBegin** (input_control) . . . . . line.begin.y(-array) ⇝ *real* / integer
Zeilenkoordinaten der Anfangspunkte der Eingabelinien.

▷ **ColBegin** (input_control) . . . . . line.begin.x(-array) ⇝ *real* / integer
Spaltenkoordinaten der Anfangspunkte der Eingabelinien.

▷ **RowEnd** (input_control) . . . . . line.end.y(-array) ⇝ *real* / integer
Zeilenkoordinaten der Endpunkte der Eingabelinien.

▷ **ColEnd** (input_control) . . . . . line.end.x(-array) ⇝ *real* / integer
Spaltenkoordinaten der Endpunkte der Eingabelinien.

▷ **Phi** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Orientierung der Eingabelinien.

Ergebnis

`line_orientation` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`sobel_amp`, `edges_image`, `threshold`, `hysteresis_threshold`, `split_skeleton_region`, `split_skeleton_lines`

Nachfolger

`set_line_width`, `disp_line`

Alternativen

`line_position`

Siehe auch

`line_position`, `detect_edge_segments`

Modul

Foundation

line_position (: : RowBegin, ColBegin, RowEnd, ColEnd : RowCenter, ColCenter, Length, Phi)

Berechnung des Schwerpunktes, der Länge und der Orientierung von Linien.

`line_position` liefert den Schwerpunkt (`RowCenter`, `ColCenter`), die (euklidische) Länge (`Length`) und die Orientierung ($-pi/2 <$ `Phi` $\leq pi/2$) der übergebenen Linien. Falls mehr als eine Linie behandelt werden soll, können die Zeilen- und Spaltenindizes als Tupel übergeben werden. In diesem Fall sind die Ausgabeparameter natürlich ebenfalls Tupel.

Die Routine wird beispielsweise auf Modelllinien angewandt, um Suchbereiche für die Kantendetektion (`detect_edge_segments`) zu bestimmen.

Parameter

▷ **RowBegin** (input_control) . . . . . line.begin.y(-array) ⇝ *real* / integer
Zeilenkoordinaten der Anfangspunkte der Eingabelinien.

▷ **ColBegin** (input_control) . . . . . line.begin.x(-array) ⇝ *real* / integer
Spaltenkoordinaten der Anfangspunkte der Eingabelinien.

▷ **RowEnd** (input_control) . . . . . line.end.y(-array) ⇝ *real* / integer
Zeilenkoordinaten der Endpunkte der Eingabelinien.

▷ **ColEnd** (input_control) . . . . . line.end.x(-array) ⇝ *real* / integer
Spaltenkoordinaten der Endpunkte der Eingabelinien.

▷ **RowCenter** (output_control) . . . . . point.y(-array) ⇝ *real*
Zeilenkoordinaten der Schwerpunkte der Eingabelinien.

▷ **ColCenter** (output_control) ..... point.x(-array) ⇝ *real*
Spaltenindizes der Schwerpunkte der Eingabelinien.

▷ **Length** (output_control) ..... real(-array) ⇝ *real*
Euklidische Länge der Eingabelinien.

▷ **Phi** (output_control) ..... angle.rad(-array) ⇝ *real*
Orientierung der Eingabelinien.

Ergebnis

`line_position` liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`sobel_amp`, `edges_image`, `threshold`, `hysteresis_threshold`, `split_skeleton_region`, `split_skeleton_lines`

Nachfolger

`set_line_width`, `disp_line`

Alternativen

`line_orientation`

Siehe auch

`line_orientation`, `detect_edge_segments`

Modul

Foundation

26.8 Mosaicking

```
adjust_mosaic_images ( Images : CorrectedImages : From, To,
    ReferenceImage, HomMatrices2D, EstimationMethod,
    EstimateParameters, OECFModel : )
```

Anwenden einer automatischen Farbkorrektur auf Bilder in Panoramen.

`adjust_mosaic_images` führt eine radiometrische Anpassung von Bildern in Panoramen durch. Die zu korrigierenden Bilder werden im Parameter `Images` gegeben, die korrigierten Bilder werden in `CorrectedImages` zurück geliefert.

Die Parameter `From` und `To` geben die Start- und Zielindizes der einzelnen Bildpaare innerhalb des Panoramas an. Zu jedem dieser Paare muss im Parameter `HomMatrices2D` die entsprechende projektive 3x3-Matrix gegeben werden. Der Parameter `ReferenceImage` legt das Bild fest, dessen Helligkeit und Weißabgleich als Referenz verwendet werden sollen.

Das bedeutet, dass ein Bild über 'perfekte' Belichtung und Weißabgleich verfügt. Alle anderen Bilder werden entsprechend angepasst, so dass deren Belichtung und Weißabgleich dem Referenzbild entspricht. Das Referenzbild wird also nicht verändert, alle anderen Bilder hingegen schon.

`EstimationMethod` bestimmt, ob nur eine ungenaue, aber dafür schnelle Anpassung durchgeführt werden soll, oder eine langsamere, aber dafür wesentlich genauere. Dies wird mit den Werten *'standard'* bzw. *'gold_standard'* festgelegt.

Die Verfahren, die mit *'standard'* gewählt werden können verwenden nur den durchschnittlichen Grauwertunterschied im Überlappungsbereich jedes Bildpaares. Wenn *'gold_standard'* gewählt ist werden dagegen die Grauwertunterschiede zwischen allen korrespondierenden Bildpunkten im Überlappungsbereich berücksichtigt.

Die zu minimierende Fehlerfunktion ergibt sich in allen Fällen als die Quadratsumme der Grauwertunterschiede zwischen den Bildern bzw. Bildpunkten.

Die Verfügbarkeit der einzelnen Methoden ist jedoch abhängig von den gewählten `EstimateParameters`. Dieser Parameter bestimmt, welches Modell zur Bestimmung der radiometrischen Korrektur verwendet werden soll. Es ist prinzipiell immer möglich, die Vignettierung der Bilder durch Angabe von *'vignetting'* bestimmen zu lassen. In diesem Fall muss jedoch für `EstimationMethod` auch *'gold_standard'* gewählt werden. Für die sonstigen radiometrischen Korrekturen bietet das Verfahren drei Optionen:

Die Bestimmung der Vignettierung in Bilder basiert auf dem weithin bekannten

$$\cos^4 Alpha$$

Ansatz. Dabei wird angenommen, dass die Vignettierung im Mittelpunkt des Bildes nicht existiert und mit dem Öffnungswinkel des Objektivs gemäß der obigen Gleichung zunimmt.

1. Die Korrektur mit Hilfe eines additiven Verfahrens. Diese ist lediglich für kleinste Unterschiede zu verwenden. Sie wird durch Angabe des Wertes *'add_gray'* in `EstimateParameters` gewählt. Dieses Modell kann nur alleine mit `EstimationMethod` = *'standard'*, oder gemeinsam mit `EstimateParameters` = *'vignetting'* mit `EstimationMethod` = *'gold_standard'* gewählt werden.

Dieses Verfahren basiert auf der Annahme, dass die Grauwertunterschiede zwischen den Bildern dadurch korrigiert werden können, dass zu den Grauwerten in jedem Bild ein bestimmter Term addiert wird. Dies entspricht im Grunde genau einem Aufruf von `scale_image` für jedes Bild:

scale_image(Image_n,CorrectedImage_n,1.0,Add_n)

mit Add_n dem Korrekturterm für dieses Bild.

2. Die Korrektur mit Hilfe des linearen Verfahrens. In diesem Fall wird angenommen, dass die Eingabebilder mit einer linearen Kamera aufgenommen wurden. Die Unterschiede zwischen den einzelnen Bildern werden als Multiplikationsfaktoren berechnet, mit denen die Bilder korrigiert werden. Dieses Verfahren wird durch Angabe von *'mult_gray'* gewählt. Es kann sowohl mit `EstimationMethod` = *'standard'*, als auch mit `EstimationMethod` = *'gold_standard'* durchgeführt werden. Die Kombination mit `EstimateParameters` = *'vignetting'* ist ebenfalls möglich, jedoch nur mit `EstimationMethod` = *'gold_standard'*.

Dieses Verfahren basiert auf der Annahme, dass die Grauwertunterschiede zwischen den Bildern dadurch korrigiert werden können, dass die Grauwerte in jedem Bild mit einem bestimmten Faktor multipliziert werden. Dies entspricht wieder genau einem Aufruf von `scale_image` für jedes Bild:

scale_image(Image_n,CorrectedImage_n,Mult_n,0)

mit Mult_n dem Korrekturfaktor für dieses Bild.

3. Die Korrektur mit Hilfe des kalibrierten Verfahrens. Bei diesem Modell wird angenommen, dass die Eingabebilder mit einer nichtlinearen Kamera aufgenommen wurden. Zur Modellierung der Transferfunktion wird die im Parameter `OECFModel` gewählte Funktionsklasse verwendet. Zusätzlich werden genau wie im linearen Fall die Unterschiede zwischen den Bildern als Multiplikationsfaktoren berechnet und die Bilder entsprechend korrigiert. Dieses Modell wird durch Angabe von *['mult_gray', 'response']* gewählt und kann nur mit `EstimationMethod` = *'gold_standard'* ausgewählt werden. Eine Kombination mit `EstimateParameters` = *'vignetting'* ist möglich.

Dieses Verfahren ist dem linearen Verfahren ähnlich. In diesem Fall kann die Kamera jedoch eine nichtlineare Antwortfunktion besitzen. Dies bedeutet, dass die Grauwerte der Bilder zuerst wieder linearisiert werden müssen, bevor sie mit einem Skalierungsfaktor multipliziert werden können. Um diesen Schritt vornehmen zu können muss die Antwortfunktion der Kamera bestimmt werden. Da diese normalerweise innerhalb einer Bildsequenz nicht variiert, wird sie als konstant über alle Bilder angenommen.

Prinzipiell könnte jede Art von Funktion als OECF verwendet werden. So ist es zum Beispiel denkbar polynomielle Funktionen wie in `radiometric_self_calibration` zu verwenden. Diese Art von Funktion wäre jedoch in der vorliegenden Anwendung zu komplex und würde nicht zu zufriedenstellenden Ergebnissen führen. Der Grund liegt darin, dass eine solche Funktion über zu viele Parameter verfügt. Stattdessen werden nur einfachere Funktionstypen unterstützt. Derzeit sind lediglich sogenannte Laguerre-Funktionen verfügbar.

Die Antwortfunktion einer OECF der Laguerre-Klasse wird von nur einem Parameter mit dem Namen Phi definiert. Im ersten Schritt der Transformation wird also das Grauwertspektrum (für 8bit Bilder also die Werte zwischen 0 und 255) in das Intervall [0:1] abgebildet. Die resultierenden Fließkommazahlen werden dann mittels der OECF linearisiert und die resultierenden Werte werden wieder in das Ursprungsintervall projiziert.

Die inverse Antwortfunktion, mit der Grauwerte wieder linearisiert werden können lässt sich für Laguerre-Funktionen wie folgt berechnen:

$$I_l = I_nl + \frac{2}{\pi} \cdot \arctan(\frac{Phi \cdot \sin(\pi \cdot I_nl)}{1 - Phi \cdot \cos(\pi \cdot I_nl)})$$

mit I_l dem linearen Grauwert und I_nl dem nichtlinearen Grauwert.

Der Parameter `OECFModel` wird nur verwendet, wenn eine Korrektur mittels des kalibrierten Verfahrens durchgeführt wird. Andernfalls werden eventuelle Eingaben ignoriert.

Der Parameter `EstimateParameters` kann auch dazu benutzt werden, das Laufzeitverhalten und den Speicherverbrauch des Operators zu beeinflussen. Wird der Parameter auf *'no_cache'* gesetzt, so wird der interne Cache Mechanismus des Operators abgeschaltet. Dies bedeutet aber auch, dass diese Einstellung nur dann eine Wirkung hat, wenn `EstimationMethod` auf *'gold_standard'* gesetzt ist. Andernfalls wird die Einstellung ignoriert. Wird der interne Cache Mechanismus abgeschaltet, so benötigt der Operator deutlich weniger Arbeitsspeicher, da die korrespondierenden Grauwertpaare in jeder Iteration der Minimierung erneut berechnet werden. Aus diesem Grund ist die Abschaltung des internen Cache Mechanismus nur dann angeraten, wenn der Speicherbedarf des Operators den verfügbaren physikalischen Speicher des Rechners übersteigt und das Betriebssystem daher auf die Auslagerungsdatei zurück greift.

Eine weitere Möglichkeit die Laufzeit und den Speicherverbrauch zu beeinflussen bietet sich durch die Verwendung von Subsampling. Wird `EstimateParameters` auf *'subsampling_2'* gesetzt, so werden alle Bilder intern um den Faktor 2 verkleinert. Im Gegensatz zu den unten aufgeführten Vorschlägen sind nicht nur die Faktoren 2 und 4 verfügbar, sondern ein beliebiger Ganzzahlwert kann dadurch angegeben werden, dass er an subsampling_ in `EstimateParameters` angehängt wird. Durch dieses Vorgehen kann die gesamte Datenmenge erheblich reduziert werden, was sich in einer deutlich geringeren Laufzeit des verwendeten Minimierungsalgorithmus bemerkbar macht. Zusätzlich kann ein leichtes Subsampling sogar das Ergebnis der Korrektur verbessern, da dadurch der Einfluss von geringfügigen Abbildungsfehlern zwischen den Bildern verringert wird. Obwohl Subsampling auch die Laufzeit beeinflusst, wenn `EstimationMethod` auf *'standard'* gesetzt ist, wird die Verwendung hauptsächlich für *'gold_standard'* empfohlen.

Hier noch einige allgemeine Anmerkungen zur Verwendung von adjust_mosaic_images:

- Die Bestimmung der Vignettierung kann nur dann zu sinnvollen Ergebnissen führen, wenn eine signifikante Vignettierung in den Bildern sichtbar ist. Andernfalls kann das Ergebnis verfälscht werden.
- Die Bestimmung der Antwortfunktion benötigt deutlich mehr Rechenzeit. Daher ist es nicht empfehlenswert, die Antwortfunktion in zeitkritischen Anwendungen zu bestimmen. Abgesehen davon kann die Antwortfunktion auch nur dann robust bestimmt werden, wenn die Helligkeitsunterschiede zwischen den Bildern groß sind.
- Es ist nicht möglich Sättigung zu korrigieren. Bereiche, die gesättigt sind werden dies auch nach der Korrektur bleiben.
- adjust_mosaic_images kann nur dazu verwendet werden unterschiedliche Helligkeit und Weißabgleich in Bildern zu korrigieren, wenn diese auf unterschiedlichen Belichtung (Zeit, Blende) oder unterschiedliche Beleuchtungsintensität zurück zu führen sind. Der Operator kann nicht dazu verwendet werden Helligkeitsunterschiede auszugleichen, die aus inhomogener Beleuchtung innerhalb der einzelnen Bilder resultieren.

Parameter

▷ **Images** (input_object) . . . . . (multichannel-)image-array ⇝ *object :* byte
Eingabebilder.

▷ **CorrectedImages** (output_object) . . . . . (multichannel-)image-array ⇝ *object :* byte
Ausgabebilder.

▷ **From** (input_control) . . . . . integer-array ⇝ *integer*
Liste der Startbilder.

▷ **To** (input_control) . . . . . integer-array ⇝ *integer*
Liste der Zielbilder.

▷ **ReferenceImage** (input_control) . . . . . integer ⇝ *integer*
Referenzbild.

▷ **HomMatrices2D** (input_control) . . . . . real-array ⇝ *real*
Projektive Matrizen.

▷ **EstimationMethod** (input_control) ..... string ⇝ *string*
Schätzalgorithmus für die Korrektur.
Defaultwert: 'standard'
Werteliste: `EstimationMethod` ∈ {'standard', 'gold_standard'}

▷ **EstimateParameters** (input_control) ..... string(-array) ⇝ *string*
Zu bestimmende Parameter.
Defaultwert: ['mult_gray']
Wertevorschläge: `EstimateParameters` ∈ {'add_gray', 'mult_gray', 'response', 'vignetting', 'subsampling_2', 'subsampling_4', 'no_cache'}

▷ **OECFModel** (input_control) ..... string ⇝ *string*
Modell zur Bestimmung der OECF.
Defaultwert: ['laguerre']
Werteliste: `OECFModel` ∈ {'laguerre'}

Beispiel

```
* For the input data to stationary_camera_self_calibration, please
* refer to the example for stationary_camera_self_calibration.
stationary_camera_self_calibration (4, 640, 480, 1, From, To, \
                                    HomMatrices2D, Rows1, Cols1, \
                                    Rows2, Cols2, NumMatches, \
                                    'gold_standard', \
                                    ['focus','principal_point'], \
                                    'true', CameraMatrix, Kappa, \
                                    RotationMatrices, X, Y, Z, Error)
adjust_mosaic_images(Images,CorrectedImages,From,To,1,HomMatrices2D, \
                     'gold_standard',['mult_gray','response'],'laguerre')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `adjust_mosaic_images` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`stationary_camera_self_calibration`

Nachfolger

`gen_spherical_mosaic`

Literatur

David Hasler, Sabine Süsstrunk: Mapping colour in image stitching applications. Journal of Visual Communication and Image Representation, 15(1):65-90, 2004.

Modul

Foundation

bundle_adjust_mosaic (: : NumImages, ReferenceImage, MappingSource, MappingDest, HomMatrices2D, Rows1, Cols1, Rows2, Cols2, NumCorrespondences, Transformation : MosaicMatrices2D, Rows, Cols, Error)

Bündelausgleichung eines Mosaiks.

`bundle_adjust_mosaic` führt eine Bündelausgleichung eines Mosaiks durch. Dies kann dazu verwendet werden, die Geometrie eines Mosaiks möglichst stabil zu berechnen und somit die Abbildungen des Mosaiks genauer zu berechnen, als dies mit den einzelnen Bildpaaren möglich ist.

Um dies zu erreichen, sollte mit proj_match_points_ransac für jedes überlappende Bildpaar die projektive Abbildung zwischen den zwei beteiligten Bildern bestimmt werden. So sind z.B. für einen 2×2 Block von Bildern in folgender Anordnung

1	2
3	4

unter der Annahme, dass sich alle Bilder gegenseitig überlappen, folgende projektive Abbildungen zu bestimmen: 1↦2, 1↦3, 1↦4, 2↦3, 2↦4 und 3↦4. Die Indizes der Bilder, die die jeweilige Abbildung bestimmen, werden in MappingSource und MappingDest übergeben. Die Indizes der Bilder beginnen ab 1. In obigem Beispiel ist also MappingSource = *[1,1,1,2,2,3]* und MappingDest = *[2,3,4,3,4,4]* zu verwenden. Die Gesamtanzahl von Bildern, aus denen das Mosaik besteht, wird in NumImages übergeben. Sie wird verwendet, um zu überprüfen, ob jedes Bild über eine Kette von Abbildungen erreicht werden kann. Der Index des Referenzbildes wird in ReferenceImage übergeben. Dieses Bild erhält in der Ausgabe als Abbildung die Einheitsmatrix.

Die zu den Bildpaaren gehörigen 3×3 projektiven Transformationsmatrizen werden in HomMatrices2D übergeben. Zusätzlich müssen in Rows1, Cols1, Rows2 und Cols2 die Koordinaten der zugeordneten Punktpaare in den einzelnen Bildpaaren übergeben werden. Sie können aus der Ausgabe von proj_match_points_ransac mit tuple_select oder mit der HDevelop-Funktion subset bestimmt werden. Damit bundle_adjust_mosaic bestimmen kann, zu welcher Abbildung ein Punktpaar gehört, muss in NumCorrespondences für jedes Bildpaar die Anzahl der gefundenen Zuordnungen übergeben werden.

Der Parameter Transformation bestimmt die Klasse der Transformationen, die zur Berechnung der Bündelausgleichung für die Transformationen der Bilder verwendet werden. Dies kann verwendet werden, um die zulässigen Transformationen einzuschränken. Für Transformation = *'projective'* werden projektive Abbildungen verwendet (siehe vector_to_proj_hom_mat2d). Für Transformation = *'affine'* werden affine Abbildungen verwendet (siehe vector_to_hom_mat2d), für Transformation = *'similarity'* Ähnlichkeitsabbildungen (siehe vector_to_similarity) und für Transformation = *'rigid'* starre Abbildungen (siehe vector_to_rigid).

Die berechneten Transformationsmatrizen werden als ein Array von 3×3 projektiven Transformationsmatrizen in MosaicMatrices2D zurückgegeben. Zusätzlich werden in (Rows, Cols) die durch die Bündelausgleichung rekonstruierten Punkte zurückgegeben. Der mittlere Projektionsfehler der rekonstruierten Punkte wird in Error zurückgegeben. Dies kann verwendet werden, um zu überprüfen, ob die Optimierung zu sinnvollen Werten konvergiert ist.

Parameter

- ▷ **NumImages** (input_control) . . . integer ⇝ *integer*
  Anzahl der verschiedenen Bilder, die zur Kalibrierung verwendet werden.
  Restriktion: NumImages >= 2
- ▷ **ReferenceImage** (input_control) . . . integer ⇝ *integer*
  Index des Referenzbildes.
- ▷ **MappingSource** (input_control) . . . integer-array ⇝ *integer*
  Indizes der Ausgangsbilder der Transformationen.
- ▷ **MappingDest** (input_control) . . . integer-array ⇝ *integer*
  Indizes der Zielbilder der Transformationen.
- ▷ **HomMatrices2D** (input_control) . . . hom_mat2d-array ⇝ *real*
  Array von 3×3 projektiven Transformationsmatrizen.
- ▷ **Rows1** (input_control) . . . point.x-array ⇝ *real* / integer
  Zeilenkoordinaten korrespondierender Punkte in den jeweiligen Ausgangsbildern.
- ▷ **Cols1** (input_control) . . . point.y-array ⇝ *real* / integer
  Spaltenkoordinaten korrespondierender Punkte in den jeweiligen Ausgangsbildern.
- ▷ **Rows2** (input_control) . . . point.x-array ⇝ *real* / integer
  Zeilenkoordinaten korrespondierender Punkte in den jeweiligen Zielbildern.
- ▷ **Cols2** (input_control) . . . point.y-array ⇝ *real* / integer
  Spaltenkoordinaten korrespondierender Punkte in den jeweiligen Zielbildern.
- ▷ **NumCorrespondences** (input_control) . . . integer-array ⇝ *integer*
  Anzahl der Punktkorrespondenzen im jeweiligen Bildpaar.

▷ **Transformation** (input_control) . . . . . string ⇝ *string*
Verwendete Transformationsklasse.
Defaultwert: 'projective'
Werteliste: Transformation ∈ {'projective', 'affine', 'similarity', 'rigid'}

▷ **MosaicMatrices2D** (output_control) . . . . . hom_mat2d-array ⇝ *real*
Array von 3×3 projektiven Transformationsmatrizen, die die Lage der Bilder im Mosaik bestimmen.

▷ **Rows** (output_control) . . . . . point.x-array ⇝ *real*
Zeilenkoordinaten der durch die Bündelausgleichung rekonstruierten Punkte.

▷ **Cols** (output_control) . . . . . point.y-array ⇝ *real*
Spaltenkoordinaten der durch die Bündelausgleichung rekonstruierten Punkte.

▷ **Error** (output_control) . . . . . real(-array) ⇝ *real*
Durchschnittlicher Fehler pro rekonstruiertem Punkt.

Beispiel

```
* Assume that Images contains the four images of the mosaic in the
* layout given in the above description.  Then the following example
* computes the bundle-adjusted transformation matrices.
From := [1,1,1,2,2,3]
To := [2,3,4,3,4,4]
HomMatrices2D := []
Rows1 := []
Cols1 := []
Rows2 := []
Cols2 := []
NumMatches := []
for J := 0 to |From|-1 by 1
    select_obj (Images, ImageF, From[J])
    select_obj (Images, ImageT, To[J])
    points_foerstner (ImageF, 1, 2, 3, 100, 0.1, 'gauss', 'true', \
                      RowsF, ColsF, _, _, _, _, _, _, _, _)
    points_foerstner (ImageT, 1, 2, 3, 100, 0.1, 'gauss', 'true', \
                      RowsT, ColsT, _, _, _, _, _, _, _, _)
    proj_match_points_ransac (ImageF, ImageT, RowsF, ColsF, RowsT, ColsT, \
                              'ncc', 10, 0, 0, 480, 640, 0, 0.5, \
                              'gold_standard', 2, 42, HomMat2D, \
                              Points1, Points2)
    HomMatrices2D := [HomMatrices2D,HomMat2D]
    Rows1 := [Rows1,subset(RowsF,Points1)]
    Cols1 := [Cols1,subset(ColsF,Points1)]
    Rows2 := [Rows2,subset(RowsT,Points2)]
    Cols2 := [Cols2,subset(ColsT,Points2)]
    NumMatches := [NumMatches,|Points1|]
endfor
bundle_adjust_mosaic (4, 1, From, To, HomMatrices2D, Rows1, Cols1, \
                      Rows2, Cols2, NumMatches, 'rigid', MosaicMatrices, \
                      Rows, Columns, Error)
gen_bundle_adjusted_mosaic (Images, MosaicImage, HomMatrices2D, \
                            'default', 'false', TransMat2D)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert bundle_adjust_mosaic den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

proj_match_points_ransac, proj_match_points_ransac_guided

Nachfolger

gen_bundle_adjusted_mosaic

Siehe auch

gen_projective_mosaic

Modul

Matching

gen_bundle_adjusted_mosaic (Images : MosaicImage : HomMatrices2D, StackingOrder, TransformDomain : TransMat2D)

Kombinieren mehrerer Bilder zu einem Mosaikbild.

gen_bundle_adjusted_mosaic setzt die Eingabebilder, welche im Objekt Images enthalten sind, zu einem Mosaikbild MosaicImage zusammen. Die Relativpositionen der Bilder zueinander werden dabei durch 3×3 projektive Transformationsmatrizen beschrieben. Das Array HomMatrices2D besteht aus einer Aufeinanderfolge solcher linearisierter Matrizen. Die Transformationsmatrizen können mit bundle_adjust_mosaic berechnet werden.

Der Bildausschnitt von MosaicImage und seine Größe werden so gewählt, dass alle Ausgangsbilder gerade ganz sichtbar sind.

Die Reihenfolge, in der die Bilder dem Mosaik hinzugefügt werden, wird durch StackingOrder bestimmt. Das erste Element dieses Arrays bildet das unterste Bild des Mosaiks, während das letzte ganz oben auf dem Mosaik erscheint. Wird statt einem Array von ganzen Zahlen der Wert *'default'* angegeben, so wird die kanonische Ordnung (Bilder in der Reihenfolge in Images) verwendet.

Mit TransformDomain kann festgelegt werden, ob auch die Definitionsbereiche von Images transformiert werden. Da die Transformation der Definitionsbereiche Zeit kostet, kann dieser Parameter dazu verwendet werden, festzulegen, ob dies erwünscht ist. Wenn TransformDomain auf *'false'* gesetzt wird, wird der Definitionsbereich der Eingabebilder ignoriert. Somit werden die kompletten Bilder transformiert.

Als Ausgabe gibt der Parameter TransMat2D eine projektiven Transformationsmatrix, die die Translation beschreibt, die notwendig war, um alle Bilder vollständig in das Ausgabebild abzubilden.

Parameter

- ▷ **Images** (input_object) . . . . . (multichannel-)image-array ⇝ *object :* byte / uint2 / real
  Eingabebilder.
- ▷ **MosaicImage** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2 / real
  Ergebnisbild.
- ▷ **HomMatrices2D** (input_control) . . . . . hom_mat2d-array ⇝ *real*
  Array von 3×3 projektiven Transformationsmatrizen.
- ▷ **StackingOrder** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Reihenfolge der Bilder im Mosaik.
  Defaultwert: 'default'
  Wertevorschläge: StackingOrder ∈ {'default'}
- ▷ **TransformDomain** (input_control) . . . . . string ⇝ *string*
  Sollen die Regionen der Eingabebilder auch transformiert werden?
  Defaultwert: 'false'
  Werteliste: TransformDomain ∈ {'true', 'false'}
- ▷ **TransMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
  3×3 projektive Transformationsmatrix, die die Translation beschreibt, die notwendig war, um alle Bilder vollständig in das Ausgabebild abzubilden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

bundle_adjust_mosaic

Alternativen

gen_projective_mosaic

Siehe auch

projective_trans_image, projective_trans_image_size, projective_trans_region, projective_trans_contour_xld, projective_trans_point_2d, projective_trans_pixel

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2000.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Matching

gen_cube_map_mosaic (Images : Front, Rear, Left, Right, Top, Bottom : CameraMatrices, RotationMatrices, CubeMapDimension, StackingOrder, Interpolation :)

Erzeugen von 6 Cube Maps eines sphärischen Mosaiks.

gen_cube_map_mosaic erzeugt 6 Cube Maps eines sphärischen Mosaiks Front, Left, Rear, Right, Top und Bottom aus den in Images übergebenen Bildern. Die Lage der Bilder im Raum, aus der die Lage der Bilder in Bezug auf die Kugeloberfläche berechnet wird, kann mit stationary_camera_self_calibration bestimmt werden. Die dort berechneten Kameramatrizen und Rotationsmatrizen können direkt in CameraMatrices und RotationMatrices übergeben werden. Ein sphärisches Mosaik kann nur von Aufnahmen einer stationären Kamera berechnet werden (siehe stationary_camera_self_calibration).

Die Breite und Höhe der Cube Maps wird durch CubeMapDimension festgelegt. Dieser Parameter wird in Pixeln angegeben.

Der Modus, in dem die Bilder dem Mosaik hinzugefügt werden, wird durch StackingOrder bestimmt. Für StackingOrder = *'voronoi'* werden die Punkte im Mosaikbild aus der Voronoi-Zelle des jeweiligen Eingabebildes entnommen. Das heißt, dass die Grauwerte aus den zum Mittelpunkt des jeweiligen Bildes auf der Kugeloberfläche am nächsten gelegenen Punkten bestimmt werden. Dieser Modus hat den Vorteil, dass eventuelle Vignettierungen oder unkorrigierte radiale Verzeichnungen im Mosaik weniger auffällig sind, da sie typischerweise symmetrisch zum Bildmittelpunkt sind. Alternativ kann mit der im folgenden beschriebenen Wahl der Parameter ein Modus gewählt werden, der denselben Effekt hat, wie wenn die Bilder nacheinander in das Mosaikbild gezeichnet würden. Hierbei ist die Reihenfolge des Eintragens der Bilder in das Mosaik wichtig. Deswegen kann in StackingOrder ein Array von ganzen Zahlen angegeben werden. Das erste Element dieses Arrays bildet das unterste Bild des Mosaiks, während das letzte ganz oben auf dem Mosaik erscheint. Wird statt einem Array von ganzen Zahlen der Wert *'default'* angegeben, so wird die kanonische Ordnung (Bilder in der Reihenfolge in Images) verwendet. Wenn weder *'voronoi'* noch *'default'* verwendet wird, muss also in StackingOrder eine Permutation der Zahlen 1,...,n verwendet werden, wobei n die Anzahl der in Images übergebenen Bilder ist. Es ist zu beachten, dass der Modus *'voronoi'* nicht immer angewendet werden kann. So müssen z.B. mindestens zwei Bilder übergeben werden. Weiterhin können die Voronoi-Zellen für sehr spezielle Lagen der Bildmittelpunkte auf der Kugel nicht eindeutig bestimmt werden. Darüber hinaus steht mit dem Modus *'blend'* ein Modus bereit, welcher die Bilder des Mosaiks weich ineinander überblendet. Als Grenzen zwischen den Bildern werden hier die gleichen Voronoi-Zellen verwendet, wie im Modus *'voronoi'*. Dieser Modus liefert besonders angenehm wirkende Bilder, benötigt jedoch erheblich mehr Ressourcen. Sollte der Modus *'voronoi'* oder *'blend'* aus irgendeinem Grund nicht anwendbar sein, wird intern automatisch auf den Modus *'default'* umgeschaltet.

Mit dem Parameter Interpolation wird der gewünschte Interpolationsmodus für die Erstellung der Cube Maps gewählt. Es besteht die Wahlmöglichkeit zwischen bilinearer und bikubischer Interpolation für alle Modi

von `StackingOrder`. *'nearest_neighbor'* ist nur verfügbar wenn für `StackingOrder` *'default'* oder *'voronoi'* gewählt ist.

Parameter

- ▷ **`Images`** (input_object) . . . . . (multichannel-)image-array ⇝ *object :* byte / uint2 / real
  Eingabebilder.
- ▷ **`Front`** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2 / real
  Vordere Cube Map.
- ▷ **`Rear`** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2 / real
  Hinterer Cube Map.
- ▷ **`Left`** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2 / real
  Linke Cube Map.
- ▷ **`Right`** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2 / real
  Rechte Cube Map.
- ▷ **`Top`** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2 / real
  Obere Cube Map.
- ▷ **`Bottom`** (output_object) . . . . . (multichannel-)image ⇝ *object :* byte / uint2 / real
  Untere Cube Map.
- ▷ **`CameraMatrices`** (input_control) . . . . . hom_mat2d-array ⇝ *real*
  (Array von) 3×3 projektiven Kameramatrizen, die die internen Kameraparameter bestimmen.
- ▷ **`RotationMatrices`** (input_control) . . . . . hom_mat2d-array ⇝ *real*
  Array von 3×3 Transformationsmatrizen, die die Rotation der Kamera im jeweiligen Bild bestimmen.
- ▷ **`CubeMapDimension`** (input_control) . . . . . number ⇝ *integer*
  Breite und Höhe der zu berechnenden Cube Maps.
  Defaultwert: 1000
  Restriktion: CubeMapDimension >= 0
- ▷ **`StackingOrder`** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Modus des Eintragens der Bilder in das Mosaik.
  Defaultwert: 'voronoi'
  Wertevorschläge: `StackingOrder` ∈ {'blend', 'voronoi', 'default'}
- ▷ **`Interpolation`** (input_control) . . . . . string ⇝ *string*
  Modus der Interpolation der Bilder.
  Defaultwert: 'bilinear'
  Wertevorschläge: `Interpolation` ∈ {'nearest_neighbor', 'bilinear', 'bicubic'}

Beispiel

```
* For the input data to stationary_camera_self_calibration, please
* refer to the example for stationary_camera_self_calibration.
stationary_camera_self_calibration (4, 640, 480, 1, From, To, \
                                    HomMatrices2D, Rows1, Cols1, \
                                    Rows2, Cols2, NumMatches, \
                                    'gold_standard', \
                                    ['focus','principal_point'], \
                                    'true', CameraMatrix, Kappa, \
                                    RotationMatrices, X, Y, Z, Error)
gen_cube_map_mosaic (Images, Front, Left, Rear, Right, Top, Bottom, \
                     CameraMatrix, RotationMatrices, 1000, 'default', \
                     'bicubic')

* Alternatively, if kappa should be determined, the following calls
* can be made:
stationary_camera_self_calibration (4, 640, 480, 1, From, To, \
                                    HomMatrices2D, Rows1, Cols1, \
                                    Rows2, Cols2, NumMatches, \
                                    'gold_standard', \
                                    ['focus','principal_point','kappa'], \
                                    'true', CameraMatrix, Kappa, \
```

```
                                    RotationMatrices, X, Y, Z, Error)
cam_mat_to_cam_par (CameraMatrix, Kappa, 640, 480, CamParam)
change_radial_distortion_cam_par ('fixed', CamParam, 0, CamParOut)
gen_radial_distortion_map (Map, CamParam, CamParOut, 'bilinear')
map_image (Images, Map, ImagesRect)
gen_cube_map_mosaic (Images, Front, Left, Rear, Right, Top, Bottom, \
                     CameraMatrix, RotationMatrices, 1000, 'default', \
                     'bicubic')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_cube_map_mosaic den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

stationary_camera_self_calibration

Alternativen

gen_spherical_mosaic, gen_projective_mosaic

Literatur

Lourdes Agapito, E. Hayman, I. Reid: „Self-Calibration of Rotating and Zooming Cameras“; International Journal of Computer Vision; vol. 45, no. 2; pp. 107–127; 2001.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Matching

gen_projective_mosaic (Images : MosaicImage : StartImage, MappingSource, MappingDest, HomMatrices2D, StackingOrder, TransformDomain : MosaicMatrices2D)

Kombinieren mehrerer Bilder zu einem Mosaikbild.

gen_projective_mosaic setzt die Eingabebilder, welche im Objekt Images enthalten sind, zu einem Mosaikbild MosaicImage zusammen. Die Relativpositionen der Bilder zueinander werden dabei durch 3×3 projektive Transformationsmatrizen beschrieben. Das Array HomMatrices2D besteht aus einer Aufeinanderfolge solcher linearisierter Matrizen. Die Werte in MappingSource und MappingDest geben an, zwischen welchen Bildern die zugehörige Matrix anzuwenden ist. MappingSource=*4* und MappingDest=*7* bedeutet also, dass die Matrix die Transformation des Inhalts von Bild *4* in die Ebene des Bildes *7* beschreibt. Die Transformationsmatrizen zwischen den jeweiligen durch MappingSource und MappingDest gegebenen Bildpaaren werden typischerweise mit proj_match_points_ransac bestimmt.

Wie bei den Bildobjektzugriffsoperatoren (z.B. select_obj) üblich werden die Bilder ab 1 gezählt, d.h. MappingSource, MappingDest, StartImage und StackingOrder) müssen Werte zwischen 1 und der Anzahl der übergebenen Bilder enthalten.

Der Parameter StartImage gibt an, welches der Bilder die Bildebene des Gesamtbildes darstellen soll, welches Bild also unverändert ins Zielbild übernommen wird. Dies ist im allgemeinen ein Bild, das sich in der Mitte des Mosaiks befindet.

Der Bildausschnitt von MosaicImage und seine Größe werden so gewählt, dass alle Ausgangsbilder gerade ganz sichtbar sind.

Die Reihenfolge, in der die Bilder dem Mosaik hinzugefügt werden, wird durch StackingOrder bestimmt. Das erste Element dieses Arrays bildet das unterste Bild des Mosaiks, während das letzte ganz oben auf dem

Mosaik erscheint. Wird statt einem Array von ganzen Zahlen der Wert *'default'* angegeben, so wird die kanonische Ordnung (Bilder in der Reihenfolge in `Images`) verwendet.

Mit `TransformDomain` kann festgelegt werden, ob auch die Definitionsbereiche von `Images` transformiert werden. Da die Transformation der Definitionsbereiche Zeit kostet, kann dieser Parameter dazu verwendet werden, festzulegen, ob dies erwünscht ist. Wenn `TransformDomain` auf *'false'* gesetzt wird, wird der Definitionsbereich der Eingabebilder ignoriert. Somit werden die kompletten Bilder transformiert.

Als Ausgabe gibt der Parameter `MosaicMatrices2D` eine Menge von 3×3 projektiven Transformationsmatrizen, die für jedes Bild in `Images` die Abbildung des Bildes an seine Position im Mosaik beschreibt.

Parameter

- ▷ **Images** (input_object) ..... (multichannel-)image-array ⇝ *object :* byte / uint2 / real
  Eingabebilder.
- ▷ **MosaicImage** (output_object) ..... (multichannel-)image ⇝ *object :* byte / uint2 / real
  Ergebnisbild.
- ▷ **StartImage** (input_control) ..... integer ⇝ *integer*
  Index des zentralen Eingabebildes.
- ▷ **MappingSource** (input_control) ..... integer-array ⇝ *integer*
  Indizes der Ausgangsbilder der Transformationen.
- ▷ **MappingDest** (input_control) ..... integer-array ⇝ *integer*
  Indizes der Zielbilder der Transformationen.
- ▷ **HomMatrices2D** (input_control) ..... hom_mat2d-array ⇝ *real*
  Array von 3×3 projektiven Transformationsmatrizen.
- ▷ **StackingOrder** (input_control) ..... string(-array) ⇝ *string* / integer
  Reihenfolge der Bilder im Mosaik.
  Defaultwert: 'default'
  Wertevorschläge: `StackingOrder` ∈ {'default'}
- ▷ **TransformDomain** (input_control) ..... string ⇝ *string*
  Sollen die Regionen der Eingabebilder auch transformiert werden?
  Defaultwert: 'false'
  Werteliste: `TransformDomain` ∈ {'true', 'false'}
- ▷ **MosaicMatrices2D** (output_control) ..... hom_mat2d-array ⇝ *real*
  Array von 3×3 projektiven Transformationsmatrizen, die die Lage der Bilder im Mosaik bestimmen.

Beispiel

```
gen_empty_obj (Images)
for J := 1 to 6 by 1
    read_image (Image, 'mosaic/pcb_'+J$'02')
    concat_obj (Images, Image, Images)
endfor
From := [1,2,3,4,5]
To := [2,3,4,5,6]
Num := |From|
ProjMatrices := []
for J := 0 to Num-1 by 1
    F := From[J]
    T := To[J]
    select_obj (Images, ImageF, F)
    select_obj (Images, ImageT, T)
    points_foerstner (ImageF, 1, 2, 3, 200, 0.3, 'gauss', 'false', \
                      RowJunctionsF, ColJunctionsF, CoRRJunctionsF, \
                      CoRCJunctionsF, CoCCJunctionsF, RowAreaF, \
                      ColAreaF, CoRRAreaF, CoRCAreaF, CoCCAreaF)
    points_foerstner (ImageT, 1, 2, 3, 200, 0.3, 'gauss', 'false', \
                      RowJunctionsT, ColJunctionsT, CoRRJunctionsT, \
                      CoRCJunctionsT, CoCCJunctionsT, RowAreaT, \
                      ColAreaT, CoRRAreaT, CoRCAreaT, CoCCAreaT)
    proj_match_points_ransac (ImageF, ImageT, RowJunctionsF, \
```

```
                                ColJunctionsF, RowJunctionsT, \
                                ColJunctionsT, 'ncc', 21, 0, 0, 480, 640, \
                                0, 0.5, 'gold_standard', 1, 4364537, \
                                ProjMatrix, Points1, Points2)
    ProjMatrices := [ProjMatrices,ProjMatrix]
endfor
gen_projective_mosaic (Images, MosaicImage, 2, From, To, ProjMatrices, \
                       'default', 'false', MosaicMatrices2D)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

proj_match_points_ransac, proj_match_points_ransac_guided, vector_to_proj_hom_mat2d, hom_vector_to_proj_hom_mat2d

Siehe auch

projective_trans_image, projective_trans_image_size, projective_trans_region, projective_trans_contour_xld, projective_trans_point_2d, projective_trans_pixel

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2000.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Matching

gen_spherical_mosaic

```
gen_spherical_mosaic ( Images : MosaicImage : CameraMatrices,
    RotationMatrices, LatMin, LatMax, LongMin, LongMax, LatLongStep,
    StackingOrder, Interpolation : )
```

Erzeugen eines sphärischen Mosaikbildes.

gen_spherical_mosaic erzeugt ein sphärisches Mosaik MosaicImage aus den in Images übergebenen Bildern. Die Lage der Bilder im Raum, aus der die Lage der Bilder in Bezug auf die Kugeloberfläche berechnet wird, kann mit stationary_camera_self_calibration bestimmt werden. Die dort berechneten Kameramatrizen und Rotationsmatrizen können direkt in CameraMatrices und RotationMatrices übergeben werden. Ein sphärisches Mosaik kann nur von Aufnahmen einer stationären Kamera berechnet werden (siehe stationary_camera_self_calibration).

Das Mosaik wird in Kugelkoordinaten (Länge und Breite) berechnet. Dabei entspricht die Breite der Zeilenachse und die Länge der Spaltenachse von MosaicImage. Der berechnete Ausschnitt der Kugeloberfläche wird durch LatMin, LatMax, LongMin und LongMax festgelegt. Diese Parameter werden in Grad angegeben und legen einen rechteckigen Ausschnitt der Längen- und Breitenkoordinaten fest. Dabei entspricht eine Breite von *-90* dem Nordpol (also der Blickrichtung senkrecht nach oben) und eine Breite von *90* dem Südpol (Blickrichtung senkrecht nach unten). Die Länge von *0* entspricht der Betrachtungsrichtung nach vorne. Negative Längen entsprechen Blickwinkeln nach links, positive Längen Blickwinkeln nach rechts. Um ein vollständiges Bild der Kugeloberfläche zu erhalten, ist also LatMin = *-90*, LatMax = *90*, LongMin = *-180* und LongMax = *180* zu verwenden. In vielen Fällen wird das Mosaik nicht die gesamte Kugeloberfläche abdecken. In diesem Fall ist es sinnvoll, einen Ausschnitt der Kugeloberfläche mit den obigen Parametern auszuwählen. Dies kann zum einen geschehen, indem das Rechteck explizit angegeben wird. Oftmals ist es aber wünschenswert, das kleinstmögliche Rechteck, das alle Bilder enthält, automatisch bestimmen zu lassen. Dies kann geschehen, indem LatMin < *-90*, LatMax > *90*, LongMin < *-180* und LongMax > *180* spezifiziert wird. Dabei werden nur diejenigen Parameter automatisch bestimmt, die außerhalb des normalen Wertebereichs liegen.

Die Winkelschrittweite pro Pixel in MosaicImage kann mit LatLongStep festgelegt werden. LatLongStep wird auch in Grad angegeben. Hiermit kann die Auflösung des Mosaikbildes bestimmt werden. Falls LatLongStep auf *0* gesetzt wird, wird die Winkelschrittweite automatisch bestimmt, indem versucht wird, die Pixelgröße der Originalbilder so gut wie möglich zu erhalten.

Der Modus, in dem die Bilder dem Mosaik hinzugefügt werden, wird durch StackingOrder bestimmt. Für StackingOrder = *'voronoi'* werden die Punkte im Mosaikbild aus der Voronoi-Zelle des jeweiligen Eingabebildes entnommen. Das heißt, dass die Grauwerte aus den zum Mittelpunkt des jeweiligen Bildes auf der Kugeloberfläche am nächsten gelegenen Punkten bestimmt werden. Dieser Modus hat den Vorteil, dass eventuelle Vignettierungen oder unkorrigierte radiale Verzeichnungen im Mosaik weniger auffällig sind, da sie typischerweise symmetrisch zum Bildmittelpunkt sind. Alternativ kann mit der im Folgenden beschriebenen Wahl der Parameter ein Modus gewählt werden, der denselben Effekt hat, wie wenn die Bilder nacheinander in das Mosaikbild gezeichnet würden. Hierbei ist die Reihenfolge des Eintragens der Bilder in das Mosaik wichtig. Deswegen kann in StackingOrder ein Array von ganzen Zahlen angegeben werden. Das erste Element dieses Arrays bildet das unterste Bild des Mosaiks, während das letzte ganz oben auf dem Mosaik erscheint. Wird statt einem Array von ganzen Zahlen der Wert *'default'* angegeben, so wird die kanonische Ordnung (Bilder in der Reihenfolge in Images) verwendet. Wenn weder *'voronoi'* noch *'default'* verwendet wird, muss in StackingOrder eine Permutation der Zahlen 1,...,n verwendet werden, wobei n die Anzahl der in Images übergebenen Bilder ist. Es ist zu beachten, dass der Modus *'voronoi'* nicht immer angewendet werden kann. So müssen z.B. mindestens zwei Bilder übergeben werden. Weiterhin können die Voronoi-Zellen für sehr spezielle Lagen der Bildmittelpunkte auf der Kugel nicht eindeutig bestimmt werden. Darüber hinaus steht mit dem Modus *'blend'* ein Modus bereit, welcher die Bilder des Mosaiks weich ineinander überblendet. Als Grenzen zwischen den Bildern werden hier die gleichen Voronoi-Zellen verwendet, wie im Modus *'voronoi'*. Dieser Modus liefert besonders angenehm wirkende Bilder, benötigt jedoch erheblich mehr Ressourcen. Sollte der Modus *'voronoi'* oder *'blend'* aus irgendeinem Grund nicht anwendbar sein, wird intern automatisch auf den Modus *'default'* umgeschaltet.

Mit dem Parameter Interpolation wird gewünschte Interpolationsmodus für die Erstellung des Mosaiks gewählt. Es besteht die Wahlmöglichkeit zwischen bilinearer und bikubischer Interpolation für alle Modi von StackingOrder. *'nearest_neighbor'* ist nur verfügbar wenn für StackingOrder *'default'* oder *'voronoi'* gewählt ist.

Parameter

- ▷ **Images** (input_object) ..... (multichannel-)image-array ⇝ *object :* byte / uint2 / real
  Eingabebilder.
- ▷ **MosaicImage** (output_object) ..... (multichannel-)image ⇝ *object :* byte / uint2 / real
  Ergebnisbild.
- ▷ **CameraMatrices** (input_control) ..... hom_mat2d-array ⇝ *real*
  (Array von) 3×3 projektiven Kameramatrizen, die die internen Kameraparameter bestimmen.
- ▷ **RotationMatrices** (input_control) ..... hom_mat2d-array ⇝ *real*
  Array von 3×3 Transformationsmatrizen, die die Rotation der Kamera im jeweiligen Bild bestimmen.
- ▷ **LatMin** (input_control) ..... angle.deg ⇝ *real* / integer
  Minimaler Breitengrad der Punkte im sphärischen Mosaik.
  Defaultwert: -90
  Wertevorschläge: LatMin ∈ {-100, -90, -80, -70, -60, -50, -40, -30, -20, -10}
  Restriktion: LatMin <= 90
- ▷ **LatMax** (input_control) ..... angle.deg ⇝ *real* / integer
  Maximaler Breitengrad der Punkte im sphärischen Mosaik.
  Defaultwert: 90
  Wertevorschläge: LatMax ∈ {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
  Restriktion: LatMax >= -90 && LatMax > LatMin
- ▷ **LongMin** (input_control) ..... angle.deg ⇝ *real* / integer
  Minimaler Längengrad der Punkte im sphärischen Mosaik.
  Defaultwert: -180
  Wertevorschläge: LongMin ∈ {-200, -180, -160, -140, -120, -100, -90, -80, -70, -60, -50, -40, -30, -20, -10}
  Restriktion: LongMin <= 180
- ▷ **LongMax** (input_control) ..... angle.deg ⇝ *real* / integer
  Maximaler Längengrad der Punkte im sphärischen Mosaik.
  Defaultwert: 180
  Wertevorschläge: LongMax ∈ {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 140, 160, 180, 200}
  Restriktion: LongMax >= -90 && LongMax > LongMin

▷ **LatLongStep** (input_control) . . . . . angle.deg ⇝ *real* / integer
Winkelschrittweite der Längen und Breiten.
Defaultwert: 0.1
Wertevorschläge: LatLongStep ∈ {0, 0.02, 0.05, 0.1, 0.2, 0.5, 1}
Restriktion: LatLongStep >= 0

▷ **StackingOrder** (input_control) . . . . . string(-array) ⇝ *string* / integer
Modus des Eintragens der Bilder in das Mosaik.
Defaultwert: 'voronoi'
Wertevorschläge: StackingOrder ∈ {'blend', 'voronoi', 'default'}

▷ **Interpolation** (input_control) . . . . . string ⇝ *string* / integer
Interpolationsmodus für die Berechnung des Mosaiks.
Defaultwert: 'bilinear'
Wertevorschläge: Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic'}

Beispiel

```
* For the input data to stationary_camera_self_calibration, please
* refer to the example for stationary_camera_self_calibration.
stationary_camera_self_calibration (4, 640, 480, 1, From, To, \
                                    HomMatrices2D, Rows1, Cols1, \
                                    Rows2, Cols2, NumMatches, \
                                    'gold_standard', \
                                    ['focus','principal_point'], \
                                    'true', CameraMatrix, Kappa, \
                                    RotationMatrices, X, Y, Z, Error)
gen_spherical_mosaic (Images, MosaicImage, CameraMatrix, \
                      RotationMatrices, -100, 100, -200, 200, 0, \
                      'default','bilinear')

* Alternatively, if kappa should be determined, the following calls
* can be made:
stationary_camera_self_calibration (4, 640, 480, 1, From, To, \
                                    HomMatrices2D, Rows1, Cols1, \
                                    Rows2, Cols2, NumMatches, \
                                    'gold_standard', \
                                    ['focus','principal_point','kappa'], \
                                    'true', CameraMatrix, Kappa, \
                                    RotationMatrices, X, Y, Z, Error)
cam_mat_to_cam_par (CameraMatrix, Kappa, 640, 480, CamParam)
change_radial_distortion_cam_par ('fixed', CamParam, 0, CamParOut)
gen_radial_distortion_map (Map, CamParam, CamParOut, 'bilinear')
map_image (Images, Map, ImagesRect)
gen_spherical_mosaic (ImagesRect, MosaicImage, CameraMatrix, \
                      RotationMatrices, -100, 100, -200, 200, 0, \
                      'default','bilinear')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_spherical_mosaic den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

stationary_camera_self_calibration

Alternativen

gen_cube_map_mosaic, gen_projective_mosaic

Literatur

Lourdes Agapito, E. Hayman, I. Reid: „Self-Calibration of Rotating and Zooming Cameras“; International Journal of Computer Vision; vol. 45, no. 2; pp. 107–127; 2001.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Matching

proj_match_points_distortion_ransac (Image1, Image2 : : Rows1, Cols1, Rows2, Cols2, GrayMatchMethod, MaskSize, RowMove, ColMove, RowTolerance, ColTolerance, Rotation, MatchThreshold, EstimationMethod, DistanceThreshold, RandSeed : HomMat2D, Kappa, Error, Points1, Points2)

Automatische Bestimmung der projektiven Transformationsmatrix zwischen zwei Bildern und des radialen Verzeichnungskoeffizienten durch Zuordnung von Bildpunkten.

Ausgehend von einer Menge charakteristischer Punkte (Rows1, Cols1) und (Rows2, Cols2) in den beiden Eingabebildern Image1 und Image2, welche dieselbe Größe besitzen müssen, bestimmt proj_match_points_distortion_ransac automatisch die Korrespondenz der Punkte, die homogene projektive Transformationsmatrix HomMat2D und den radialen Verzeichnungskoeffizienten Kappa (κ), welche die folgende Gleichung am besten erfüllen:

$$\begin{pmatrix} r_2 \\ c_2 \\ 1 \end{pmatrix} = \texttt{HomMat2D} \cdot \begin{pmatrix} r_1 \\ c_1 \\ 1 \end{pmatrix}.$$

Hierbei sind (r_1, c_1) und (r_2, c_2) Bildpunkte, die durch Entzerrung der Eingabebildpunkte mit dem Divisionsmodell entstehen (siehe Kalibrierung):

$$r = \frac{\tilde{r}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)} \qquad c = \frac{\tilde{c}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)}$$

Hierbei bezeichnen $(\tilde{r}_1, \tilde{c}_1) = (\texttt{Rows1} - 0.5(h-1), \texttt{Cols1} - 0.5(w-1))$ und $(\tilde{r}_2, \tilde{c}_2) = (\texttt{Rows2} - 0.5(h-1), \texttt{Cols2} - 0.5(w-1))$ die verzerrten Bildpunkte relativ zum Bildmittelpunkt und w und h die Breite und Höhe der Eingabebilder. proj_match_points_distortion_ransac nimmt also an, dass der Hauptpunkt, d.h. das Zentrum der radialen Verzeichnungen, im Bildmittelpunkt liegt.

Das zurückgelieferte Kappa kann dazu verwendet werden, Kameraparameter zu konstruieren, die zur Entzerrung von Bildern oder Bildpunkten verwendet werden können (siehe change_radial_distortion_cam_par, change_radial_distortion_image und change_radial_distortion_points):

$$\texttt{CamPar} = ['area_scan_telecentric_division', 1.0, \texttt{Kappa}, 1.0, 1.0, 0.5(w-1), 0.5(h-1), w, h]$$

Das Matchingverfahren beruht auf charakteristischen Punkten, welche mit Punktoperatoren, wie z.B. points_foerstner oder points_harris, extrahiert wurden. Die Bestimmung der Korrespondenzen erfolgt in 2 Schritten: Zuerst werden die Grauwertkorrelationen der Umgebungen der Eingabepunkte im ersten und zweiten Bild bestimmt und anhand dieser ein initiales Matching zwischen den Punkten ermittelt. Dann wird das RANSAC-Verfahren angewendet, um die projektive Transformationsmatrix und den radialen Verzeichnungskoeffizienten zu finden, welche die Anzahl der korrespondierenden Punktpaare unter Erfüllung der obigen Bedingung maximieren.

Die Größe der Grauwertfenster, die für das Matching verwendet werden, beträgt MaskSize × MaskSize. Es können drei Metriken für die Korrelation gewählt werden. Hat GrayMatchMethod den Wert *'ssd'*, so wird das Quadrat der Grauwertdifferenzen verwendet, *'sad'* entspricht dem Betrag der Grauwertdifferenzen und *'ncc'* ist die normierte Kreuzkorrelation (siehe auch binocular_disparity). Diese Metrik wird über alle Punktpaare

minimiert (*'ssd'*, *'sad'*) bzw. maximiert (*'ncc'*), eine so gefundene Korrespondenz wird aber nur akzeptiert, falls der Wert der Metrik unter dem Wert von `MatchThreshold` (*'ssd'*, *'sad'*) bzw. über demselben (*'ncc'*) liegt.

Zur Geschwindigkeitssteigerung kann der Suchbereich für die Match-Kandidaten auf ein Rechteck, das durch Größe und Verschiebung spezifiziert wird, eingeschränkt werden. Nur Punkte innerhalb eines $2 \cdot$ `RowTolerance` $\times 2 \cdot$ `ColTolerance` Punkte großen Fensters werden betrachtet. Die Verschiebung des Mittelpunkts dieses Fensters im zweiten Bild gegenüber der Position des aktuellen Punktes im ersten Bild wird durch die Parameter `RowMove` und `ColMove` bestimmt.

Enthält die gesuchte Transformation eine Drehung, ist also das eine gegenüber dem anderen Bild gedreht, kann im Parameter `Rotation` eine Schätzung des Drehwinkels bzw. ein Winkelintervall im Bogenmaß übergeben werden. Eine gute Schätzung des Winkels erhöht die Qualität des Grauwertwertmatchings. Falls sich die tatsächliche Rotation zu stark von der angegebenen Schätzung unterscheidet, schlägt das Matching typischerweise fehl. In diesem Fall sollte ein Winkelintervall angegeben werden. Je größer das angegebene Winkelintervall, desto langsamer läuft der Operator, denn für alle relevanten (automatisch bestimmten) Winkel innerhalb des Intervalls wird das Verfahren komplett durchlaufen.

Ist das initiale Punktmatching bestimmt, wird es anschließend durch einen randomisierten Auswahlalgorithmus (RANSAC) zur Bestimmung der projektiven Transformationsmatrix `HomMat2D` und des radialen Verzeichnungskoeffizienten `Kappa` benutzt. Dabei wird versucht, diese Parameter so zu wählen, dass sie bezüglich der Schranke `DistanceThreshold` zu möglichst vielen Punktpaaren konsistent ist.

Der Parameter `EstimationMethod` gibt an, welches Verfahren zur Berechnung der projektiven Transformationsmatrix angewendet wird. Falls `EstimationMethod` auf *'linear'* gesetzt wird, wird ein lineares Verfahren verwendet, das einen algebraischen Fehler basierend auf den obigen Gleichungen minimiert. Dieses Verfahren ist sehr schnell und liefert genaue Ergebnisse für geringes bis mittleres Rauschen der Punktkoordinaten und für die meisten Verzeichnungen (außer kleinen Verzeichnungen). Für `EstimationMethod` = *'gold_standard'* wird eine mathematisch optimale, dafür aber langsamere, Optimierung durchführt, die den geometrischen Reprojektionsfehler minimiert. Im allgemeinen sollte `EstimationMethod` = *'gold_standard'* gewählt werden.

Der Wert `Error` gibt die Qualität des Matchings an und ist der mittlere symmetrische euklidische Abstand der Punkte zu ihren korrespondierenden transformierten Punkten, gemessen in Pixeln.

Punktpaare, welche die Konsistenzbedingungen erfüllen, werden als Korrespondenzen akzeptiert. `Points1` enthält die Indizes der zugeordneten Eingabepunkte im ersten Bild, `Points2` die Indizes der dazu korrespondierenden Punkte im zweiten Bild.

Der Parameter `RandSeed` kann benutzt werden, um das randomisierte Verhalten des RANSAC-Verfahrens zu kontrollieren und somit reproduzierbare Ergebnisse zu erhalten. Wird `RandSeed` auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit `RandSeed` initialisiert wird. Ist `RandSeed` = *0*, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Parameter

▷ **`Image1`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Eingabebild 1.

▷ **`Image2`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
Eingabebild 2.

▷ **`Rows1`** (input_control) . . . . . point.y-array ⇝ *real* / integer
Eingabepunkte in Bild 1 (Zeilenkoordinate).
Restriktion: length(Rows1) >= 5

▷ **`Cols1`** (input_control) . . . . . point.x-array ⇝ *real* / integer
Eingabepunkte in Bild 1 (Spaltenkoordinate).
Restriktion: length(Cols1) == length(Rows1)

▷ **`Rows2`** (input_control) . . . . . point.y-array ⇝ *real* / integer
Eingabepunkte in Bild 2 (Zeilenkoordinate).
Restriktion: length(Rows2) >= 5

▷ **`Cols2`** (input_control) . . . . . point.x-array ⇝ *real* / integer
Eingabepunkte in Bild 2 (Spaltenkoordinate).
Restriktion: length(Cols2) == length(Rows2)

▷ **`GrayMatchMethod`** (input_control) . . . . . string ⇝ *string*
Metrik für den Vergleich der Grauwerte.
Defaultwert: 'ncc'
Werteliste: `GrayMatchMethod` ∈ {'ncc', 'ssd', 'sad'}

▷ **MaskSize** (input_control) . . . . . integer ⇝ *integer*
Größe der Grauwertmasken.
Defaultwert: 10
Typischer Wertebereich: $3 \leq$ MaskSize ≤ 15
Restriktion: MaskSize >= 1

▷ **RowMove** (input_control) . . . . . integer ⇝ *integer*
Mittlere Zeilenverschiebung.
Defaultwert: 0

▷ **ColMove** (input_control) . . . . . integer ⇝ *integer*
Mittlere Spaltenverschiebung.
Defaultwert: 0

▷ **RowTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Höhe des Suchfensters für das Punktmatching.
Defaultwert: 200
Restriktion: RowTolerance >= 1

▷ **ColTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Breite des Suchfensters für das Punktmatching.
Defaultwert: 200
Restriktion: ColTolerance >= 1

▷ **Rotation** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Schätzung für die Rotation des zweiten Bildes relativ zum ersten Bild.
Defaultwert: 0.0
Wertevorschläge: Rotation ∈ {0.0, 0.1, -0.1, 0.7854, 1.571, 3.142}

▷ **MatchThreshold** (input_control) . . . . . number ⇝ *integer* / real
Schwellenwert für Grauwertkorrespondenzen.
Defaultwert: 0.7
Wertevorschläge: MatchThreshold ∈ {0.9, 0.7, 0.5, 10, 20, 50, 100}

▷ **EstimationMethod** (input_control) . . . . . string ⇝ *string*
Algorithmus zur Berechnung der projektiven Transformationsmatrix.
Defaultwert: 'gold_standard'
Werteliste: EstimationMethod ∈ {'linear', 'gold_standard'}

▷ **DistanceThreshold** (input_control) . . . . . number ⇝ *real* / integer
Schwelle für Konsistenzüberprüfung.
Defaultwert: 1
Restriktion: DistanceThreshold > 0

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Startwert für den Zufallszahlengenerator.
Defaultwert: 0

▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
Berechnete homogene projektive Transformationsmatrix.

▷ **Kappa** (output_control) . . . . . real ⇝ *real*
Berechneter Verzeichnungskoeffizient.

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlerer quadratischer Transformationsfehler.

▷ **Points1** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 1.

▷ **Points2** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 2.

Beispiel

```
points_foerstner (Image1, 1, 2, 3, 50, 0.1, 'gauss', 'true', \
                  Rows1, Cols1, _, _, _, _, _, _, _, _)
points_foerstner (Image2, 1, 2, 3, 50, 0.1, 'gauss', 'true', \
                  Rows2, Cols2, _, _, _, _, _, _, _, _)
get_image_size (Image1, Width, Height)
proj_match_points_distortion_ransac (Image1, Image2, Rows1, Cols1, \
```

```
                                        Rows2, Cols2, 'ncc', 10, 0, 0, \
                                        Height, Width, 0, 0.5, \
                                        'gold_standard', 1, 42, \
                                        HomMat2D, Kappa, Error, \
                                        Points1, Points2)
CamParDist := ['area_scan_division',0.0,Kappa,1.0,1.0, \
               0.5*(Width-1),0.5*(Height-1),Width,Height]
change_radial_distortion_cam_par ('fixed', CamParDist, 0, CamPar)
change_radial_distortion_image (Image1, Image1, Image1Rect, \
                                CamParDist, CamPar)
change_radial_distortion_image (Image2, Image2, Image2Rect, \
                                CamParDist, CamPar)
concat_obj (Image1Rect, Image2Rect, ImagesRect)
gen_projective_mosaic (ImagesRect, MosaicImage, 1, 1, 2, HomMat2D, \
                       'default', 'false', MosaicMatrices2D)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

points_foerstner, points_harris

Nachfolger

vector_to_proj_hom_mat2d_distortion, change_radial_distortion_cam_par, change_radial_distortion_image, change_radial_distortion_points, gen_binocular_proj_rectification, projective_trans_image, projective_trans_image_size, projective_trans_region, projective_trans_contour_xld, projective_trans_point_2d, projective_trans_pixel

Alternativen

proj_match_points_distortion_ransac_guided

Siehe auch

proj_match_points_ransac, proj_match_points_ransac_guided, hom_vector_to_proj_hom_mat2d, vector_to_proj_hom_mat2d

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Matching

proj_match_points_distortion_ransac_guided (Image1, Image2 : : Rows1, Cols1, Rows2, Cols2, GrayMatchMethod, MaskSize, HomMat2DGuide, KappaGuide, DistanceTolerance, MatchThreshold, EstimationMethod, DistanceThreshold, RandSeed : HomMat2D, Kappa, Error, Points1, Points2)

Automatische Bestimmung der projektiven Transformationsmatrix zwischen zwei Bildern und des radialen Verzeichnungskoeffizienten durch Zuordnung von Bildpunkten basierend auf bekannten Näherungen der projektiven Transformationsmatrix und des radialen Verzeichnungskoeffizienten.

Ausgehend von einer Menge charakteristischer Punkte (Rows1, Cols1) und (Rows2, Cols2) in den beiden Eingabebildern Image1 und Image2, welche dieselbe Größe besitzen müssen, und bekannten Näherungen HomMat2DGuide und KappaGuide für die Transformation von Image1 zu Image2 bestimmt proj_match_points_distortion_ransac_guided automatisch die Korrespondenz der Punkte, die homogene projektive Transformationsmatrix HomMat2D und den radialen Verzeichnungskoeffizienten Kappa (κ), welche die folgende Gleichung am besten erfüllen:

$$\begin{pmatrix} r_2 \\ c_2 \\ 1 \end{pmatrix} = \texttt{HomMat2D} \cdot \begin{pmatrix} r_1 \\ c_1 \\ 1 \end{pmatrix}.$$

Hierbei sind (r_1, c_1) und (r_2, c_2) Bildpunkte, die durch Entzerrung der Eingabebildpunkte mit dem Divisionsmodell entstehen (siehe Kalibrierung):

$$r = \frac{\tilde{r}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)} \qquad c = \frac{\tilde{c}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)}$$

Hierbei bezeichnen $(\tilde{r}_1, \tilde{c}_1) = (\texttt{Rows1} - 0.5(h-1), \texttt{Cols1} - 0.5(w-1))$ und $(\tilde{r}_2, \tilde{c}_2) = (\texttt{Rows2} - 0.5(h-1), \texttt{Cols2} - 0.5(w-1))$ die verzerrten Bildpunkte relativ zum Bildmittelpunkt und w und h die Breite und Höhe der Eingabebilder. proj_match_points_distortion_ransac_guided nimmt also an, dass der Hauptpunkt, d.h. das Zentrum der radialen Verzeichnungen, im Bildmittelpunkt liegt.

Das zurückgelieferte Kappa kann dazu verwendet werden, Kameraparameter zu konstruieren, die zur Entzerrung von Bildern oder Bildpunkten verwendet werden können (siehe change_radial_distortion_cam_par, change_radial_distortion_image und change_radial_distortion_points):

$$\texttt{CamPar} = ['area_scan_telecentric_division', 1.0, \texttt{Kappa}, 1.0, 1.0, 0.5(w-1), 0.5(h-1), w, h]$$

Die Näherung HomMat2DGuide kann z.B. mit proj_match_points_distortion_ransac auf niedriger aufgelösten Versionen von Image1 zu Image2 bestimmt werden. Siehe dazu das Beispiel unten.

Das Matchingverfahren beruht auf charakteristischen Punkten, welche mit Punktoperatoren, wie z.B. points_foerstner oder points_harris, extrahiert wurden. Die Bestimmung der Korrespondenzen erfolgt in 2 Schritten: Zuerst werden die Grauwertkorrelationen von Umgebungen der Eingabepunkte im ersten und zweiten Bild bestimmt und anhand dieser ein initiales Matching zwischen den Punkten ermittelt. Dann wird das RANSAC-Verfahren angewendet, um die projektive Transformationsmatrix und den radialen Verzeichnungskoeffizienten zu finden, welche die Anzahl der korrespondierenden Punktpaare unter Erfüllung der obigen Bedingung maximieren.

Die Größe der Grauwertfenster, die für das Matching verwendet werden, beträgt MaskSize × MaskSize. Es können drei Metriken für die Korrelation gewählt werden. Hat GrayMatchMethod den Wert *'ssd'*, so wird das Quadrat der Grauwertdifferenzen verwendet, *'sad'* entspricht dem Betrag der Grauwertdifferenzen und *'ncc'* ist die normierte Kreuzkorrelation (siehe auch binocular_disparity). Diese Metrik wird über alle Punktpaare minimiert (*'ssd'*, *'sad'*) bzw. maximiert (*'ncc'*), eine so gefundene Korrespondenz wird aber nur akzeptiert, falls der Wert der Metrik unter dem Wert von MatchThreshold (*'ssd'*, *'sad'*) bzw. über demselben (*'ncc'*) liegt.

Zur Performanzsteigerung wird der Suchbereich für die Match-Kandidaten basierend auf der Näherung der Transformation eingeschränkt, die durch HomMat2DGuide und KappaGuide bestimmt wird. Nur Punkte, die innerhalb eines Abstandes von DistanceTolerance um die Transformation eines Punktes aus Image1 mittelsHomMat2DGuide und KappaGuide in Image2 liegen, werden für das Matching in Betracht gezogen.

Ist das initiale Punktmatching bestimmt, wird es anschließend durch einen randomisierten Auswahlalgorithmus (RANSAC) zur Bestimmung der projektiven Transformationsmatrix HomMat2D und des radialen Verzeichnungskoeffizienten Kappa benutzt. Dabei wird versucht, diese Parameter so zu wählen, dass sie bezüglich der Schranke DistanceThreshold zu möglichst vielen Punktpaaren konsistent sind. DistanceThreshold sollte also kleiner als DistanceTolerance sein.

Der Parameter EstimationMethod gibt an, welches Verfahren zur Berechnung der projektiven Transformationsmatrix angewendet wird. Falls EstimationMethod auf *'linear'* gesetzt wird, wird ein lineares Verfahren verwendet, das einen algebraischen Fehler basierend auf den obigen Gleichungen minimiert. Dieses Verfahren ist sehr schnell und liefert genaue Ergebnisse für geringes bis mittleres Rauschen der Punktkoordinaten und für die meisten Verzeichnungen (außer kleinen Verzeichnungen). Für EstimationMethod = *'gold_standard'* wird

eine mathematisch optimale, dafür aber langsamere, Optimierung durchführt, die den geometrischen Reprojektionsfehler minimiert. Im allgemeinen sollte `EstimationMethod` = *'gold_standard'* gewählt werden.

Der Wert `Error` gibt die Qualität des Matchings an und ist der mittlere symmetrische euklidische Abstand der Punkte zu ihren korrespondierenden transformierten Punkten, gemessen in Pixeln.

Punktpaare, welche die Konsistenzbedingungen erfüllen, werden als Korrespondenzen akzeptiert. `Points1` enthält die Indizes der zugeordneten Eingabepunkte im ersten Bild, `Points2` die Indizes der dazu korrespondierenden Punkte im zweiten Bild.

Der Parameter `RandSeed` kann benutzt werden, um das randomisierte Verhalten des RANSAC-Verfahrens zu kontrollieren und somit reproduzierbare Ergebnisse zu erhalten. Wird `RandSeed` auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit `RandSeed` initialisiert wird. Ist `RandSeed` = *0*, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Parameter

- ▷ **`Image1`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 1.
- ▷ **`Image2`** (input_object) . . . . . singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 2.
- ▷ **`Rows1`** (input_control) . . . . . point.y-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Zeilenkoordinate).
  Restriktion: length(Rows1) >= 5
- ▷ **`Cols1`** (input_control) . . . . . point.x-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Spaltenkoordinate).
  Restriktion: length(Cols1) == length(Rows1)
- ▷ **`Rows2`** (input_control) . . . . . point.y-array ⇝ *real* / integer
  Eingabepunkte in Bild 2 (Zeilenkoordinate).
  Restriktion: length(Rows2) >= 5
- ▷ **`Cols2`** (input_control) . . . . . point.x-array ⇝ *real* / integer
  Eingabepunkte in Bild 2 (Spaltenkoordinate).
  Restriktion: length(Cols2) == length(Rows2)
- ▷ **`GrayMatchMethod`** (input_control) . . . . . string ⇝ *string*
  Metrik für den Vergleich der Grauwerte.
  Defaultwert: 'ncc'
  Werteliste: `GrayMatchMethod` ∈ {'ncc', 'ssd', 'sad'}
- ▷ **`MaskSize`** (input_control) . . . . . integer ⇝ *integer*
  Größe der Grauwertmasken.
  Defaultwert: 10
  Typischer Wertebereich: 3 ≤ `MaskSize` ≤ 15
  Restriktion: MaskSize >= 1
- ▷ **`HomMat2DGuide`** (input_control) . . . . . hom_mat2d ⇝ *real*
  Näherung der projektiven Transformationsmatrix zwischen den zwei Bildern.
- ▷ **`KappaGuide`** (input_control) . . . . . real ⇝ *real*
  Näherung des radialen Verzeichnungskoeffizienten in den zwei Bildern.
- ▷ **`DistanceTolerance`** (input_control) . . . . . real ⇝ *real*
  Toleranz des Suchfensters für das Punktmatching.
  Defaultwert: 20.0
  Wertevorschläge: `DistanceTolerance` ∈ {0.2, 0.5, 1.0, 2.0, 3.0, 5.0, 10.0, 20.0, 50.0}
  Restriktion: DistanceTolerance > 0
- ▷ **`MatchThreshold`** (input_control) . . . . . number ⇝ *integer* / real
  Schwellenwert für Grauwertkorrespondenzen.
  Defaultwert: 0.7
  Wertevorschläge: `MatchThreshold` ∈ {0.9, 0.7, 0.5, 10, 20, 50, 100}
- ▷ **`EstimationMethod`** (input_control) . . . . . string ⇝ *string*
  Schätzalgorithmus für die Transformationsmatrix.
  Defaultwert: 'gold_standard'
  Werteliste: `EstimationMethod` ∈ {'linear', 'gold_standard'}

▷ **DistanceThreshold** (input_control) ..... number ⇝ *real* / integer
Schwelle für die Konsistenzüberprüfung.
Defaultwert: 1
Restriktion: DistanceThreshold > 0

▷ **RandSeed** (input_control) ..... integer ⇝ *integer*
Startwert für den Zufallszahlengenerator.
Defaultwert: 0

▷ **HomMat2D** (output_control) ..... hom_mat2d ⇝ *real*
Berechnete homogene projektive Transformationsmatrix.

▷ **Kappa** (output_control) ..... real ⇝ *real*
Berechneter Verzeichnungskoeffizient.

▷ **Error** (output_control) ..... real ⇝ *real*
Mittlerer quadratischer Transformationsfehler.

▷ **Points1** (output_control) ..... integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 1.

▷ **Points2** (output_control) ..... integer-array ⇝ *integer*
Indizes der zugeordneten Eingabepunkte aus Bild 2.

Beispiel

```
Factor := 0.5
zoom_image_factor (Image1, Image1Zoomed, Factor, Factor, 'constant')
zoom_image_factor (Image2, Image2Zoomed, Factor, Factor, 'constant')
points_foerstner (Image1Zoomed, 1, 2, 3, 200, 0.3, 'gauss', 'true', \
                  Rows1, Cols1, _, _, _, _, _, _, _, _)
points_foerstner (Image2Zoomed, 1, 2, 3, 200, 0.3, 'gauss', 'true', \
                  Rows2, Cols2, _, _, _, _, _, _, _, _)
get_image_size (Image1Zoomed, Width, Height)
proj_match_points_distortion_ransac (Image1Zoomed, Image2Zoomed, \
                                     Rows1, Cols1, Rows2, Cols2, \
                                     'ncc', 10, 0, 0, Height, Width, \
                                     0, 0.5, 'gold_standard', 2, 0, \
                                     HomMat2D, Kappa, Error, \
                                     Points1, Points2)
hom_mat2d_scale_local (HomMat2D, Factor, Factor, HomMat2DGuide)
hom_mat2d_scale (HomMat2DGuide, 1.0/Factor, 1.0/Factor, 0, 0, \
                 HomMat2DGuide)
KappaGuide := Kappa*Factor*Factor
points_foerstner (Image1, 1, 2, 3, 200, 0.3, 'gauss', 'true', \
                  Rows1, Cols1, _, _, _, _, _, _, _, _)
points_foerstner (Image2, 1, 2, 3, 200, 0.3, 'gauss', 'true', \
                  Rows2, Cols2, _, _, _, _, _, _, _, _)
proj_match_points_distortion_ransac_guided (Image1, Image2, \
                                            Rows1, Cols1, \
                                            Rows2, Cols2, \
                                            'ncc', 10, \
                                            HomMat2DGuide, \
                                            KappaGuide, 5, 0.5, \
                                            'gold_standard', 2, 0, \
                                            HomMat2D, Kappa, \
                                            Error, Points1, Points2)
get_image_size (Image1, Width, Height)
CamParDist := ['area_scan_division',0.0,Kappa,1.0,1.0, \
               0.5*(Width-1),0.5*(Height-1),Width,Height]
change_radial_distortion_cam_par ('fixed', CamParDist, 0, CamPar)
change_radial_distortion_image (Image1, Image1, Image1Rect, \
                                CamParDist, CamPar)
change_radial_distortion_image (Image2, Image2, Image2Rect, \
                                CamParDist, CamPar)
```

```
concat_obj (Image1Rect, Image2Rect, ImagesRect)
gen_projective_mosaic (ImagesRect, MosaicImage, 1, 1, 2, HomMat2D, \
                       'default', 'false', MosaicMatrices2D)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

points_foerstner, points_harris

Nachfolger

vector_to_proj_hom_mat2d_distortion, change_radial_distortion_cam_par, change_radial_distortion_image, change_radial_distortion_points, gen_binocular_proj_rectification, projective_trans_image, projective_trans_image_size, projective_trans_region, projective_trans_contour_xld, projective_trans_point_2d, projective_trans_pixel

Alternativen

proj_match_points_distortion_ransac

Siehe auch

proj_match_points_ransac, proj_match_points_ransac_guided, hom_vector_to_proj_hom_mat2d, vector_to_proj_hom_mat2d

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Matching

proj_match_points_ransac (Image1, Image2 : : Rows1, Cols1, Rows2, Cols2, GrayMatchMethod, MaskSize, RowMove, ColMove, RowTolerance, ColTolerance, Rotation, MatchThreshold, EstimationMethod, DistanceThreshold, RandSeed : HomMat2D, Points1, Points2)

Bestimmt eine projektive Transformationsmatrix automatisch durch Zuordnung von Punkten zwischen zwei Bildern.

Ausgehend von einer Menge charakteristischer Punkte (Cols1,Rows1) und (Cols2,Rows2) in den beiden Eingabebildern Image1 und Image2 bestimmt proj_match_points_ransac automatisch die Korrespondenz der Punkte und die homogene projektive Transformationsmatrix HomMat2D, die die korrespondierenden Punkte in den beiden Bildern am besten ineinander überführt. Die charakteristischen Punkte können z.B. mit points_foerstner oder points_harris extrahiert werden.

Die Bestimmung der Transformation erfolgt in 2 Schritten: Zuerst werden die Grauwertkorrelationen von Umgebungen der Eingabepunkte im ersten und zweiten Bild bestimmt und anhand dieser ein initiales Matching zwischen den Punkten ermittelt. Die Größe dieser Grauwertfenster beträgt MaskSize $\times$ MaskSize . Es können drei Metriken für die Korrelation gewählt werden. Hat GrayMatchMethod den Wert *'ssd'*, so wird das Quadrat der Grauwertdifferenzen verwendet, *'sad'* entspricht dem Betrag der Grauwertdifferenzen und *'ncc'* ist die normierte Kreuzkorrelation (siehe auch binocular_disparity). Diese Metrik wird über alle Punktepaare minimiert (*'ssd'*, *'sad'*) bzw. maximiert (*'ncc'*), ein so gefundenes Matching wird aber nur akzeptiert, falls der Wert der Metrik unter dem Wert von MatchThreshold (*'ssd'*, *'sad'*) bzw. über demselben (*'ncc'*) liegt.

Zur Performanzsteigerung kann der Suchbereich für die Matchings eingeschränkt werden. Nur Punkte innerhalb eines $2 \cdot$ RowTolerance $\times 2 \cdot$ ColTolerance Punkte großen Fensters werden betrachtet. Die Verschiebung

des Mittelpunkts dieses Fensters im zweiten Bild gegenüber der Position des aktuellen Punktes im ersten Bild wird durch die Parameter `RowMove` und `ColMove` bestimmt.

Enthält die gesuchte Transformation eine Drehung, ist also das eine gegenüber dem anderen Bild gedreht, so kann im Parameter `Rotation` eine Schätzung des Drehwinkels bzw. ein Winkelintervall im Bogenmaß übergeben werden. Eine gute Schätzung des Winkels erhöht die Qualität des Grauwertwertmatchings. Falls sich die tatsächliche Rotation zu stark von der angegebenen Schätzung unterscheidet, schlägt das Matching typischerweise fehl. Je größer das angegebene Winkelintervall, desto langsamer läuft der Operator, denn für alle relevanten (endlich vielen) Winkel innerhalb des Intervalls wird das Verfahren komplett durchlaufen.

Ist das initiale Punktmatching bestimmt, wird es anschließend durch einen randomisierten Auswahlalgorithmus (RANSAC) zur Bestimmung der Transformationsmatrix `HomMat2D` benutzt. Dabei wird versucht die Matrix so zu wählen, dass sie bezüglich der Schranke `DistanceThreshold` zu möglichst vielen Punktpaaren konsistent ist.

Ist eine solche Auswahl getroffen, wird die zugehörige Matrix unter Einbeziehung aller konsistenten Punkte nochmals optimiert. Dabei kann mit `EstimationMethod` zwischen der langsameren, dafür aber mathematisch optimalen *'gold_standard'* Methode und der schnelleren *'normalized_dlt'* gewählt werden. Dabei werden die Verfahren von `vector_to_proj_hom_mat2d` verwendet.

Punktepaare, die die Konsistenzbedingung für diese endgültige Transformation immer noch verletzen, werden aussortiert, der Rest als Steuerparameter zurückgeliefert. `Points1` enthält die Indizes der gematchten Eingabepunkte im ersten Bild, `Points2` die Indizes der dazu korrespondierenden Punkte im zweiten Bild.

Der Parameter `RandSeed` kann benutzt werden, um das randomisierte Verhalten des RANSAC-Verfahrens zu kontrollieren und somit reproduzierbare Ergebnisse zu erhalten. Wird `RandSeed` auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit dem angegebenen Startwert initialisiert wird. Ist `RandSeed` $= 0$, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Parameter

- ▷ **`Image1`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 1.
- ▷ **`Image2`** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 2.
- ▷ **`Rows1`** (input_control) ..... point.x-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 1.
- ▷ **`Cols1`** (input_control) ..... point.y-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 1.
- ▷ **`Rows2`** (input_control) ..... point.x-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 2.
- ▷ **`Cols2`** (input_control) ..... point.y-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 2.
- ▷ **`GrayMatchMethod`** (input_control) ..... string ⇝ *string*
  Metrik für den Vergleich der Grauwerte.
  Defaultwert: 'ssd'
  Werteliste: `GrayMatchMethod` ∈ {'ssd', 'sad', 'ncc'}
- ▷ **`MaskSize`** (input_control) ..... integer ⇝ *integer*
  Größe der Grauwertmaske.
  Defaultwert: 10
  Typischer Wertebereich: MaskSize ≤ 90
- ▷ **`RowMove`** (input_control) ..... integer ⇝ *integer*
  Mittlere Zeilenverschiebung.
  Defaultwert: 0
- ▷ **`ColMove`** (input_control) ..... integer ⇝ *integer*
  Mittlere Spaltenverschiebung.
  Defaultwert: 0
- ▷ **`RowTolerance`** (input_control) ..... integer ⇝ *integer*
  Halbe Höhe des Suchfensters für das Punktmatching.
  Defaultwert: 256

▷ **ColTolerance** (input_control) . . . . . integer ⇝ *integer*
Halbe Breite des Suchfensters für das Punktmatching.
Defaultwert: 256

▷ **Rotation** (input_control) . . . . . real(-array) ⇝ *real*
Drehwinkelbereich.
Defaultwert: 0.0
Wertevorschläge: `Rotation` ∈ {0.0, 0.7854, 1.571, 3.142}

▷ **MatchThreshold** (input_control) . . . . . number ⇝ *integer* / real
Schwellenwert für Grauwertkorrespondenzen.
Defaultwert: 10
Wertevorschläge: `MatchThreshold` ∈ {10, 20, 50, 100, 0.9, 0.7}

▷ **EstimationMethod** (input_control) . . . . . string ⇝ *string*
Schätzalgorithmus für Transformationsmatrix.
Defaultwert: 'normalized_dlt'
Werteliste: `EstimationMethod` ∈ {'normalized_dlt', 'gold_standard'}

▷ **DistanceThreshold** (input_control) . . . . . real ⇝ *real*
Schwelle für Konsistenzüberprüfung.
Defaultwert: 0.2

▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
Startwert für den Zufallszahlengenerator.
Defaultwert: 0

▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
Homogene projektive Transformationsmatrix.

▷ **Points1** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der akzeptierten Eingabepunkte aus Bild 1.

▷ **Points2** (output_control) . . . . . integer-array ⇝ *integer*
Indizes der akzeptierten Eingabepunkte aus Bild 2.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`points_foerstner`, `points_harris`

Nachfolger

`projective_trans_image`, `projective_trans_image_size`, `projective_trans_region`, `projective_trans_contour_xld`, `projective_trans_point_2d`, `projective_trans_pixel`

Alternativen

`hom_vector_to_proj_hom_mat2d`, `vector_to_proj_hom_mat2d`

Siehe auch

`proj_match_points_ransac_guided`

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2000.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Matching

proj_match_points_ransac_guided (Image1, Image2 : : Rows1, Cols1, Rows2, Cols2, GrayMatchMethod, MaskSize, HomMat2DGuide, DistanceTolerance, MatchThreshold, EstimationMethod, DistanceThreshold, RandSeed : HomMat2D, Points1, Points2)

Bestimmt eine projektive Transformationsmatrix automatisch durch Zuordnung von Punkten zwischen zwei Bildern basierend auf einer bekannten Näherung der projektiven Transformationsmatrix.

Ausgehend von einer Menge charakteristischer Punkte (Cols1, Rows1) und (Cols2, Rows2) in den beiden Eingabebildern Image1 und Image2 und einer bekannten Näherung HomMat2DGuide für die Transformation von Image1 zu Image2 bestimmt proj_match_points_ransac_guided automatisch die Korrespondenz der Punkte und die homogene projektive Transformationsmatrix HomMat2D, die die korrespondierenden Punkte in den beiden Bildern am besten ineinander überführt. Die charakteristischen Punkte können z.B. mit points_foerstner oder points_harris extrahiert werden. Die Näherung HomMat2DGuide kann z.B. mit proj_match_points_ransac auf niedriger aufgelösten Versionen von Image1 zu Image2 bestimmt werden.

Die Bestimmung der Transformation erfolgt in 2 Schritten: Zuerst werden die Grauwertkorrelationen von Umgebungen der Eingabepunkte im ersten und zweiten Bild bestimmt und anhand dieser ein initiales Matching zwischen den Punkten ermittelt. Die Größe dieser Grauwertfenster beträgt MaskSize $\times$ MaskSize . Es können drei Metriken für die Korrelation gewählt werden. Hat GrayMatchMethod den Wert *'ssd'*, so wird das Quadrat der Grauwertdifferenzen verwendet, *'sad'* entspricht dem Betrag der Grauwertdifferenzen und *'ncc'* ist die normierte Kreuzkorrelation (siehe auch binocular_disparity). Diese Metrik wird über alle Punktepaare minimiert (*'ssd'*, *'sad'*) bzw. maximiert (*'ncc'*), ein so gefundenes Matching wird aber nur akzeptiert, falls der Wert der Metrik unter dem Wert von MatchThreshold (*'ssd'*, *'sad'*) bzw. über demselben (*'ncc'*) liegt.

Zur Performanzsteigerung wird der Suchbereich für die Matchings basierend auf der Näherung der Transformation HomMat2DGuide eingeschränkt. Nur Punkte innerhalb eines Abstandes von DistanceTolerance um die Transformation eines Punktes aus Image1 mittelsHomMat2DGuide in Image2 liegen, werden für das Matching in Betracht gezogen.

Ist das initiale Punktmatching bestimmt, wird es anschließend durch einen randomisierten Auswahlalgorithmus (RANSAC) zur Bestimmung der Transformationsmatrix HomMat2D benutzt. Dabei wird versucht die Matrix so zu wählen, dass sie bezüglich der Schranke DistanceThreshold zu möglichst vielen Punktpaaren konsistent ist. DistanceThreshold sollte also kleiner als DistanceTolerance sein.

Ist eine solche Auswahl getroffen, wird die zugehörige Matrix unter Einbeziehung aller konsistenten Punkte nochmals optimiert. Dabei kann mit EstimationMethod zwischen der langsameren, dafür aber mathematisch optimalen *'gold_standard'* Methode und der schnelleren *'normalized_dlt'* gewählt werden. Dabei werden die Verfahren von vector_to_proj_hom_mat2d verwendet.

Punktepaare, die die Konsistenzbedingung für diese endgültige Transformation immer noch verletzen, werden aussortiert, der Rest als Steuerparameter zurückgeliefert. Points1 enthält die Indizes der gematchten Eingabepunkte im ersten Bild, Points2 die Indizes der dazu korrespondierenden Punkte im zweiten Bild.

Der Parameter RandSeed kann benutzt werden, um das randomisierte Verhalten des RANSAC-Verfahrens zu kontrollieren und somit reproduzierbare Ergebnisse zu erhalten. Wird RandSeed auf einen positiven Wert gesetzt, so liefert der Operator bei jedem Aufruf mit denselben Parametern auch dasselbe Resultat, da der intern verwendete Zufallsgenerator mit dem angegebenen Startwert initialisiert wird. Ist RandSeed $= 0$, so wird der Zufallsgenerator mit der aktuellen Zeit initialisiert. Daher sind in diesem Fall die Ergebnisse unter Umständen nicht reproduzierbar.

Parameter

- ▷ **Image1** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 1.
- ▷ **Image2** (input_object) ..... singlechannelimage ⇝ *object :* byte / uint2
  Eingabebild 2.
- ▷ **Rows1** (input_control) ..... point.x-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 1.
- ▷ **Cols1** (input_control) ..... point.y-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 1.
- ▷ **Rows2** (input_control) ..... point.x-array ⇝ *real* / integer
  Zeilenkoordinaten charakteristischer Punkte in Bild 2.

- ▷ **Cols2** (input_control) . . . . . point.y-array ⇝ *real* / integer
  Spaltenkoordinaten charakteristischer Punkte in Bild 2.
- ▷ **GrayMatchMethod** (input_control) . . . . . string ⇝ *string*
  Metrik für den Vergleich der Grauwerte.
  Defaultwert: 'ssd'
  Werteliste: `GrayMatchMethod` ∈ {'ssd', 'sad', 'ncc'}
- ▷ **MaskSize** (input_control) . . . . . integer ⇝ *integer*
  Größe der Grauwertmaske.
  Defaultwert: 10
  Typischer Wertebereich: MaskSize ≤ 90
- ▷ **HomMat2DGuide** (input_control) . . . . . hom_mat2d ⇝ *real*
  Näherung der projektiven Transformationsmatrix zwischen den zwei Bildern.
- ▷ **DistanceTolerance** (input_control) . . . . . real ⇝ *real*
  Toleranz des Suchfensters für das Punktmatching.
  Defaultwert: 20.0
  Wertevorschläge: `DistanceTolerance` ∈ {0.2, 0.5, 1.0, 2.0, 3.0, 5.0, 10.0, 20.0, 50.0}
- ▷ **MatchThreshold** (input_control) . . . . . number ⇝ *integer* / real
  Schwellenwert für Grauwertkorrespondenzen.
  Defaultwert: 10
  Wertevorschläge: `MatchThreshold` ∈ {10, 20, 50, 100, 0.9, 0.7}
- ▷ **EstimationMethod** (input_control) . . . . . string ⇝ *string*
  Schätzalgorithmus für Transformationsmatrix.
  Defaultwert: 'normalized_dlt'
  Werteliste: `EstimationMethod` ∈ {'normalized_dlt', 'gold_standard'}
- ▷ **DistanceThreshold** (input_control) . . . . . real ⇝ *real*
  Schwelle für Konsistenzüberprüfung.
  Defaultwert: 0.2
- ▷ **RandSeed** (input_control) . . . . . integer ⇝ *integer*
  Startwert für den Zufallszahlengenerator.
  Defaultwert: 0
- ▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **Points1** (output_control) . . . . . integer-array ⇝ *integer*
  Indizes der akzeptierten Eingabepunkte aus Bild 1.
- ▷ **Points2** (output_control) . . . . . integer-array ⇝ *integer*
  Indizes der akzeptierten Eingabepunkte aus Bild 2.

Beispiel

```
zoom_image_factor (Image1, Image1Zoomed, 0.5, 0.5, 'constant')
zoom_image_factor (Image2, Image2Zoomed, 0.5, 0.5, 'constant')
points_foerstner (Image1Zoomed, 1, 2, 3, 200, 0.3, 'gauss', 'false', \
                  Rows1, Cols1, _, _, _, _, _, _, _, _)
points_foerstner (Image2Zoomed, 1, 2, 3, 200, 0.3, 'gauss', 'false', \
                  Rows2, Cols2, _, _, _, _, _, _, _, _)
get_image_pointer1 (Image1Zoomed, Pointer, Type, Width, Height)
proj_match_points_ransac (Image1Zoomed, Image2Zoomed, Rows1, Cols1, \
                           Rows2, Cols2, 'ncc', 10, 0, 0, \
                           Height, Width, 0, 0.5, 'gold_standard', \
                           5, 0, HomMat2D, Points1, Points2)
hom_mat2d_scale_local (HomMat2D, 0.5, 0.5, HomMat2DGuide)
hom_mat2d_scale (HomMat2DGuide, 2, 2, 0, 0, HomMat2DGuide)
points_foerstner (Image1, 1, 2, 3, 200, 0.3, 'gauss', 'false', \
                  Rows1, Cols1, _, _, _, _, _, _, _, _)
points_foerstner (Image2, 1, 2, 3, 200, 0.3, 'gauss', 'false', \
                  Rows2, Cols2, _, _, _, _, _, _, _, _)
proj_match_points_ransac_guided (Image1, Image2, Rows1, Cols1, \
                                  Rows2, Cols2, 'ncc', 10, \
```

```
HomMat2DGuide, 40, 0.5, \
'gold_standard', 10, 0, HomMat2D, \
Points1, Points2)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

points_foerstner, points_harris

Nachfolger

projective_trans_image, projective_trans_image_size, projective_trans_region, projective_trans_contour_xld, projective_trans_point_2d, projective_trans_pixel

Alternativen

hom_vector_to_proj_hom_mat2d, vector_to_proj_hom_mat2d

Siehe auch

proj_match_points_ransac

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2000.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Matching

Kapitel 27

Transformationen

27.1 2D-Transformationen

Um den Ort eines Punktes in einem Bild festzulegen, braucht es eine Konvention, wie dies geschehen soll. Eine solche Konvention wird durch das Koordinatensystem festgelegt. In HALCON werden verschiedene Koordinatensysteme verwendet. Diese Kapitelreferenz gibt einen Überblick über die Systeme, die in 2D verwendet werden.

Pixel sind eigenständig und um sie zu adressieren gibt es ein Koordinatensystem, welches nur die ganzen Zahlen verwendet, das Pixel-Koordinatensystem. Mit einer feineren Auflösung werden Koordinaten in Fließkommazahlen benötigt, z.B. $(3.6, 4.1)$. Dies führt zu Subpixel-genauen Koordinatensystemen. In HALCON verwenden wir drei verschiedene Implementierungen dieser Subpixel-Koordinatensystemen:

- Pixel-zentrierte Koordinaten, das HALCON Standard-Koordinatensystem
- Ecken-zentrierte Koordinaten
- Polarkoordinaten

Davon unterscheiden sich die ersten beiden bloß in der Wahl des Ursprungspunktes, wie in den folgenden Abbildungen ersichtlich wird. Kalibrierung ermöglicht die Zuordnung von Bild-Koordinaten und reellen Distanzen. Für mehr Informationen zu kalibrierten Koordinaten wird auf den „`Solution Guide III-C - 3D Vision`“ verwiesen.

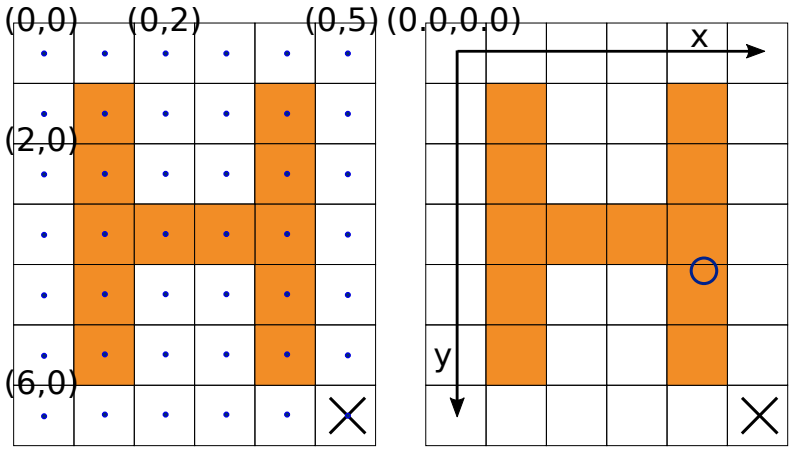
HALCON Standard-Koordinatensystem

Pixel-genaues Koordinaten Das Pixel-Koordinatensystem behandelt das Bild wie ein Netz diskreter Einheiten, den Pixeln. In HALCON setzen wir den Ursprung $(0, 0)$ in die Mitte des Pixels oben links. Zur Zuordnung der Koordinaten eines Pixels werden wie in einer Matrix Zeile und Spalte angegeben.

Für ein Bild der Größe Höhe $\times$ Breite = $m \times n$ Pixel bedeutet dies, dass die Zeilen-Koordinate von 0 bis $m-1$ läuft und die Spalten-Koordinate von 0 bis $n-1$, eine Visualisierung findet sich in der unten folgenden Abbildung.

Subpixel-genaues Koordinatensystem: Pixel-zentriert Der Ursprung dieses Koordinatensystems liegt im Zentrum des Pixels oben links und damit bettet diese Konvention das Pixel-Koordinatensystem ein. Dies bedeutet, dass die Ecke oben links die Koordinaten $(-0.5, -0.5)$ hat und für ein Bild der Größe Höhe $\times$ Breite = $m \times n$ Pixel hat die Ecke unten rechts die Koordinaten $(m - 0.5, n - 0.5)$ (=$(m - 1 + 0.5, n - 1 + 0.5)$, die Koordinaten-Werte starten bei 0). Der Pixel (k,l) bedeckt die Fläche des Rechteckes $(k - 0.5, l - 0.5)$, $(k + 0.5, l - 0.5)$, $(k - 0.5, l + 0.5)$, $(k + 0.5, l + 0.5)$. Diese Konvention wird in HALCON das Standard-Koordinatensystem oder auch das Bild-Koordinatensystem genannt.

Dieses Koordinatensystem ist wie eine Projektion der xy-Achse des 3D-Kamera-Koordinatensystems. Entsprechend haben die Achsen die Zuordnung Zeile: y-Koordinate, Spalte: x-Koordinate.



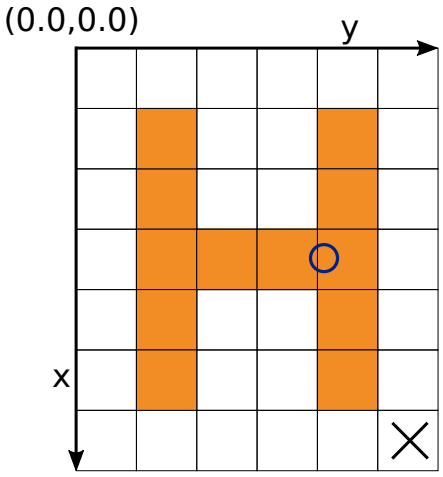
(1) (2)

Darstellung der standard Pixel- und Subpixel-Koordinatensysteme in HALCON. Das Kreuz bezeichnet den Pixel in der Ecke unten rechts. Seine Mitte hat die Koordinaten $(6, 5)$ (in Pixel-Koordinaten (1)), $(6.0, 5.0)$ (Standard-Koordinaten (2)). Das Kreiszentrum hat die Koordinaten $(3.6, 4.1)$.

Kartesisches Nicht-Standard Koordinatensystem in HALCON

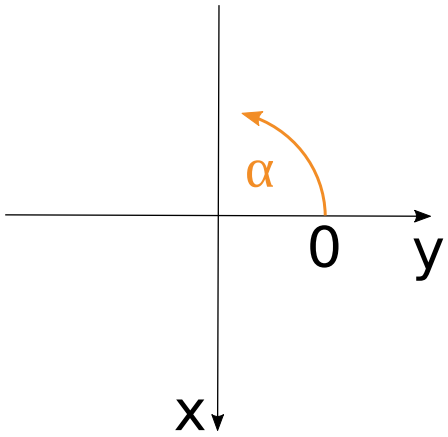
Wird ein Bild um $\pi/2$ (=90 Grad) um seinen Ursprung rotiert, so erwartet man, dass sich ursprüngliches Bild und gedrehtes Bild in einer Ecke berühren, sich aber nicht überlagern. Ebenso soll eine Skalierung des Bildes nicht zu negativen Bild-Koordinaten führen. Dafür muss der Ursprung $(0.0, 0.0)$ in einer Ecke sein. Dies motiviert das folgende Koordinatensystem.

Subpixel-genaues Koordinatensystem: Ecken-zentriert: In diesem Koordinatensystem liegt der Ursprung $(0.0, 0.0)$ in der Ecke oben links. Somit hat das Zentrum des Pixels oben links die Koordinaten $(0.5, 0.5)$ und für ein Bild der Größe Höhe $\times$ Breite = $m \times n$ Pixel hat die Ecke unten rechts die Koordinaten (m, n). Der Pixel (k, l) bedeckt die Fläche des Rechtecks (k, l), $(k, l + 1)$, $(k + 1, l)$, $(k + 1, l + 1)$.



Darstellung des kartesischen Ecken-zentrierten Subpixel-genauen Koordinatensystems in HALCON. Das Kreuz bezeichnet den Pixel in der Ecke unten rechts. Seine Mitte hat die Koordinaten $(6.5, 5.5)$. Das Kreiszentrum hat auch in dieser Darstellung die Koordinaten $(3.6, 4.1)$.

Für dieses Koordinatensystem sind Rotationen im mathematisch positiven Drehsinn und somit gegen den Uhrzeigersinn definiert. Somit bildet eine Rotation um $\pi/2$ (=90 Grad) die erste Achse (= x-Achse) auf die zweite Achse (= y-Achse) ab. Entsprechend sind die Achsen wie folgt zugeordnet, Zeile: x-Koordinate, Spalte: y-Koordinate.



Darstellung einer Rotation (α) im Ecken-zentrierten Koordinatensystem.

Operatoren, welche die Parameter in beliebigen Kartesischen Koordinatensysteme akzeptieren

Der Operator `affine_trans_point_2d` wendet die durch `HomMat2D` definierte Transformation auf die Punkt-Koordinaten an. Das heißt, der Operator `affine_trans_point_2d` kann in beiden Kartesischen Koordinatensystemen verwendet werden, solange die Transformation und die Punkte im selben Koordinatensystem gegeben werden.

Die Operatoren `angle_ll` und `angle_lx` können zwar Eingabepunkte in Pixel-zentrierten Koordinaten erhalten, doch der Winkel wird stets nach der Konvention der Rotationen im mathematisch positiven Drehsinn zurückgegeben, d.h., gegen den Uhrzeigersinn, und mit der horizontalen Achse als *0*-Wert, wie im ecken-zentrierter Koordinatensystem.

Operatoren, welche die Parameter in verschiedenen Koordinatensystemen erwarten

In HALCON gibt es Operatoren, welche die Eingaben in verschiedenen Koordinatensystemen erwarten. Einerseits ist das Objekt in seinen üblichen Koordinaten, den Standard-Koordinaten anzugeben. Andererseits erwartet der Operator für die Transformationsmatrix `HomMat2D` ecken-zentrierte Koordinaten mit deren oben erläuterten Vorteilen bezüglich Transformationen. Diese Operatoren transformieren die Koordinaten des Objekts von Standard-Koordinaten (mit dem Ursprung in der Mitte des Pixels oben links) in ecken-zentrierte Koordinaten (mit dem Ursprung in der Ecke oben links des Pixels oben links). Nach der Transformation mit `HomMat2D` wird das Resultat zurücktransformiert und in Standard-Koordinaten ausgegeben.

Dies gilt für folgende Operatoren:

- `affine_trans_contour_xld`
- `affine_trans_image`
- `affine_trans_image_size`
- `affine_trans_pixel`
- `affine_trans_polygon_xld`
- `affine_trans_region`
- `projective_trans_contour_xld`
- `projective_trans_image`
- `projective_trans_image_size`
- `projective_trans_pixel`
- `projective_trans_region`

Eine Transformation (z.B. eine Rotation um einen bestimmten Punkt), die durch eine Matrix im Standard-Koordinatensystem repräsentiert wird, kann z.B. über folgende Transformation als entsprechende Matrix in Ecken-zentrierten Koordinaten notiert werden:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
```

Es ist zu beachten, dass die zuvor erwähnten Operatoren beginnend mit `projective_` eine projektive Transformationsmatrix verwenden. Diese Transformationsmatrix kann z.B. über 3D Kamera-Posen erhalten werden. Wird dies getan, so ist die verwendete Matrix in einer Projektion der xy-Ebene des 3D Koordinatensystems gegeben. Entsprechend haben die Achsen die Zuordnung Zeile: y-Koordinate, Spalte: x-Koordinate.

Form-basiertes Matching und Korrelations-basiertes Matching

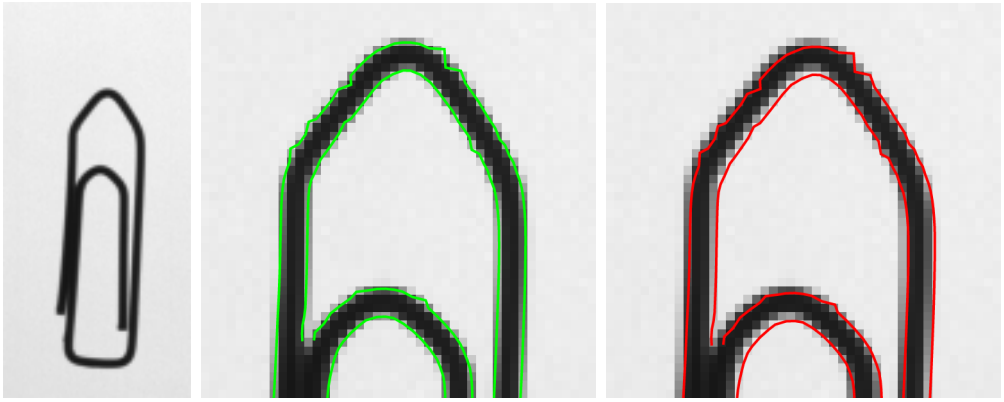
Die Resultate von form-basiertem Matching, wie z.B. `find_shape_model`, werden in ecken-zentrierten Koordinaten gegeben. Somit kann man diese Resultate direkte zur Erstellung von Transformationsmatrizen `HomMat2D` in ecken-zentrierten Koordinaten verwenden. Solche Transformationsmatrizen können mit Operatoren wie z.B. `affine_trans_contour_xld` und weiteren unter `Operatoren, welche die Parameter in verschiedenen Koordinatensystemen erwarten` gelisteten Operatoren angewendet werden.

Die Verwendung der Prozedur `dev_display_shape_matching_results` wird zur Darstellung der Resultate von form-basiertem Matching, insbesondere der Operatoren

- `find_aniso_shape_model`
- `find_aniso_shape_models`
- `find_scaled_shape_model`
- `find_scaled_shape_models`
- `find_shape_model`
- `find_shape_models`

stark empfohlen.

Die folgenden Abbildungen zeigen ein Beispiel, wie die Darstellung eines Matches aussieht, wenn die verwendete Transformationsmatrix in den korrekten, respektive falschen Koordinaten angegeben wird. Für Letztgenanntes, gezeigt in Abbildung (3), wurde die Transformationsmatrix in Pixel-zentrierten Koordinaten verwendet und so liegt der gezeigte Match mit `affine_trans_contour_xld` um 0.5 Pixel falsch. Man beachte, dass dieser Effekt nur sichtbar wird, wenn Rotationen involviert sind.



(1) (2) (3)

Das Originalbild der Büroklammer (1), ein Teil des Matches mit den Eingabegrößen in vom Operator erwarteten Koordinaten (2) und ein Match, bei dem die Transformationsmatrix in nicht-passenden Koordinaten verwendet wurde.

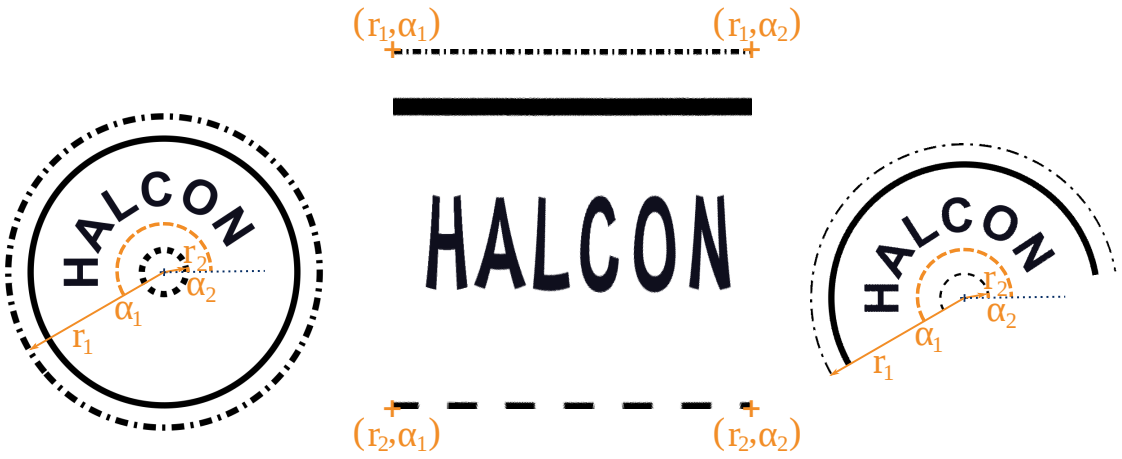
Die Resultate von find_ncc_model und find_ncc_models werden ebenso in ecken-zentrierten Koordinaten gegeben. Deshalb kann man auch mit diesen Resultaten direkt eine Transformationsmatrix HomMat2D in ecken-zentrierten Koordinaten erstellen. Diese Transformationsmatrizen sind zur direkten Anwendung mit Operatoren wie z.B. affine_trans_contour_xld und weiteren unter Operatoren, welche die Parameter in verschiedenen Koordinatensystemen erwarten gelisteten Operatoren geeignet.

Zur Darstellung der Resultate korrelations-basierten Matchings wird die Prozedur dev_display_ncc_matching_results stark empfohlen.

Nicht-Kartesische Koordinatensysteme

Subpixel-genaues Koordinatensystem: Polarkoordinaten In Polarkoordinaten werden Punkte durch einen Abstand und einen Winkel definiert. Der Abstand, Radialkoordinate genannt, wird zum festen Punkt, dem Pol, angegeben. Die Winkelkoordinate wird in Bezug zu einer fixen Achse, der Polarachse gemessen. In HALCON wird der Pol über (Row, Column) festgelegt und die horizontalen Achse bildet die Polarachse.

Bei einer Transformation mit polar_trans_image_ext entspricht der Pixel oben links des Ausgabebildes jenem Punkt des Kreisrings im Eingabebild, der durch RadiusStart und AngleStart festgelegt wurde. Analog entspricht der Pixel unten rechts des Ausgabebildes jenem Punkt des Kreisrings im Eingabebild, der durch RadiusEnd und AngleEnd spezifiziert wurde. Dies bedeutet, dass in der üblichen Verwendungsweise (AngleStart $<$ AngleEnd und RadiusStart $<$ RadiusEnd) die Transformation im mathematisch positiven Drehsinn (für den Betrachter gegen den Uhrzeigersinn) durchgeführt wird. Weiter werden dann Punkte mit kleinerem Radius weiter oben im Ausgabebild angezeigt. Durch geeignetes Austauschen der Parameterwerte (z.B. AngleStart $>$ AngleEnd oder RadiusStart $>$ RadiusEnd) kann jeder beliebige Drehsinn des Ausgabebildes erreicht werden.



(1) (2) (3)

Der Kreisbogen ist gegeben durch den Pol (Row,Column) (+), die Polarachse (...), die zwei Winkelkoordinaten AngleStart ($\alpha 1$), AngleEnd ($\alpha 2$) und die zwei Radialkoordinaten RadiusStart ($r1$), RadiusEnd ($r2$). (1) Das Originalbild sowie die den Kreisbogen definierenden Parameter. (2) Der Kreisbogen, erhalten durch polar_trans_image_ext, in einer Darstellung bei der die Polarkoordinaten Winkel und Radius ein äquidistantes Koordinatennetz formen. (3) Der Kreisbogen in der selben Darstellung wie das Originalbild. Die Kartesischen Koordinaten wurden durch Anwendung von polar_trans_image_inv auf Bild (2) erhalten. Der Ursprung liegt im Zentrum des Pixels oben links.

Folgende Operatoren verwenden Polarkoordinaten:

- polar_trans_image_ext
- polar_trans_image_inv
- polar_trans_region
- polar_trans_region_inv
- polar_trans_contour_xld
- polar_trans_contour_xld_inv

- polar_trans_image (legacy)

Bilder mit reduzierter Domäne, Regionen und Modelle

In den vorherigen Abschnitten wurden die Koordinaten von Bildern behandelt. In Bezug auf den Ursprung und das benutzte Koordinatensystem werden Bilder mit reduzierter Domäne, Regionen und Modelle jedoch unterschiedlich behandelt.

Bilder mit reduzierter Domäne und Regionen Sowohl Bilder mit reduzierter Domäne als auch Regionen behalten das Koordinatensystem des Bildes, auf welchem sie erstellt wurden. Dies bedeutet, dass sie den Ursprung erben und die Koordinatenwerte, die sie im Originalbild hatten, behalten.

Modelle Modelle, können ein lokales Koordinatensystem haben. So haben z.B. Modelle, die über create_shape_model erhalten wurden, ihren Ursprung im Schwerpunkt der ROI, aus der sie erstellt wurden. Für mehr Informationen wird auf den „Solution Guide II-B - Matching" verwiesen.

Kalibrierte Koordinaten

Aus Pixel-Einheiten kann keine direkte Information über reelle Distanzen gewonnen werden. Durch die Kalibrierung der Kamera ist es jedoch möglich, die Bilder zu entzerren. Dann können den Bildern Welt-Koordinaten zugeordnet werden. Für mehr Informationen wird auf den „Solution Guide III-C - 3D Vision" verwiesen.

affine_trans_pixel (: : HomMat2D, Row, Col : RowTrans, ColTrans)

Wendet eine beliebige affine 2D-Transformation auf Pixelkoordinaten an.

affine_trans_pixel wendet eine beliebige affine 2D-Transformation (Skalierung, Drehung, Verschiebung, Scherung) auf die Eingabepixel (Row,Col) an und liefert die Ergebnispixel in (RowTrans,ColTrans) zurück; die Ein- und Ausgabepixel sind jeweils subpixelgenaue Punkte. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in HomMat2D übergeben wird.

Im Gegensatz zu affine_trans_point_2d transformiert affine_trans_pixel die Eingabepixelkoordinaten zuerst vom HALCON-Standard-Koordinatensystem (mit dem Ursprung im Mittelpunkt des oberen linken Pixels) in ein Koordinatensystem mit dem Ursprung in der oberen linken Ecke des oberen linken Pixels. Das Ergebnis der darauf folgenden Transformation mit HomMat2D wird dann wieder in das Standard-Koordinatensystem zurücktransformiert. Dadurch ist affine_trans_pixel mit den Operatoren affine_trans_image, affine_trans_image_size, affine_trans_region, affine_trans_contour_xld und affine_trans_polygon_xld kompatibel.

Eine Transformation mit affine_trans_pixel entspricht der folgenden Kette von Transformationen (Ein- und Ausgabepixel als homogene Vektoren dargestellt):

$$\begin{pmatrix} \mathtt{RowTrans} \\ \mathtt{ColTrans} \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & -0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \mathtt{HomMat2D} \cdot \begin{bmatrix} 1 & 0 & +0.5 \\ 0 & 1 & +0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} \mathtt{Row} \\ \mathtt{Col} \\ 1 \end{pmatrix}$$

Damit entspricht

```
affine_trans_pixel (HomMat2D, Row, Col, RowTrans, ColTrans)
```

der folgenden Operatorfolge:

```
affine_trans_point_2d (HomMat2D, Row+0.5, Col+0.5, RowTmp, ColTmp)
RowTrans := RowTmp-0.5
ColTrans := ColTmp-0.5
```

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Row** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Eingabepixel (Zeilenkoordinate).
  Defaultwert: 64
  Wertevorschläge: `Row` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Col** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Eingabepixel (Spaltenkoordinate).
  Defaultwert: 64
  Wertevorschläge: `Col` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **RowTrans** (output_control) . . . . . point.x(-array) ⇝ *real*
  Ausgabepixel (Zeilenkoordinate).
- ▷ **ColTrans** (output_control) . . . . . point.y(-array) ⇝ *real*
  Ausgabepixel (Spaltenkoordinate).

Ergebnis

Falls die Matrix `HomMat2D` eine affine Abbildung darstellt (d.h. keine projektive Abbildung), liefert `affine_trans_pixel` den Wert TRUE zurück. Ansonsten wird eine Ausnahmebehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`hom_mat2d_translate`, `hom_mat2d_translate_local`, `hom_mat2d_scale`, `hom_mat2d_scale_local`, `hom_mat2d_rotate`, `hom_mat2d_rotate_local`, `hom_mat2d_slant`, `hom_mat2d_slant_local`, `hom_mat2d_reflect`, `hom_mat2d_reflect_local`

Alternativen

`affine_trans_point_2d`

Modul

Foundation

affine_trans_point_2d (: : HomMat2D, Px, Py : Qx, Qy)

Wendet eine beliebige affine 2D-Transformation auf Punkte an.

`affine_trans_point_2d` wendet eine beliebige affine 2D-Transformation (Skalierung, Drehung, Verschiebung, Scherung) auf die Eingabepunkte (`Px`,`Py`) an und liefert die Ergebnispunkte in (`Qx`,`Qy`) zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in `HomMat2D` übergeben wird. Dies entspricht der folgenden Gleichung (Ein- und Ausgabepunkte als homogene Vektoren dargestellt):

$$\begin{pmatrix} Qx \\ Qy \\ 1 \end{pmatrix} = HomMat2D \cdot \begin{pmatrix} Px \\ Py \\ 1 \end{pmatrix}$$

Falls die zu transformierenden Punkte in Standard-Bildkoordinaten vorliegen, müssen die *Zeilen*-Koordinaten der Punkte in `Px` und die *Spalten*-Koordinaten in `Py` übergeben werden. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Die Transformationsmatrix kann mit Hilfe der Operatoren `hom_mat2d_identity`, `hom_mat2d_rotate`, `hom_mat2d_translate` etc. aufgebaut werden oder das Resultat von Operatoren wie `vector_angle_to_rigid` sein.

Falls zum Beispiel HomMat2D eine starre Transformation ist, also aus Rotation und Translation besteht, werden die Punkte folgendermaßen transformiert:

$$\begin{pmatrix} Qx \\ Qy \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0\,0 & 1 \end{bmatrix} \cdot \begin{pmatrix} Px \\ Py \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} \cdot \begin{pmatrix} Px \\ Py \end{pmatrix} + \mathbf{T} \\ 1 \end{pmatrix}$$

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Px** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Eingabepunkt(e) (x- bzw. Zeilen-Koordinate).
  Defaultwert: 64
  Wertevorschläge: Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Py** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Eingabepunkt(e) (y- bzw. Spalten-Koordinate).
  Defaultwert: 64
  Wertevorschläge: Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Qx** (output_control) . . . . . point.x(-array) ⇝ *real*
  Ausgabepunkt(e) (x- bzw. Zeilen-Koordinate).
- ▷ **Qy** (output_control) . . . . . point.y(-array) ⇝ *real*
  Ausgabepunkt(e) (y- bzw. Spalten-Koordinate).

Ergebnis

Falls die Matrix HomMat2D eine affine Abbildung darstellt (d.h. keine projektive Abbildung), liefert affine_trans_point_2d den Wert TRUE zurück. Ansonsten wird eine Ausnahmebehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

hom_mat2d_translate, hom_mat2d_translate_local, hom_mat2d_scale, hom_mat2d_scale_local, hom_mat2d_rotate, hom_mat2d_rotate_local, hom_mat2d_slant, hom_mat2d_slant_local, hom_mat2d_reflect, hom_mat2d_reflect_local

Nachfolger

hom_mat2d_translate, hom_mat2d_translate_local, hom_mat2d_scale, hom_mat2d_scale_local, hom_mat2d_rotate, hom_mat2d_rotate_local, hom_mat2d_slant, hom_mat2d_slant_local, hom_mat2d_reflect, hom_mat2d_reflect_local

Modul

Foundation

deserialize_hom_mat2d (: : SerializedItemHandle : HomMat2D)

Deserialisieren einer homogenen 2D-Transformationsmatrix.

deserialize_hom_mat2d deserialisiert eine homogene 2D-Transformationsmatrix, welche mit serialize_hom_mat2d serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte Transformationsmatrix wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einer dafür automatisch erzeugten Transformationsmatrix mit dem Handle HomMat2D gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_hom_mat2d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_hom_mat2d`

Modul

Foundation

hom_mat2d_compose (: : HomMat2DLeft,
HomMat2DRight : HomMat2DCompose)

Multipliziert zwei homogene 2D-Transformationsmatrizen.

`hom_mat2d_compose` generiert eine neue 2D-Transformationsmatrix durch Multiplikation der zwei Eingabe-Transformationsmatrizen:

$$\texttt{HomMat2DCompose} = \texttt{HomMat2DLeft} \cdot \texttt{HomMat2DRight}$$

Falls zum Beispiel die Eingabematrizen starre Transformationen darstellen, also Transformationen bestehend aus Rotation und Translation, wird die Ergebnismatrix wie folgt bestimmt:

$$\texttt{HomMat2DCompose} = \begin{bmatrix} \mathbf{R}_l & \mathbf{T}_l \\ 0\,0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}_r & \mathbf{T}_r \\ 0\,0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_l \cdot \mathbf{R}_r & \mathbf{R}_l \cdot \mathbf{T}_r + \mathbf{T}_l \\ 0 \quad 0 & 1 \end{bmatrix}$$

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

▷ **HomMat2DLeft** (input_control) . . . . . hom_mat2d ⇝ *real*
Linke Eingabe-Transformationsmatrix.

▷ **HomMat2DRight** (input_control) . . . . . hom_mat2d ⇝ *real*
Rechte Eingabe-Transformationsmatrix.

▷ **HomMat2DCompose** (output_control) . . . . . hom_mat2d ⇝ *real*
Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `hom_mat2d_compose` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat2d_compose`, `hom_mat2d_translate`, `hom_mat2d_translate_local`, `hom_mat2d_scale`, `hom_mat2d_scale_local`, `hom_mat2d_rotate`, `hom_mat2d_rotate_local`, `hom_mat2d_slant`, `hom_mat2d_slant_local`, `hom_mat2d_reflect`, `hom_mat2d_reflect_local`

Nachfolger

`hom_mat2d_compose`, `hom_mat2d_translate`, `hom_mat2d_translate_local`, `hom_mat2d_scale`, `hom_mat2d_scale_local`, `hom_mat2d_rotate`, `hom_mat2d_rotate_local`, `hom_mat2d_slant`, `hom_mat2d_slant_local`, `hom_mat2d_reflect`, `hom_mat2d_reflect_local`

Modul

Foundation

hom_mat2d_determinant (: : HomMat2D : Determinant)

Berechnet die Determinante einer homogenen 2D-Transformationsmatrix.

`hom_mat2d_determinant` berechnet die Determinante der homogenen 2D-Transformationsmatrix, die in `HomMat2D` übergeben wird und gibt die Determinante in `Determinant` zurück.

Parameter

▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
Eingabe-Transformationsmatrix.

▷ **Determinant** (output_control) ..... real ⇝ *real*
Determinante der Eingabematrix.

Ergebnis

`hom_mat2d_determinant` liefert immer den Wert TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat2d_translate`, `hom_mat2d_translate_local`, `hom_mat2d_scale`, `hom_mat2d_scale_local`, `hom_mat2d_rotate`, `hom_mat2d_rotate_local`, `hom_mat2d_slant`, `hom_mat2d_slant_local`, `hom_mat2d_reflect`, `hom_mat2d_reflect_local`

Modul

Foundation

hom_mat2d_identity (: : : HomMat2DIdentity)

Erzeugt die homogene Transformationsmatrix der identischen 2D-Abbildung.

hom_mat2d_identity erzeugt die homogene Transformationsmatrix HomMat2DIdentity, die die identische 2D-Abbildung beschreibt:

$$\texttt{HomMat2DIdentity} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Demnach wird die Matrix HomMat2DIdentity als das Tupel [1,0,0,0,1,0] abgespeichert.

Parameter

▷ **HomMat2DIdentity** (output_control) . . . . . hom_mat2d ⇝ *real*
Transformationsmatrix.

Ergebnis

hom_mat2d_identity liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

hom_mat2d_translate, hom_mat2d_translate_local, hom_mat2d_scale, hom_mat2d_scale_local, hom_mat2d_rotate, hom_mat2d_rotate_local, hom_mat2d_slant, hom_mat2d_slant_local, hom_mat2d_reflect, hom_mat2d_reflect_local

Modul

Foundation

hom_mat2d_invert (: : HomMat2D : HomMat2DInvert)

Invertiert eine homogene 2D-Transformationsmatrix.

hom_mat2d_invert invertiert die homogene 2D-Transformationsmatrix, die in HomMat2D übergeben wird. Die resultierende Matrix wird in HomMat2DInvert zurückgeliefert.

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
Eingabe-Transformationsmatrix.

▷ **HomMat2DInvert** (output_control) . . . . . hom_mat2d ⇝ *real*
Ausgabe-Transformationsmatrix.

Ergebnis

hom_mat2d_invert liefert den Wert TRUE, falls die Parameterwerte korrekt sind und die Eingabematrix invertierbar ist. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_translate, hom_mat2d_translate_local, hom_mat2d_scale, hom_mat2d_scale_local, hom_mat2d_rotate, hom_mat2d_rotate_local, hom_mat2d_slant, hom_mat2d_slant_local, hom_mat2d_reflect, hom_mat2d_reflect_local

Nachfolger

hom_mat2d_translate, hom_mat2d_translate_local, hom_mat2d_scale, hom_mat2d_scale_local, hom_mat2d_rotate, hom_mat2d_rotate_local, hom_mat2d_slant, hom_mat2d_slant_local, hom_mat2d_reflect, hom_mat2d_reflect_local

Modul

Foundation

hom_mat2d_reflect (: : HomMat2D, Px, Py, Qx, Qy : HomMat2DReflect)

Fügt eine Spiegelung zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_reflect fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Spiegelung um die Achse, die durch die zwei Punkte (Px,Py) und (Qx,Qy) gegeben wird, hinzu und liefert die resultierende Matrix in HomMat2DReflect zurück. Die Spiegelung wird durch eine 2×2 Spiegelungsmatrix $\mathbf{M}$ beschrieben. Sie wird relativ zum globalen (d.h. feststehenden) Koordinatensystem ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DReflect} = \begin{bmatrix} \mathbf{M} & \begin{matrix}0\\0\end{matrix} \\ \begin{matrix}0 & 0\end{matrix} & 1 \end{bmatrix} \cdot \texttt{HomMat2D} \qquad \mathbf{M} = \mathbf{I} - \frac{2}{v^T v} v v^T$$

Hierbei ist $v = (\texttt{Py} - \texttt{Qy}, \texttt{Qx} - \texttt{Px})^T$ der Normalenvektor zur Achse.

Die Achse $(\texttt{Px}, \texttt{Py}) - (\texttt{Qx}, \texttt{Qy})$ ist bei der Spiegelung fixiert, d.h. die Achse bleibt unverändert, wenn man sie mit HomMat2DReflect transformiert. Dieses Verhalten wird erreicht, indem zur Eingabe-Transformationsmatrix zuerst eine Translation hinzugefügt wird, die die Achse in den Ursprung des globalen Koordinatensystems verschiebt. Dann wird die Spiegelung hinzugefügt, und zuletzt eine Translation, die die Achse wieder zurück in ihre Ausgangsposition verschiebt. Dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DReflect} = \begin{bmatrix} 1 & 0 & +\texttt{Px} \\ 0 & 1 & +\texttt{Py} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{M} & \begin{matrix}0\\0\end{matrix} \\ \begin{matrix}0 & 0\end{matrix} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -\texttt{Px} \\ 0 & 1 & -\texttt{Py} \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D}$$

Um die Transformation im lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, durchzuführen, kann hom_mat2d_reflect_local verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Px** (input_control) . . . . . point.x ⇝ *real* / integer
  Erster Punkt auf der Achse (x-Koordinate).
  Defaultwert: 0
  Wertevorschläge: Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Py** (input_control) . . . . . point.y ⇝ *real* / integer
  Erster Punkt auf der Achse (y-Koordinate).
  Defaultwert: 0
  Wertevorschläge: Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Qx** (input_control) . . . . . point.x ⇝ *real* / integer
  Zweiter Punkt auf der Achse (x-Koordinate).
  Defaultwert: 16
  Wertevorschläge: Qx ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Qy** (input_control) . . . . . point.y ⇝ *real* / integer
  Zweiter Punkt auf der Achse (y-Koordinate).
  Defaultwert: 32
  Wertevorschläge: Qy ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat2DReflect** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

hom_mat2d_reflect liefert den Wert TRUE, falls beide Punkte auf der Achse verschieden sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant, hom_mat2d_reflect

Nachfolger

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant, hom_mat2d_reflect

Siehe auch

hom_mat2d_reflect_local

Modul

Foundation

hom_mat2d_reflect_local (: : HomMat2D, Px, Py : HomMat2DReflect)

Fügt eine Spiegelung zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_reflect_local fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Spiegelung um die Achse, die durch die zwei Punkte (0,0) und (Px,Py) gegeben wird, hinzu und liefert die resultierende Matrix in HomMat2DReflect zurück. Die Spiegelung wird durch eine 2×2 Spiegelungsmatrix $\mathbf{M}$ beschrieben.

Im Gegensatz zu hom_mat2d_reflect wird sie relativ zum lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DReflect} = \begin{bmatrix} \mathbf{M} & & 0 \\ & & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D} \qquad \mathbf{M} = \mathbf{I} - \frac{2}{v^T v} v v^T$$

Hierbei ist $v = (-\texttt{Py}, \texttt{Px})^T$ der Normalenvektor zur Achse.

Die Achse $(0,0) - (\texttt{Px}, \texttt{Py})$ ist bei der Spiegelung fixiert, d.h. die Achse bleibt unverändert, wenn man sie mit HomMat2DReflect transformiert.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Px** (input_control) ..... point.x ⇝ *real* / integer
  Punkt, der die Achse definiert (x-Koordinate).
  Defaultwert: 16
  Wertevorschläge: Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Py** (input_control) ..... point.y ⇝ *real* / integer
  Punkt, der die Achse definiert (y-Koordinate).
  Defaultwert: 32
  Wertevorschläge: Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat2DReflect** (output_control) ..... hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

hom_mat2d_reflect_local liefert den Wert TRUE, falls der Punkt (Px, Py) nicht $(0,0)$ ist. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate_local, hom_mat2d_scale_local, hom_mat2d_rotate_local, hom_mat2d_slant_local, hom_mat2d_reflect_local

Nachfolger

hom_mat2d_translate_local, hom_mat2d_scale_local, hom_mat2d_rotate_local, hom_mat2d_slant_local, hom_mat2d_reflect_local

Siehe auch

hom_mat2d_reflect

Modul

Foundation

hom_mat2d_rotate (: : HomMat2D, Phi, Px, Py : HomMat2DRotate)

Fügt eine Rotation zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_rotate fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Rotation um den Winkel Phi hinzu. Die Rotation wird beschrieben durch die 2×2 Rotationsmatrix $\mathbf{R}$. Sie wird relativ zum globalen (d.h. feststehenden) Koordinatensystem ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DRotate} = \begin{bmatrix} \mathbf{R} & & 0 \\ & & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D} \qquad \mathbf{R} = \begin{bmatrix} \cos(\texttt{Phi}) & -\sin(\texttt{Phi}) \\ \sin(\texttt{Phi}) & \cos(\texttt{Phi}) \end{bmatrix}$$

Der Punkt (Px,Py) ist dabei der Fixpunkt der Rotation, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat2DRotate transformiert. Dieses Verhalten wird erreicht, indem zur Eingabe-Transformationsmatrix zuerst eine Translation hinzugefügt wird, die den Fixpunkt in den Ursprung des globalen Koordinatensystems verschiebt. Dann wird die Rotation hinzugefügt, und zuletzt eine Translation, die den Fixpunkt wieder zurück in seine Ausgangsposition verschiebt. Dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DRotate} = \begin{bmatrix} 1 & 0 & +\texttt{Px} \\ 0 & 1 & +\texttt{Py} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & & 0 \\ & & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -\texttt{Px} \\ 0 & 1 & -\texttt{Py} \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D}$$

Um die Transformation im lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, durchzuführen, kann hom_mat2d_rotate_local verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
Eingabe-Transformationsmatrix.

▷ **Phi** (input_control) ..... angle.rad ⇝ *real* / integer
Rotationswinkel.
Defaultwert: 0.78
Wertevorschläge: Phi ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
Typischer Wertebereich: 0 ≤ Phi ≤ 6.28318530718

▷ **Px** (input_control) . . . . . point.x ⇝ *real* / integer
Fixpunkt der Transformation (x-Koordinate).
Defaultwert: 0
Wertevorschläge: Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}

▷ **Py** (input_control) . . . . . point.y ⇝ *real* / integer
Fixpunkt der Transformation (y-Koordinate).
Defaultwert: 0
Wertevorschläge: Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}

▷ **HomMat2DRotate** (output_control) . . . . . hom_mat2d ⇝ *real*
Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert hom_mat2d_rotate den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant, hom_mat2d_reflect

Nachfolger

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant, hom_mat2d_reflect

Siehe auch

hom_mat2d_rotate_local

Modul

Foundation

hom_mat2d_rotate_local (: : HomMat2D, Phi : HomMat2DRotate)

Fügt eine Rotation zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_rotate_local fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Rotation um den Winkel Phi hinzu. Die Rotation wird beschrieben durch die 2×2 Rotationsmatrix $\mathbf{R}$. Im Gegensatz zu hom_mat2d_rotate wird sie relativ zum lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DRotate} = \texttt{HomMat2D} \cdot \begin{bmatrix} \mathbf{R} & & 0 \\ & & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} \cos(\texttt{Phi}) & -\sin(\texttt{Phi}) \\ \sin(\texttt{Phi}) & \cos(\texttt{Phi}) \end{bmatrix}$$

Der Fixpunkt der Rotation ist der Ursprung des lokalen Koordinatensystems, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat2DRotate transformiert.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Phi** (input_control) . . . . . angle.rad ⇝ *real* / integer
  Rotationswinkel.
  Defaultwert: 0.78
  Wertevorschläge: Phi ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
  Typischer Wertebereich: 0 ≤ Phi ≤ 6.28318530718
- ▷ **HomMat2DRotate** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert hom_mat2d_rotate_local den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate_local, hom_mat2d_scale_local, hom_mat2d_rotate_local, hom_mat2d_slant_local, hom_mat2d_reflect_local

Nachfolger

hom_mat2d_translate_local, hom_mat2d_scale_local, hom_mat2d_rotate_local, hom_mat2d_slant_local, hom_mat2d_reflect_local

Siehe auch

hom_mat2d_rotate

Modul

Foundation

hom_mat2d_scale (: : HomMat2D, Sx, Sy, Px, Py : HomMat2DScale)

Fügt eine Skalierung zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_scale fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Skalierung um die Skalierungsfaktoren Sx und Sy hinzu und liefert die resultierende Matrix in HomMat2DScale zurück. Die Skalierung wird durch eine 2×2 Skalierungsmatrix **S** beschrieben. Sie wird relativ zum globalen (d.h. feststehenden) Koordinatensystem ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\text{HomMat2DScale} = \begin{bmatrix} & \mathbf{S} & \\ & & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \text{HomMat2D} \qquad \mathbf{S} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix}$$

Der Punkt (Px,Py) ist dabei der Fixpunkt der Skalierung, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat2DScale transformiert. Dieses Verhalten wird erreicht, indem zur Eingabe-Transformationsmatrix

zuerst eine Translation hinzugefügt wird, die den Fixpunkt in den Ursprung des globalen Koordinatensystems verschiebt. Dann wird die Skalierung hinzugefügt, und zuletzt eine Translation, die den Fixpunkt wieder zurück in seine Ausgangsposition verschiebt. Dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DScale} = \begin{bmatrix} 1 & 0 & +\texttt{Px} \\ 0 & 1 & +\texttt{Py} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{S} & & 0 \\ & & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -\texttt{Px} \\ 0 & 1 & -\texttt{Py} \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D}$$

Um die Transformation im lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, durchzuführen, kann hom_mat2d_scale_local verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Sx** (input_control) . . . . . number ⇝ *real* / integer
  Skalierungsfaktor in x-Richtung.
  Defaultwert: 2
  Wertevorschläge: Sx ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 16}
  Restriktion: Sx != 0
- ▷ **Sy** (input_control) . . . . . number ⇝ *real* / integer
  Skalierungsfaktor in y-Richtung.
  Defaultwert: 2
  Wertevorschläge: Sy ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 16}
  Restriktion: Sy != 0
- ▷ **Px** (input_control) . . . . . point.x ⇝ *real* / integer
  Fixpunkt der Transformation (x-Koordinate).
  Defaultwert: 0
  Wertevorschläge: Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Py** (input_control) . . . . . point.y ⇝ *real* / integer
  Fixpunkt der Transformation (y-Koordinate).
  Defaultwert: 0
  Wertevorschläge: Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat2DScale** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

hom_mat2d_scale liefert den Wert TRUE, falls beide Skalierungsfaktoren verschieden von 0 sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant, hom_mat2d_reflect

Nachfolger

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant, hom_mat2d_reflect

Siehe auch

hom_mat2d_scale_local

Modul

Foundation

hom_mat2d_scale_local (: : HomMat2D, Sx, Sy : HomMat2DScale)

Fügt eine Skalierung zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_scale_local fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Skalierung um die Skalierungsfaktoren Sx und Sy hinzu und liefert die resultierende Matrix in HomMat2DScale zurück. Die Skalierung wird durch eine 2×2 Skalierungsmatrix **S** beschrieben. Im Gegensatz zu hom_mat2d_scale wird sie relativ zum lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DScale} = \texttt{HomMat2D} \cdot \begin{bmatrix} & \mathbf{S} & & 0 \\ & & & 0 \\ 0 & & 0 & 1 \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} \texttt{Sx} & 0 \\ 0 & \texttt{Sy} \end{bmatrix}$$

Der Fixpunkt der Skalierung ist der Ursprung des lokalen Koordinatensystems, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat2DScale transformiert.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
Eingabe-Transformationsmatrix.

▷ **Sx** (input_control) . . . . . number ⇝ *real* / integer
Skalierungsfaktor in x-Richtung.
Defaultwert: 2
Wertevorschläge: Sx ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 16}
Restriktion: Sx != 0

▷ **Sy** (input_control) . . . . . number ⇝ *real* / integer
Skalierungsfaktor in y-Richtung.
Defaultwert: 2
Wertevorschläge: Sy ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 16}
Restriktion: Sy != 0

▷ **HomMat2DScale** (output_control) . . . . . hom_mat2d ⇝ *real*
Ausgabe-Transformationsmatrix.

Ergebnis

hom_mat2d_scale_local liefert den Wert TRUE, falls beide Skalierungsfaktoren verschieden von 0 sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate_local, hom_mat2d_scale_local, hom_mat2d_rotate_local, hom_mat2d_slant_local, hom_mat2d_reflect_local

Nachfolger

hom_mat2d_translate_local, hom_mat2d_scale_local, hom_mat2d_rotate_local, hom_mat2d_slant_local, hom_mat2d_reflect_local

Siehe auch

hom_mat2d_scale

Modul

Foundation

hom_mat2d_slant (: : HomMat2D, Theta, Axis, Px, Py : HomMat2DSlant)

Fügt eine Scherung zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_slant fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Scherung um den Winkel Theta hinzu und liefert die resultierende Matrix in HomMat2DSlant zurück. Eine Scherung ist eine affine Transformation, bei der eine Koordinatenachse festgehalten wird, während die andere Koordinatenachse um den Winkel Theta im Gegenuhrzeigersinn gedreht wird. Der Parameter Axis bestimmt, für welche Koordinatenachse die Scherung ausgeführt werden soll. Für Axis = *'x'* wird die x-Achse geschert und die y-Achse festgehalten, für Axis = *'y'* wird die y-Achse geschert und die x-Achse festgehalten. Die Scherung wird relativ zum globalen (d.h. feststehenden) Koordinatensystem ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{Axis} = 'x': \quad \texttt{HomMat2DSlant} = \begin{bmatrix} \cos(\texttt{Theta}) & 0 & 0 \\ \sin(\texttt{Theta}) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D}$$

$$\texttt{Axis} = 'y': \quad \texttt{HomMat2DSlant} = \begin{bmatrix} 1 & -\sin(\texttt{Theta}) & 0 \\ 0 & \cos(\texttt{Theta}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D}$$

Der Punkt (Px,Py) ist dabei der Fixpunkt der Scherung, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat2DSlant transformiert. Dieses Verhalten wird erreicht, indem zur Eingabe-Transformationsmatrix zuerst eine Translation hinzugefügt wird, die den Fixpunkt in den Ursprung des globalen Koordinatensystems verschiebt. Dann wird die Scherung hinzugefügt, und zuletzt eine Translation, die den Fixpunkt wieder zurück in seine Ausgangsposition verschiebt. Dies entspricht der folgenden Kette von Transformationsmatrizen für Axis = *'x'*:

$$\texttt{HomMat2DSlant} = \begin{bmatrix} 1 & 0 & +\texttt{Px} \\ 0 & 1 & +\texttt{Py} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\texttt{Theta}) & 0 & 0 \\ \sin(\texttt{Theta}) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -\texttt{Px} \\ 0 & 1 & -\texttt{Py} \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D}$$

Um die Transformation im lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, durchzuführen, kann hom_mat2d_slant_local verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Theta** (input_control) ..... angle.rad ⇝ *real* / integer
  Winkel der Scherung.
  Defaultwert: 0.78
  Wertevorschläge: Theta ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
  Typischer Wertebereich: 0 ≤ Theta ≤ 6.28318530718
- ▷ **Axis** (input_control) ..... string ⇝ *string*
  Koordinatenachse, die geschert wird.
  Defaultwert: 'x'
  Werteliste: Axis ∈ {'x', 'y'}
- ▷ **Px** (input_control) ..... point.x ⇝ *real* / integer
  Fixpunkt der Transformation (x-Koordinate).
  Defaultwert: 0
  Wertevorschläge: Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Py** (input_control) ..... point.y ⇝ *real* / integer
  Fixpunkt der Transformation (y-Koordinate).
  Defaultwert: 0
  Wertevorschläge: Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat2DSlant** (output_control) ..... hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert hom_mat2d_slant den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant, hom_mat2d_reflect

Nachfolger

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant, hom_mat2d_reflect

Siehe auch

hom_mat2d_slant_local

Modul

Foundation

hom_mat2d_slant_local (: : HomMat2D, Theta, Axis : HomMat2DSlant)

Fügt eine Scherung zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_slant_local fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Scherung um den Winkel Theta hinzu und liefert die resultierende Matrix in HomMat2DSlant zurück. Eine Scherung ist eine affine Transformation, bei der eine Koordinatenachse festgehalten wird, während die andere Koordinatenachse um den Winkel Theta im Gegenuhrzeigersinn gedreht wird. Der Parameter Axis bestimmt, für welche Koordinatenachse die Scherung ausgeführt werden soll. Für Axis = *'x'* wird die x-Achse geschert und die y-Achse festgehalten, für Axis = *'y'* wird die y-Achse geschert und die x-Achse festgehalten. Im Gegensatz zu hom_mat2d_slant wird die Scherung relativ zum lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{Axis} = 'x': \quad \texttt{HomMat2DSlant} = \texttt{HomMat2D} \cdot \begin{bmatrix} \cos(\texttt{Theta}) & 0 & 0 \\ \sin(\texttt{Theta}) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\texttt{Axis} = 'y': \quad \texttt{HomMat2DSlant} = \texttt{HomMat2D} \cdot \begin{bmatrix} 1 & -\sin(\texttt{Theta}) & 0 \\ 0 & \cos(\texttt{Theta}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Der Fixpunkt der Scherung ist der Ursprung des lokalen Koordinatensystems, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat2DSlant transformiert.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
Eingabe-Transformationsmatrix.

▷ **Theta** (input_control) . . . . . angle.rad ⇝ *real* / integer
Winkel der Scherung.
Defaultwert: 0.78
Wertevorschläge: `Theta` ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
Typischer Wertebereich: $0 \leq$ `Theta` ≤ 6.28318530718

▷ **Axis** (input_control) . . . . . string ⇝ *string*
Koordinatenachse, die geschert wird.
Defaultwert: 'x'
Werteliste: `Axis` ∈ {'x', 'y'}

▷ **HomMat2DSlant** (output_control) . . . . . hom_mat2d ⇝ *real*
Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `hom_mat2d_slant_local` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat2d_identity`, `hom_mat2d_translate_local`, `hom_mat2d_scale_local`, `hom_mat2d_rotate_local`, `hom_mat2d_slant_local`, `hom_mat2d_reflect`

Nachfolger

`hom_mat2d_translate_local`, `hom_mat2d_scale_local`, `hom_mat2d_rotate_local`, `hom_mat2d_slant_local`, `hom_mat2d_reflect_local`

Siehe auch

`hom_mat2d_slant`

Modul

Foundation

hom_mat2d_to_affine_par (: : HomMat2D : Sx, Sy, Phi, Theta, Tx, Ty)

Berechnet die affinen Transformationsparameter aus einer homogenen 2D-Transformationsmatrix.

`hom_mat2d_to_affine_par` berechnet aus einer homogenen 2D-Transformationsmatrix `HomMat2D` die zugehörigen affinen Transformationsparameter. Die Parameter `Sx` und `Sy` geben an, wie stark die ursprünglichen x- und y-Achsen von der Transformation skaliert werden. Die beiden Skalierungsfaktoren sind immer positiv. Der Winkel `Theta` beschreibt, ob die transformierten Koordinatenachsen orthogonal sind (`Theta` = 0) oder ob die y-Achse geschert ist. Falls $|$`Theta`$| > \pi/2$, enthält die Transformation eine Spiegelung. Der Winkel `Phi` beschreibt die Rotation der x-Achse des transformierten Koordinatensystems gegenüber der x-Achse des ursprünglichen Koordinatensystems. Die Parameter `Tx` und `Ty` beschreiben die Verschiebung der beiden Koordinatensysteme. Die Matrix `HomMat2D` lässt sich mit folgender Operatorsequenz aus den sechs Transformationsparametern erzeugen:

```
hom_mat2d_identity (HomMat2DIdentity)
hom_mat2d_scale (HomMat2DIdentity, Sx, Sy, 0, 0, HomMat2DScale)
hom_mat2d_slant (HomMat2DScale, Theta, 'y', 0, 0, HomMat2DSlant)
hom_mat2d_rotate (HomMat2DSlant, Phi, 0, 0, HomMat2DRotate)
hom_mat2d_translate (HomMat2DRotate, Tx, Ty, HomMat2D)
```

Dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2D} = \begin{bmatrix} 1 & 0 & +\texttt{Tx} \\ 0 & 1 & +\texttt{Ty} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\texttt{Phi}) & -\sin(\texttt{Phi}) & 0 \\ \sin(\texttt{Phi}) & \cos(\texttt{Phi}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -\sin(\texttt{Theta}) & 0 \\ 0 & \cos(\texttt{Theta}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \texttt{Sx} & 0 & 0 \\ 0 & \texttt{Sy} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Sx** (output_control) . . . . . real ⇝ *real*
  Skalierung in x-Richtung.
- ▷ **Sy** (output_control) . . . . . real ⇝ *real*
  Skalierung in y-Richtung.
- ▷ **Phi** (output_control) . . . . . angle.rad ⇝ *real*
  Rotationswinkel.
- ▷ **Theta** (output_control) . . . . . angle.rad ⇝ *real*
  Scherungswinkel.
- ▷ **Tx** (output_control) . . . . . point.x ⇝ *real*
  Translation in x-Richtung.
- ▷ **Ty** (output_control) . . . . . point.y ⇝ *real*
  Translation in y-Richtung.

Ergebnis

Falls die Matrix HomMat2D nicht degeneriert ist und eine affine Abbildung darstellt (d.h. keine projektive Abbildung), liefert hom_mat2d_to_affine_par den Wert TRUE zurück. Ansonsten wird eine Ausnahmebehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

vector_to_hom_mat2d, vector_to_rigid, vector_to_similarity, vector_to_aniso, point_line_to_hom_mat2d

Nachfolger

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Modul

Foundation

hom_mat2d_translate (: : HomMat2D, Tx, Ty : HomMat2DTranslate)

Fügt eine Translation zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_translate fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Translation um den Vektor $\mathbf{T}$ = (Tx,Ty) hinzu und liefert die resultierende Matrix in HomMat2DTranslate zurück. Die Translation wird relativ zum globalen (d.h. feststehenden) Koordinatensystem ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DTranslate} = \begin{bmatrix} 1 & 0 & \\ 0 & 1 & \mathbf{T} \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D} \qquad \mathbf{T} = \begin{pmatrix} \texttt{Tx} \\ \texttt{Ty} \end{pmatrix}$$

Um die Transformation im lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, durchzuführen, kann hom_mat2d_translate_local verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (`Row`,`Column`). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **`HomMat2D`** (input_control) ..... hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **`Tx`** (input_control) ..... point.x ⇝ *real* / integer
  Verschiebung in x-Richtung.
  Defaultwert: 64
  Wertevorschläge: `Tx` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **`Ty`** (input_control) ..... point.y ⇝ *real* / integer
  Verschiebung in y-Richtung.
  Defaultwert: 64
  Wertevorschläge: `Ty` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **`HomMat2DTranslate`** (output_control) ..... hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `hom_mat2d_translate` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat2d_identity`, `hom_mat2d_translate`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_slant`, `hom_mat2d_reflect`

Nachfolger

`hom_mat2d_translate`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_slant`, `hom_mat2d_reflect`

Siehe auch

`hom_mat2d_translate_local`

Modul

Foundation

`hom_mat2d_translate_local` `( : : HomMat2D, Tx, Ty : HomMat2DTranslate )`

Fügt eine Translation zu einer homogenen 2D-Transformationsmatrix hinzu.

hom_mat2d_translate_local fügt zur homogenen 2D-Transformationsmatrix HomMat2D eine Translation um den Vektor **T** = (Tx,Ty) hinzu und liefert die resultierende Matrix in HomMat2DTranslate zurück. Im Gegensatz zu hom_mat2d_translate wird die Translation relativ zum lokalen Koordinatensystem, das durch HomMat2D beschrieben ist, ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat2DTranslate} = \texttt{HomMat2D} \cdot \begin{bmatrix} 1 & 0 & \\ 0 & 1 & \mathbf{T} \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T} = \begin{pmatrix} \texttt{Tx} \\ \texttt{Ty} \end{pmatrix}$$

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, tc, rd, re, tf] gespeichert. Es ist aber auch möglich, volle 3×3 Matrizen, die eine projektive 2D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Tx** (input_control) . . . . . point.x ⇝ *real* / integer
  Verschiebung in x-Richtung.
  Defaultwert: 64
  Wertevorschläge: Tx ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Ty** (input_control) . . . . . point.y ⇝ *real* / integer
  Verschiebung in y-Richtung.
  Defaultwert: 64
  Wertevorschläge: Ty ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat2DTranslate** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert hom_mat2d_translate_local den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate_local, hom_mat2d_scale_local, hom_mat2d_rotate_local, hom_mat2d_slant_local, hom_mat2d_reflect_local

Nachfolger

hom_mat2d_translate_local, hom_mat2d_scale_local, hom_mat2d_rotate_local, hom_mat2d_slant_local, hom_mat2d_reflect_local

Siehe auch

hom_mat2d_translate

Modul

Foundation

hom_mat2d_transpose (: : HomMat2D : HomMat2DTranspose)

Transponiert eine homogene 2D-Transformationsmatrix.

hom_mat2d_transpose transponiert die homogene 2D-Transformationsmatrix, die in HomMat2D übergeben wird. Die Ergebnismatrix HomMat2DTranspose ist immer eine 3×3 Matrix, selbst wenn die Eingabematrix als 2×3 Matrix dargestellt ist.

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **HomMat2DTranspose** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

hom_mat2d_transpose liefert immer den Wert TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat2d_translate, hom_mat2d_translate_local, hom_mat2d_scale, hom_mat2d_scale_local, hom_mat2d_rotate, hom_mat2d_rotate_local, hom_mat2d_slant, hom_mat2d_slant_local, hom_mat2d_reflect, hom_mat2d_reflect_local

Nachfolger

hom_mat2d_compose, hom_mat2d_invert

Modul

Foundation

hom_mat3d_project (: : HomMat3D, PrincipalPointRow, PrincipalPointCol, Focus : HomMat2D)

Projiziert eine Matrix zur affinen 3D-Transformation in eine 2D projektive Transformationsmatrix.

hom_mat3d_project berechnet aus der homogenen 3×4 Transformationsmatrix HomMat3D, die eine affine Transformation im 3D beschreibt, eine homogene projektive Transformationsmatrix, die dazu verwendet werden kann, eine Ebene, insbesondere eine Ebene, die ein Bild enthält, zu projizieren. Die projektive Transformationsmatrix beschreibt eine projektive Transformation zwischen zwei (zweidimensionalen) Ebenen.

Dies kann zur Erzeugung von perspektivischen Verzerrungen genutzt werden, welche durch die Projektion einer Ebene entstehen, die um eine andere Achse als die z-Achse gedreht ist. Im Normalfall werden projektive Transformationen aber aus Punktkorrespondenzen ermittelt (siehe vector_to_proj_hom_mat2d, hom_vector_to_proj_hom_mat2d und proj_match_points_ransac).

Matrizen zur Drehung, Skalierung oder Verschiebung können mit Hilfe der Operatoren hom_mat3d_identity, hom_mat3d_scale, hom_mat3d_rotate, hom_mat3d_translate und pose_to_hom_mat3d konstruiert werden.

Dabei ist zu beachten, dass bei den 3D-Transformationen die x-Achse der Spaltenachse und die y-Achse der Zeilenachse entspricht (siehe auch Kalibrierung), während bei `projective_trans_image` in `HomMat2D` die erste Zeile der Zeilenachse und die zweite Zeile der Transformation der Spaltenachse des Bildes entspricht.

Der Punkt (`PrincipalPointRow`, `PrincipalPointCol`) ist dabei das optische Zentrum der Projektion und der Punkt (`PrincipalPointRow`, `PrincipalPointCol`, *0*) im Dreidimensionalen somit der virtuelle Standpunkt der Kamera. Die Blickrichtung entspricht der positiven z-Achse.

In diesem virtuellen Raum befindet sich die Ebene des Eingabebildes, ebenso wie die Bildebene der Projektion, an der Stelle $z =$ `Focus`, also `Focus` Pixel von der Kamera entfernt. Dies führt dazu, dass die Verwendung der identischen Abbildung als Eingabematrix `HomMat3D` eine Matrix `HomMat2D` ergibt, die ebenfalls die identische Abbildung im Zweidimensionalen darstellt.

Der Parameter `Focus` ist also die „Brennweite“ der verwendeten virtuellen Kamera und in der Einheit Pixel angegeben. Durch ihn kann der Grad der perspektivischen Verzerrungen verändert werden. Dieselbe Eingabematrix `HomMat3D` führt bei großer Brennweite zu schwächeren Verzerrungen als bei kleiner Brennweite.

Sei H die affine 3D-Matrix mit Elementen h_{ij}, $(r, c) =$ (`PrincipalPointRow`, `PrincipalPointCol`)

und $f =$ `Focus`.

Dann wird die projektive Transformationsmatrix wie folgt berechnet: Zunächst wird eine 3×4 Projektionsmatrix berechnet als:

$$Q = \begin{pmatrix} f & 0 & c \\ 0 & f & r \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -c \\ 0 & 1 & 0 & -r \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Da das Bild einer Ebene, in der die Punkte die Form $(x, y, f, 1)^T$ haben, berechnet werden soll, werden die letzten zwei Spalten von Q zusammengefasst:

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & f \cdot q_{13} + q_{14} \\ q_{21} & q_{22} & f \cdot r_{23} + q_{24} \\ q_{31} & q_{32} & f \cdot r_{33} + q_{34} \end{pmatrix} = Q \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f \\ 0 & 0 & 1 \end{pmatrix}$$

Zuletzt werden die Spalten und Zeilen von R so vertauscht, dass die erste Zeile von P der Transformation der Zeilenkoordinaten und die zweite Zeile von P der Transformation der Spaltenkoordinaten entspricht, so dass P direkt in `projective_trans_image` verwendet werden kann:

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot R \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Insgesamt ergibt sich also:

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} f & 0 & c \\ 0 & f & r \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -c \\ 0 & 1 & 0 & -r \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot H \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Parameter

▷ **`HomMat3D`** (input_control) ..... hom_mat3d ⇝ *real*
3 × 4 3D-Transformationsmatrix.

▷ **`PrincipalPointRow`** (input_control) ..... point.y ⇝ *real* / integer
Zeilenkoordinate des Hauptpunktes.
Defaultwert: 256
Wertevorschläge: `PrincipalPointRow` ∈ {16, 32, 64, 128, 240, 256, 512}

▷ **`PrincipalPointCol`** (input_control) ..... point.x ⇝ *real* / integer
Spaltenkoordinate des Hauptpunktes.
Defaultwert: 256
Wertevorschläge: `PrincipalPointCol` ∈ {16, 32, 64, 128, 256, 320, 512}

▷ **Focus** (input_control) ..... number ⇝ *real* / integer
Brennweite in Pixeln.
Defaultwert: 256
Wertevorschläge: Focus ∈ {1, 2, 5, 256, 32768}

▷ **HomMat2D** (output_control) ..... hom_mat2d ⇝ *real*
Homogene projektive Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat3d_identity, hom_mat3d_rotate, hom_mat3d_translate, hom_mat3d_scale

Nachfolger

projective_trans_image, projective_trans_point_2d, projective_trans_region, projective_trans_contour_xld, hom_mat2d_invert

Modul

Foundation

hom_vector_to_proj_hom_mat2d (: : Px, Py, Pw, Qx, Qy, Qw, Method : HomMat2D)

Bestimmt eine projektive Transformationsmatrix unter Verwendung vorgegebener Punktkorrespondenzen.

hom_vector_to_proj_hom_mat2d ermittelt aus mindestens 4 vorgegebenen Punktkorrespondenzen

$$\mathtt{HomMat2D} \cdot \begin{pmatrix} \mathtt{Px} \\ \mathtt{Py} \\ \mathtt{Pw} \end{pmatrix} = \begin{pmatrix} \mathtt{Qx} \\ \mathtt{Qy} \\ \mathtt{Qw} \end{pmatrix}$$

die homogene projektive Transformationsmatrix HomMat2D, die diese Gleichungen am besten erfüllt. Sind weniger als 4 Punktpaare (Px, Py, Pw), (Qx, Qy, Qw) angegeben, so ist eine solche Transformation nicht eindeutig bestimmbar. Sind genau 4 Punktpaare gegeben, so kann eine Matrix gefunden werden, die die Punkte genau ineinander überführt. Andernfalls wird die Matrix so gewählt, dass der Fehler möglichst klein wird. Um dieses Ergebnis zu erreichen, sind verschiedene Algorithmen verfügbar. Welches Verfahren verwendet werden soll, kann mittels des Parameters Method gesteuert werden. Für konventionelle geometrische Transformationen liefert die Einstellung Method=*'normalized_dlt'* im allgemeinen die besseren Ergebnisse. Ist aber eines der Qw oder Pw gleich *0*, so muss Method=*'dlt'* gewählt werden.

Im Gegensatz zu vector_to_proj_hom_mat2d können aufgrund der Verwendung von homogenen Koordinaten mit hom_vector_to_proj_hom_mat2d unendlich ferne Punkte (Pw = *0* oder Qw = *0*) zur Bestimmung der Transformation verwendet werden. Falls endliche Punkte verwendet werden, wird im Normalfall Pw = *1* und Qw = *1* gewählt. In diesem Fall kann auch vector_to_proj_hom_mat2d verwendet werden, wobei dort zusätzlich noch eine weitere Optimierungsmethode zur Verfügung steht und die Kovarianzen der Punkte berücksichtigt werden können. Falls die Korrespondenz zwischen den Punkten nicht bestimmt worden ist, sollte proj_match_points_ransac zur Bestimmung der Korrespondenz sowie der Transformation verwendet werden.

Falls die zu transformierenden Punkte in Standard-Bildkoordinaten vorliegen, müssen die *Zeilen*-Koordinaten der Punkte in Px und die *Spalten*-Koordinaten in Py übergeben werden. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern,

Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (`Row`,`Column`). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Weiterhin ist zu beachten, dass, falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, angenommen wird, dass der Ursprung des Koordinatensystems der homogenen Transformationsmatrix in der linken oberen Ecke des Pixels liegt. Die Bildverarbeitungsoperatoren, die Punktkoordinaten zurückliefern, nehmen hingegen ein Koordinatensystem an, in dem der Ursprung in der Mitte eines Pixels liegt. Daher muss, um eine konsistente homogene Transformationsmatrix zu erhalten, 0.5 zu den Punktkoordinaten addiert werden, bevor die Transformation berechnet wird.

Parameter

- ▷ **`Px`** (input_control) ..... number-array ⇝ *real* / integer
  Eingabepunkte 1 (x-Koordinate).
- ▷ **`Py`** (input_control) ..... number-array ⇝ *real* / integer
  Eingabepunkte 1 (y-Koordinate).
- ▷ **`Pw`** (input_control) ..... number-array ⇝ *real* / integer
  Eingabepunkte 1 (w-Koordinate).
- ▷ **`Qx`** (input_control) ..... number-array ⇝ *real*
  Eingabepunkte 2 (x-Koordinate).
- ▷ **`Qy`** (input_control) ..... number-array ⇝ *real*
  Eingabepunkte 2 (y-Koordinate).
- ▷ **`Qw`** (input_control) ..... number-array ⇝ *real*
  Eingabepunkte 2 (w-Koordinate).
- ▷ **`Method`** (input_control) ..... string ⇝ *string*
  Schätzalgorithmus.
  Defaultwert: 'normalized_dlt'
  Werteliste: `Method` ∈ {'normalized_dlt', 'dlt'}
- ▷ **`HomMat2D`** (output_control) ..... hom_mat2d ⇝ *real*
  Homogene projektive Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`proj_match_points_ransac`, `proj_match_points_ransac_guided`, `points_foerstner`, `points_harris`

Nachfolger

`projective_trans_image`, `projective_trans_image_size`, `projective_trans_region`, `projective_trans_contour_xld`, `projective_trans_point_2d`, `projective_trans_pixel`

Alternativen

`vector_to_proj_hom_mat2d`, `proj_match_points_ransac`, `proj_match_points_ransac_guided`

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2000.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Calibration

point_line_to_hom_mat2d (: : TransformationType, Px, Py, L1x, L1y, L2x, L2y : HomMat2D)

Erzeugt eine Näherung für eine affine Transformation aus Punkt-zu-Gerade-Korrespondenzen.

point_line_to_hom_mat2d berechnet eine Näherung für eine affine Transformation aus Punkt-zu-Gerade-Korrespondenzen und liefert sie in der homogenen Transformationsmatrix HomMat2D zurück (siehe hom_mat2d_to_affine_par für den Aufbau der homogenen Transformationsmatrix).

Die Punkte werden in den Tupeln (Px,Py) übergeben. Ihre korrespondierenden Geraden werden durch zwei Punkte auf der Geraden spezifiziert, die in (L1x,L1y) und (L2x,L2y) übergeben werden. Korrespondierende Punkte und Geraden müssen an denselben Indexpositionen in diesen Tupeln stehen.

Der Typ der zu bestimmenden Transformation wird mit TransformationType festgelegt. Jeder Transformationstyp erfordert eine bestimmte Mindestanzahl von Punkt-zu-Gerade-Korrespondenzen, wie in der folgenden Tabelle beschrieben:

TransformationType	Typ der Transformation	Mindestanzahl von Korrespondenzen
'translation'	Translation	2
'rigid'	Starre affine Transformation	3
'similarity'	Ähnlichkeitstransformation	4
'aniso'	Anisotrope Ähnlichkeitstransformation	5
'affine'	Allgemeine affine Transformation	6

Die unterstützten Transformationstypen können wie folgt beschrieben werden:

'translation': Eine Translation, d.h., eine Transformation, die wie folgt erzeugt werden kann:

```
hom_mat2d_identity (HomMat2D)
hom_mat2d_translate (HomMat2D, Tx, Ty, HomMat2D)
```

Dies bedeutet, dass

$$\text{HomMat2D} = \begin{bmatrix} 1 & 0 & \\ 0 & 1 & \mathbf{T} \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T})$$

wobei $\mathbf{T}$ der Translationvektor ist.

'rigid': Eine starre affine Transformation, d.h., eine Transformation, die wie folgt erzeugt werden kann:

```
hom_mat2d_identity (HomMat2D)
hom_mat2d_rotate (HomMat2D, Phi, 0, 0, HomMat2D)
hom_mat2d_translate (HomMat2D, Tx, Ty, HomMat2D)
```

Dies bedeutet, dass

$$\text{HomMat2D} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0\,0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \\ 0 & 1 & \mathbf{T} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & 0 \\ 0\,0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T}) \cdot \mathbf{H}(\mathbf{R})$$

wobei $\mathbf{R}$ eine zum Rotationswinkel Phi gehörige Rotationsmatrix und $\mathbf{T}$ der Translationvektor ist.

'similarity': Eine Ähnlichkeitstransformation, d.h., eine Transformation, die wie folgt erzeugt werden kann (beachten Sie die identischen Skalierungsfaktoren bei hom_mat2d_scale):

```
hom_mat2d_identity (HomMat2D)
hom_mat2d_scale (HomMat2D, S, S, 0, 0, HomMat2D)
hom_mat2d_rotate (HomMat2D, Phi, 0, 0, HomMat2D)
hom_mat2d_translate (HomMat2D, Tx, Ty, HomMat2D)
```

Dies bedeutet, dass

$$\text{HomMat2D} = \begin{bmatrix} \mathbf{R}\cdot\mathbf{S} & \mathbf{T} \\ 0\;\;0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \\ 0 & 1 & \mathbf{T} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & 0 \\ 0\,0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{S} & 0 \\ 0\,0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T}) \cdot \mathbf{H}(\mathbf{R}) \cdot \mathbf{H}(\mathbf{S})$$

wobei $\mathbf{S}$ eine Skalierungsmatrix mit identischer Skalierung S in x- und y-Richtung, $\mathbf{R}$ eine zum Rotationswinkel Phi gehörige Rotationsmatrix und $\mathbf{T}$ der Translationvektor ist.

'aniso': Eine anisotrope Ähnlichkeitstransformation, d.h., eine Transformation, die wie folgt erzeugt werden kann (beachten Sie die unterschiedlichen Skalierungsfaktoren bei `hom_mat2d_scale`):

```
hom_mat2d_identity (HomMat2D)
hom_mat2d_scale (HomMat2D, Sx, Sy, 0, 0, HomMat2D)
hom_mat2d_rotate (HomMat2D, Phi, 0, 0, HomMat2D)
hom_mat2d_translate (HomMat2D, Tx, Ty, HomMat2D)
```

Dies bedeutet, dass

$$\texttt{HomMat2D} = \begin{bmatrix} \mathbf{R}\cdot\mathbf{S} & \mathbf{T} \\ 0\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{T} \\ 0 & 1 & \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & & 0 \\ & & 0 \\ 0\,0 & & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T})\cdot\mathbf{H}(\mathbf{R})\cdot\mathbf{H}(\mathbf{S})$$

wobei $\mathbf{S}$ eine Skalierungsmatrix mit nicht-identischer Skalierung `Sx` und `Sy` in x- und y-Richtung, $\mathbf{R}$ eine zum Rotationswinkel `Phi` gehörige Rotationsmatrix und $\mathbf{T}$ der Translationvektor ist.

'affine': Eine allgemeine affine Transformation, d.h., eine Transformation, die wie folgt erzeugt werden kann:

```
hom_mat2d_identity (HomMat2D)
hom_mat2d_scale (HomMat2D, Sx, Sy, 0, 0, HomMat2D)
hom_mat2d_slant (HomMat2D, Theta, 'y', 0, 0, HomMat2D)
hom_mat2d_rotate (HomMat2D, Phi, 0, 0, HomMat2D)
hom_mat2d_translate (HomMat2D, Tx, Ty, HomMat2D)
```

Dies bedeutet, dass

$$\texttt{HomMat2D} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ 0 & 0 & 1 \end{bmatrix}$$

wobei $h_{1,1}$–$h_{2,3}$ beliebige Zahlen sind.

Die Transformation wird berechnet, indem die Summe der quadrierten Abstände der mit der berechneten Transformation transformierten Punkte (`Px`,`Py`) zu den Geraden, die durch (`L1x`,`L1y`) und (`L2x`,`L2y`) gegeben sind, minimiert werden. Die Geraden werden dabei als unendlich lange Geraden betrachtet. Das bedeutet, dass die mit der Transformation `HomMat2D` transformierten Punkte (`Px`,`Py`) nicht „zwischen“ (`L1x`,`L1y`) und (`L2x`,`L2y`) liegen müssen. Eine Ausnahme dieser Regel tritt auf, falls starre affine Transformationen oder anisotrope Ähnlichkeitstransformation aus der minimalen Anzahl von 3 bzw. 5 Punkt-zu-Gerade-Korrespondenzen bestimmt werden. In diesem Fall existieren im allgemeinen zwei mögliche Lösungen, die beide einen Fehler von 0 aufweisen. Der Algorithmus liefert die Lösung zurück, für die die transformierten Punkte so nahe wie möglich an dem *Geradenstück* (der *Strecke*) liegen, die durch (`L1x`,`L1y`) und (`L2x`,`L2y`) gegeben ist. Falls eine eindeutige Lösung gewünscht ist, sollte eine zusätzliche Punkt-zu-Gerade-Korrespondenz verwendet werden (d.h. 4 bzw. 6).

`HomMat2D` kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. `affine_trans_image`, verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (`Row`,`Column`). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Weiterhin ist zu beachten, dass, falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, angenommen wird, dass der Ursprung des Koordinatensystems der homogenen Transformationsmatrix in der linken oberen Ecke des Pixels liegt. Die Bildverarbeitungsoperatoren, die Punktkoordinaten zurückliefern, nehmen hingegen ein Koordinatensystem an, in dem der Ursprung in der Mitte eines Pixels liegt. Daher muss, um eine konsistente homogene Transformationsmatrix zu erhalten, 0.5 zu den Punktkoordinaten addiert werden, bevor die Transformation berechnet wird.

Parameter

▷ **TransformationType** (input_control) ..... string ⇝ *string*
Typ der zu berechnenden Transformation.
Defaultwert: 'rigid'
Werteliste: TransformationType ∈ {'translation', 'rigid', 'similarity', 'aniso', 'affine'}

▷ **Px** (input_control) ..... point.x-array ⇝ *real*
X-Koordinaten der originalen Punkte.

▷ **Py** (input_control) ..... point.y-array ⇝ *real*
Y-Koordinaten der originalen Punkte.

▷ **L1x** (input_control) ..... point.x-array ⇝ *real*
X-Koordinaten des ersten Punkts auf der korrespondierenden Gerade.

▷ **L1y** (input_control) ..... point.y-array ⇝ *real*
Y-Koordinaten des ersten Punkts auf der korrespondierenden Gerade.

▷ **L2x** (input_control) ..... point.x-array ⇝ *real*
X-Koordinaten des zweiten Punkts auf der korrespondierenden Gerade.

▷ **L2y** (input_control) ..... point.y-array ⇝ *real*
Y-Koordinaten des zweiten Punkts auf der korrespondierenden Gerade.

▷ **HomMat2D** (output_control) ..... hom_mat2d ⇝ *real*
Ausgabe-Transformationsmatrix.

Beispiel

```
* Use point_line_to_hom_mat2d for alignment.
* Read the reference image.
read_image (Image, ReferenceFileName)
* Set up the metrology model with four lines.  Four lines are used
* since this is the minimum number of point-to-line correspondences
* that results in a unique rigid transformation.
Row1 := [RowB1,RowB2,RowB3,RowB4]
Col1 := [ColB1,ColB2,ColB3,ColB4]
Row2 := [RowE1,RowE2,RowE3,RowE4]
Col2 := [ColE1,ColE2,ColE3,ColE4]
create_metrology_model (MetrologyHandle)
add_metrology_object_line_measure (MetrologyHandle, Row1, Col1, \
                                   Row2, Col2, 40, 5, 1, 30, \
                                   [], [], Index)
* Apply the metrology model to the reference image and read out
* the results.
apply_metrology_model (Image, MetrologyHandle)
get_metrology_object_result (MetrologyHandle, 'all', 'all', \
                             'result_type', 'row_begin', \
                             RowBegin)
get_metrology_object_result (MetrologyHandle, 'all', 'all', \
                             'result_type', 'column_begin', \
                             ColBegin)
get_metrology_object_result (MetrologyHandle, 'all', 'all', \
                             'result_type', 'row_end', \
                             RowEnd)
get_metrology_object_result (MetrologyHandle, 'all', 'all', \
                             'result_type', 'column_end', \
                             ColEnd)
* The reference points of the model are the center points of the
* detected line segments.  They will be used to compute the
* transformation from the current image to the reference image
* using point_line_to_hom_mat2d below.
RowRef := 0.5*(RowBegin+RowEnd)
ColRef := 0.5*(ColBegin+ColEnd)
for I := 1 to |FileNames|-1 by 1
```

```
    read_image (Image, FileNames[I])
    * Apply the metrology model to the current image and read out
    * the line segment coordinates.
    apply_metrology_model (Image, MetrologyHandle)
    get_metrology_object_result (MetrologyHandle, 'all', 'all', \
                                 'result_type', 'row_begin', \
                                 RowBegin)
    get_metrology_object_result (MetrologyHandle, 'all', 'all', \
                                 'result_type', 'column_begin', \
                                 ColBegin)
    get_metrology_object_result (MetrologyHandle, 'all', 'all', \
                                 'result_type', 'row_end', \
                                 RowEnd)
    get_metrology_object_result (MetrologyHandle, 'all', 'all', \
                                 'result_type', 'column_end', \
                                 ColEnd)
    * Determine a rigid transformation based on the point-to-line
    * correspondences from the reference points to the extracted
    * lines.  Note that this determines a transformation from the
    * reference points to the lines in the current image.
    * Therefore, we must invert this transformation to obtain the
    * transformation from the current image to the rerefence image.
    point_line_to_hom_mat2d ('rigid', RowRef+0.5, ColRef+0.5, \
                             RowBegin+0.5, ColBegin+0.5, \
                             RowEnd+0.5, ColEnd+0.5, HomMat2D)
    hom_mat2d_invert (HomMat2D, HomMat2DInvert)
    affine_trans_image (Image, ImageTrans, HomMat2DInvert, \
                        'constant', 'false')
    * Now that the current image has been aligned with the
    * reference image, we can do some processing based on the
    * aligned image ImageTrans.
    * [...]
endfor
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

affine_trans_image, affine_trans_image_size, affine_trans_region, affine_trans_contour_xld, affine_trans_polygon_xld, affine_trans_point_2d

Siehe auch

vector_to_hom_mat2d, vector_to_aniso, vector_to_similarity, vector_to_rigid

Modul

Foundation

projective_trans_pixel (: : HomMat2D, Row, Col : RowTrans, ColTrans)

Projiziert Pixelkoordinaten mittels einer homogenen projektiven Transformationsmatrix.

projective_trans_pixel wendet die homogene projektive Transformationsmatrix HomMat2D auf alle Eingabepixel (Row,Col) an und liefert eine Liste von Ergebnispixeln (RowTrans,ColTrans) zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in HomMat2D übergeben wird.

Der Unterschied zwischen projective_trans_pixel und projective_trans_point_2d liegt im verwendeten Koordinatensystem: projective_trans_pixel nutzt ein Koordinatensystem mit Ursprung in der linken oberen Ecke des Bildes, während projective_trans_point_2d das Standard-Bildkoordinatensystem verwendet, dessen Ursprung in der Mitte des linken oberen Pixels liegt und das auch von Operatoren wie area_center genutzt wird.

projective_trans_pixel entspricht den folgenden Schritten (Ein- und Ausgabepunkte als homogene Vektoren dargestellt):

$$\begin{pmatrix} \mathtt{RTrans} \\ \mathtt{CTrans} \\ \mathtt{WTrans} \end{pmatrix} = \mathtt{HomMat2D} \cdot \begin{pmatrix} \mathtt{Row} \\ \mathtt{Col} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \mathtt{RowTrans} \\ \mathtt{ColTrans} \end{pmatrix} = \begin{pmatrix} \frac{\mathtt{RTrans}}{\mathtt{WTrans}} \\ \frac{\mathtt{CTrans}}{\mathtt{WTrans}} \end{pmatrix}$$

Falls bei dieser Transformation ein unendlich ferner Punkt erzeugt wird (WTrans = 0), wird eine Fehlermeldung zurückgeliefert.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **Row** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Eingabepixel (Zeilenkoordinate).
  Defaultwert: 64
  Wertevorschläge: Row ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Col** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Eingabepixel (Spaltenkoordinate).
  Defaultwert: 64
  Wertevorschläge: Col ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **RowTrans** (output_control) ..... point.x(-array) ⇝ *real*
  Ausgabepixel (Zeilenkoordinate).
- ▷ **ColTrans** (output_control) ..... point.y(-array) ⇝ *real*
  Ausgabepixel (Spaltenkoordinate).

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

vector_to_proj_hom_mat2d, hom_vector_to_proj_hom_mat2d, proj_match_points_ransac, proj_match_points_ransac_guided, hom_mat3d_project

Siehe auch

projective_trans_image, projective_trans_image_size, projective_trans_region, projective_trans_contour_xld, projective_trans_point_2d

Modul

Foundation

projective_trans_point_2d (: : HomMat2D, Px, Py, Pw : Qx, Qy, Qw)

Projiziert einen homogenen 2D-Punkt mittels einer projektiven Transformationsmatrix.

projective_trans_point_2d wendet die homogene projektive Transformationsmatrix HomMat2D auf alle homogenen Eingabepunkte (Px,Py,Pw) an und liefert eine Liste von homogenen Ergebnispunkten (Qx,Qy,Qw) zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in HomMat2D übergeben wird. Dies entspricht der folgenden Gleichung (Ein- und Ausgabepunkte als homogene Vektoren dargestellt):

$$\begin{pmatrix} \mathtt{Qx} \\ \mathtt{Qy} \\ \mathtt{Qw} \end{pmatrix} = \mathtt{HomMat2D} \cdot \begin{pmatrix} \mathtt{Px} \\ \mathtt{Py} \\ \mathtt{Pw} \end{pmatrix}$$

Um die homogenen Koordinaten in euklidische Koordinaten zu transformieren, muss durch Qw geteilt werden:

$$\begin{pmatrix} Ex \\ Ey \end{pmatrix} = \begin{pmatrix} \frac{Qx}{Qw} \\ \frac{Qy}{Qw} \end{pmatrix}$$

Falls die zu transformierenden Punkte in Standard-Bildkoordinaten vorliegen, müssen die *Zeilen*-Koordinaten der Punkte in Px und die *Spalten*-Koordinaten in Py übergeben werden. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Parameter

- ▷ **HomMat2D** (input_control) . . . . . hom_mat2d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **Px** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (x-Koordinate).
- ▷ **Py** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (y-Koordinate).
- ▷ **Pw** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (w-Koordinate).
- ▷ **Qx** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (x-Koordinate).
- ▷ **Qy** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (y-Koordinate).
- ▷ **Qw** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (w-Koordinate).

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

vector_to_proj_hom_mat2d, hom_vector_to_proj_hom_mat2d, proj_match_points_ransac, proj_match_points_ransac_guided, hom_mat3d_project

Siehe auch

projective_trans_image, projective_trans_image_size, projective_trans_region, projective_trans_contour_xld, projective_trans_pixel

Modul

Foundation

serialize_hom_mat2d (: : HomMat2D : SerializedItemHandle)

Serialisieren einer homogenen 2D-Transformationsmatrix.

serialize_hom_mat2d serialisiert die Daten einer homogenen 2D-Transformationsmatrix (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die Transformationsmatrix wird in dem Handle HomMat2D übergeben. Die serialisierte Transformationsmatrix wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_hom_mat2d wieder deserialisiert werden.

Parameter

- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Transformationsmatrix.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_hom_mat2d den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_hom_mat2d

Modul

Foundation

vector_angle_to_rigid (: : Row1, Column1, Angle1, Row2, Column2, Angle2 : HomMat2D)

Berechnet eine starre affine Transformation aus Punkten und Winkeln.

vector_angle_to_rigid berechnet aus einer Punktkorrespondenz und zwei zugehörigen Winkeln eine starre affine Transformation, d.h. eine Transformation bestehend aus Rotation und Translation, und liefert sie in der homogenen Transformationsmatrix HomMat2D zurück. Diese beinhaltet 2 Komponenten: eine Rotationsmatrix $\mathbf{R}$ und einen Translationsvektor $\mathbf{T}$ (siehe auch hom_mat2d_rotate und hom_mat2d_translate):

$$\mathtt{HomMat2D} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0\,0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{T} \\ 0 & 1 & \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & 0 \\ & 0 \\ 0\,0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T}) \cdot \mathbf{H}(\mathbf{R})$$

Die Koordinaten des ursprünglichen Punktes werden in (Row1,Column1) übergeben, der zugehörige Winkel in Angle1. Die Koordinaten des transformierten Punktes werden in (Row2,Column2) übergeben, der zugehörige Winkel in Angle2. Die folgende Gleichung beschreibt die Transformation des Punktes mit Hilfe homogener Vektoren:

$$\begin{pmatrix} \mathtt{Row2} \\ \mathtt{Column2} \\ 1 \end{pmatrix} = \mathtt{HomMat2D} \cdot \begin{pmatrix} \mathtt{Row1} \\ \mathtt{Column1} \\ 1 \end{pmatrix}$$

Der Operator vector_angle_to_rigid ist insbesondere sinnvoll, um aus den Ergebnissen von Matching-Operatoren (z.B. find_shape_model) eine starre Transformation zu konstruieren, die ein Referenzbild in das aktuelle Bild transformiert oder (wenn die Parameter in der umgekehrten Reihenfolge übergeben werden) das aktuelle Bild in das Referenzbild transformiert.

HomMat2D kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. affine_trans_image, verwendet werden.

Parameter

- ▷ **Row1** (input_control) . . . . . point.y ⇝ *real* / integer
  Zeilenkoordinate des Ausgangspunkts.
- ▷ **Column1** (input_control) . . . . . point.x ⇝ *real* / integer
  Spaltenkoordinate des Ausgangspunkts.
- ▷ **Angle1** (input_control) . . . . . angle.rad ⇝ *real* / integer
  Winkel des Ausgangspunkts.
- ▷ **Row2** (input_control) . . . . . point.y ⇝ *real* / integer
  Zeilenkoordinate des transformierten Punkts.
- ▷ **Column2** (input_control) . . . . . point.x ⇝ *real* / integer
  Spaltenkoordinate des transformierten Punkts.
- ▷ **Angle2** (input_control) . . . . . angle.rad ⇝ *real* / integer
  Winkel des transformierten Punkts.
- ▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Beispiel

```
create_shape_model (ImageReduced, 0, rad(-45), rad(180), 0, \
                    'none', 'use_polarity', 30, 10, ModelID)
get_shape_model_contours (ModelXLD, ModelID, 1)
find_shape_model (SearchImage, ModelID, rad(-45), rad(180), \
                  0.5, 1, 0.5, 'interpolation', \
                                       0, 0, Row, Column, Angle, Score)
* Create transformation matrix
vector_angle_to_rigid (0, 0, 0, Row, Column, Angle, HomMat2DObject)
* Transform model contours for visualization
affine_trans_contour_xld (ModelXLD, ObjectXLD, HomMat2DObject)
* Calculate true position of the model origin in the search image
affine_trans_pixel (HomMat2DObject, 0, 0, RowObject, ColObject)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

find_shape_model

Nachfolger

hom_mat2d_invert, affine_trans_image, affine_trans_region, affine_trans_contour_xld, affine_trans_polygon_xld, affine_trans_point_2d

Alternativen

vector_to_rigid

Siehe auch

vector_field_to_hom_mat2d

Modul

Foundation

vector_field_to_hom_mat2d (VectorField : : : HomMat2D)

Berechnen der Näherung eines Verschiebungsvektorfeldes durch eine affine Transformation.

vector_field_to_hom_mat2d berechnet aus dem Verschiebungsvektorfeld VectorField eine affine Transformation, die die unterliegende Transformation möglichst gut annähert. Diese wird in HomMat2D zurückgeliefert.

Wenn das Verschiebungsvektorfeld aus einem Originalbild I_{orig} und einem zweiten Bild I_{res} berechnet worden ist, so enthält die intern gespeicherte Transformationsmatrix (siehe affine_trans_image) eine Transformation, die beschreibt, wie das erste Bild I_{orig} in das zweite Bild I_{res} transformiert werden kann. Es ist zu beachten, dass das VectorField in relativen Koordinaten angegeben werden muss, wie es z.B. optical_flow_mg liefert.

Parameter

- ▷ **VectorField** (input_object) ..... singlechannelimage ⇝ *object :* vector_field
  Eingabebild.
- ▷ **HomMat2D** (output_control) ..... hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

optical_flow_mg

Nachfolger

affine_trans_image

Alternativen

vector_to_hom_mat2d

Modul

Foundation

vector_to_aniso (: : Px, Py, Qx, Qy : HomMat2D)

Erzeugt eine Näherung für eine anisotrope Ähnlichkeitstransformation aus Punktkorrespondenzen.

vector_to_aniso erzeugt aus (mindestens drei) Punktkorrespondenzen eine Näherung für eine anisotrope Ähnlichkeitstransformation, d.h. eine Transformation bestehend aus einer ungleichförmigen Skalierung, einer Rotation und einer Translation, und liefert sie in der homogenen Transformationsmatrix HomMat2D zurück. Diese beinhaltet 3 Komponenten: eine Skalierungsmatrix $\mathbf{S}$ mit nicht-identischer Skalierung in x- und y-Richtung, eine Rotationsmatrix $\mathbf{R}$ und einen Translationsvektor $\mathbf{T}$ (siehe auch hom_mat2d_scale, hom_mat2d_rotate und hom_mat2d_translate):

$$\texttt{HomMat2D} = \begin{bmatrix} \mathbf{R}\cdot\mathbf{S} & \mathbf{T} \\ 0\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{T} \\ 0 & 1 & \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & & 0 \\ & & 0 \\ 0\,0 & & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T})\cdot\mathbf{H}(\mathbf{R})\cdot\mathbf{H}(\mathbf{S})$$

Die Punktkorrespondenzen werden in den Tupeln (Px,Py) und (Qx,Qy) übergeben, wobei korrespondierende Punkte an denselben Indexpositionen stehen müssen. Die Transformation ist durch die mindestens drei Punktkorrespondenzen immer überbestimmt. Deswegen ist die zurückgelieferte Transformation diejenige, die die Abstände zwischen den Eingabepunkten (Px,Py) und den transformierten Zielpunkten (Qx,Qy) minimiert, wie in der folgenden Gleichung beschrieben (Punkte als homogene Vektoren dargestellt):

$$\sum_i \left\| \begin{pmatrix} \texttt{Qx}[i] \\ \texttt{Qy}[i] \\ 1 \end{pmatrix} - \texttt{HomMat2D} \cdot \begin{pmatrix} \texttt{Px}[i] \\ \texttt{Py}[i] \\ 1 \end{pmatrix} \right\|^2 = \text{minimal}$$

HomMat2D kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. affine_trans_image, verwendet werden.

In einer anisotropen Ähnlichkeitstransformation, wie sie oben definiert ist, werden die Punkte zuerst skaliert und dann rotiert. Manchmal ist eine Transformation nützlich, bei der zuerst rotiert und dann skaliert wird. Diese Art von Transformation kann mit `vector_to_aniso` berechnet werden, indem die Punktkorrespondenzen in umgekehrter Reihenfolge übergeben werden, d.h. die Punkte (`Px`,`Py`) werden in (`Qx`,`Qy`) und umgekehrt. Die so erhaltene Transformation muss danach mit `hom_mat2d_invert` invertiert werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (`Row`,`Column`). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Weiterhin ist zu beachten, dass, falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, angenommen wird, dass der Ursprung des Koordinatensystems der homogenen Transformationsmatrix in der linken oberen Ecke des Pixels liegt. Die Bildverarbeitungsoperatoren, die Punktkoordinaten zurückliefern, nehmen hingegen ein Koordinatensystem an, in dem der Ursprung in der Mitte eines Pixels liegt. Daher muss, um eine konsistente homogene Transformationsmatrix zu erhalten, 0.5 zu den Punktkoordinaten addiert werden, bevor die Transformation berechnet wird.

Parameter

- ▷ **Px** (input_control) . . . . . point.x-array ⇝ *real*
  X-Koordinaten der originalen Punkte.
- ▷ **Py** (input_control) . . . . . point.y-array ⇝ *real*
  Y-Koordinaten der originalen Punkte.
- ▷ **Qx** (input_control) . . . . . point.x-array ⇝ *real*
  X-Koordinaten der transformierten Punkte.
- ▷ **Qy** (input_control) . . . . . point.y-array ⇝ *real*
  Y-Koordinaten der transformierten Punkte.
- ▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`hom_mat2d_invert`, `affine_trans_image`, `affine_trans_image_size`, `affine_trans_region`, `affine_trans_contour_xld`, `affine_trans_polygon_xld`, `affine_trans_point_2d`

Alternativen

`vector_to_hom_mat2d`, `vector_to_similarity`, `vector_to_rigid`

Siehe auch

`vector_field_to_hom_mat2d`, `point_line_to_hom_mat2d`

Modul

Foundation

vector_to_hom_mat2d (: : Px, Py, Qx, Qy : HomMat2D)

Erzeugt eine Näherung für eine affine Transformation aus Punktkorrespondenzen.

vector_to_hom_mat2d erzeugt aus (mindestens drei) Punktkorrespondenzen eine Näherung für eine affine Transformation und liefert sie in der homogenen Transformationsmatrix HomMat2D zurück (siehe hom_mat2d_to_affine_par für den Aufbau der homogenen Transformationsmatrix).

Die Punktkorrespondenzen werden in den Tupeln (Px,Py) und (Qx,Qy) übergeben, wobei korrespondierende Punkte an denselben Indexpositionen stehen müssen. Die Transformation ist überbestimmt, falls mehr als drei Punktkorrespondenzen übergeben werden. In diesem Fall ist die zurückgelieferte Transformation diejenige, die die Abstände zwischen den Eingabepunkten (Px,Py) und den Zielpunkten (Qx,Qy) minimiert, wie in der folgenden Gleichung beschrieben (Punkte als homogene Vektoren dargestellt):

$$\sum_i \left\| \begin{pmatrix} \mathtt{Qx}[i] \\ \mathtt{Qy}[i] \\ 1 \end{pmatrix} - \mathtt{HomMat2D} \cdot \begin{pmatrix} \mathtt{Px}[i] \\ \mathtt{Py}[i] \\ 1 \end{pmatrix} \right\|^2 = \text{minimal}$$

HomMat2D kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. affine_trans_image, verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Weiterhin ist zu beachten, dass, falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, angenommen wird, dass der Ursprung des Koordinatensystems der homogenen Transformationsmatrix in der linken oberen Ecke des Pixels liegt. Die Bildverarbeitungsoperatoren, die Punktkoordinaten zurückliefern, nehmen hingegen ein Koordinatensystem an, in dem der Ursprung in der Mitte eines Pixels liegt. Daher muss, um eine konsistente homogene Transformationsmatrix zu erhalten, 0.5 zu den Punktkoordinaten addiert werden, bevor die Transformation berechnet wird.

Parameter

- ▷ **Px** (input_control) ..... point.x-array ⇝ *real*
  X-Koordinaten der originalen Punkte.
- ▷ **Py** (input_control) ..... point.y-array ⇝ *real*
  Y-Koordinaten der originalen Punkte.
- ▷ **Qx** (input_control) ..... point.x-array ⇝ *real*
  X-Koordinaten der transformierten Punkte.
- ▷ **Qy** (input_control) ..... point.y-array ⇝ *real*
  Y-Koordinaten der transformierten Punkte.
- ▷ **HomMat2D** (output_control) ..... hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

affine_trans_image, affine_trans_image_size, affine_trans_region, affine_trans_contour_xld, affine_trans_polygon_xld, affine_trans_point_2d

Alternativen

vector_to_aniso, vector_to_similarity, vector_to_rigid

Siehe auch

vector_field_to_hom_mat2d, point_line_to_hom_mat2d

Modul

Foundation

vector_to_proj_hom_mat2d (: : Px, Py, Qx, Qy, Method, CovXX1, CovYY1, CovXY1, CovXX2, CovYY2, CovXY2 : HomMat2D, Covariance)

Bestimmt eine projektive Transformationsmatrix unter Verwendung vorgegebener Punktkorrespondenzen.

vector_to_proj_hom_mat2d ermittelt aus mindestens 4 vorgegebenen Punktkorrespondenzen

$$\mathtt{HomMat2D} \cdot \begin{pmatrix} \mathtt{Px} \\ \mathtt{Py} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathtt{Qx} \\ \mathtt{Qy} \\ 1 \end{pmatrix}.$$

die homogene projektive Transformationsmatrix HomMat2D, die die obigen Gleichungen am besten erfüllt. Sind weniger als 4 Punktpaare (Px,Py), (Qx,Qy) angegeben, so ist eine solche Transformation nicht eindeutig bestimmbar. Sind genau 4 Punktpaare gegeben, so kann eine Matrix gefunden werden, die die Punkte genau ineinander überführt. Andernfalls wird die Matrix so gewählt, dass der Fehler möglichst klein wird. Um dieses Ergebnis zu erreichen, sind verschiedene Algorithmen verfügbar. Welches Verfahren verwendet werden soll, kann mittels des Parameters Method gesteuert werden. Method=*'dlt'* verwendet einen einfachen und schnellen, dafür aber ungenaueren Fehlerschätzalgorithmus, Method=*'normalized_dlt'* bietet im allgemeinen einen guten Kompromiss zwischen Geschwindigkeit und Genauigkeit, während Method=*'gold_standard'* eine mathematisch optimale, dafür aber langsamere, Optimierung durchführt.

Falls *'gold_standard'* verwendet wird und die Punkte mit einem Operator wie points_foerstner, der die Kovarianzmatrix für jeden Punkt zurückliefert, extrahiert wurden, kann dies in der Berechnung berücksichtigt werden, indem die Kovarianzen in CovYY1, CovXX1, CovXY1 für die Punkte des ersten Bildes und in CovYY2, CovXX2, CovXY2 für die Punkte des zweiten Bildes übergeben werden. Die Kovarianzmatrizen sind symmetrische 2×2 Matrizen. CovXX1/CovXX2 und CovYY1/CovYY2 sind dabei die Diagonalelemente der Matrizen, während CovXY1/CovXY2 die Subdiagonalelemente angeben. Falls eine andere Methode als *'gold_standard'* verwendet wird oder die Kovarianzen unbekannt sind, können in den Kovarianzparametern leere Tupel übergeben werden.

Im Gegensatz zu hom_vector_to_proj_hom_mat2d können in vector_to_proj_hom_mat2d keine unendlich fernen Punkte zur Bestimmung der Transformation verwendet werden. Falls dies notwendig ist, muss hom_vector_to_proj_hom_mat2d verwendet werden. Falls die Korrespondenz zwischen den Punkten nicht bestimmt worden ist, sollte proj_match_points_ransac zur Bestimmung der Korrespondenz sowie der Transformation verwendet werden.

Falls die zu transformierenden Punkte in Standard-Bildkoordinaten vorliegen, müssen die *Zeilen*-Koordinaten der Punkte in Px und die *Spalten*-Koordinaten in Py übergeben werden. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Weiterhin ist zu beachten, dass, falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, angenommen wird, dass der Ursprung des Koordinatensystems der homogenen Transformationsmatrix in der linken oberen Ecke des Pixels liegt. Die Bildverarbeitungsoperatoren, die Punktkoordinaten zurückliefern, nehmen hingegen ein Koordinatensystem an, in dem der Ursprung in der Mitte eines Pixels liegt. Daher muss, um eine konsistente homogene Transformationsmatrix zu erhalten, 0.5 zu den Punktkoordinaten addiert werden, bevor die Transformation berechnet wird.

Parameter

- ▷ **Px** (input_control) . . . . . point.x-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Zeilenkoordinate).
- ▷ **Py** (input_control) . . . . . point.y-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Spaltenkoordinate).
- ▷ **Qx** (input_control) . . . . . point.x-array ⇝ *real*
  Eingabepunkte in Bild 2 (Zeilenkoordinate).
- ▷ **Qy** (input_control) . . . . . point.y-array ⇝ *real*
  Eingabepunkte in Bild 2 (Spaltenkoordinate).
- ▷ **Method** (input_control) . . . . . string ⇝ *string*
  Schätzalgorithmus.
  Defaultwert: 'normalized_dlt'
  Werteliste: `Method` ∈ {'normalized_dlt', 'gold_standard', 'dlt'}
- ▷ **CovXX1** (input_control) . . . . . real-array ⇝ *real*
  Varianz in Zeilenrichtung der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovYY1** (input_control) . . . . . real-array ⇝ *real*
  Varianz in Spaltenrichtung der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovXY1** (input_control) . . . . . real-array ⇝ *real*
  Kovarianz der Punkte in Bild 1.
  Defaultwert: []
- ▷ **CovXX2** (input_control) . . . . . real-array ⇝ *real*
  Varianz in Zeilenrichtung der Punkte in Bild 2.
  Defaultwert: []
- ▷ **CovYY2** (input_control) . . . . . real-array ⇝ *real*
  Varianz in Spaltenrichtung der Punkte in Bild 2.
  Defaultwert: []
- ▷ **CovXY2** (input_control) . . . . . real-array ⇝ *real*
  Kovarianz der Punkte in Bild 2.
  Defaultwert: []
- ▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **Covariance** (output_control) . . . . . real-array ⇝ *real*
  9×9 Kovarianzmatrix der projektiven Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`proj_match_points_ransac`, `proj_match_points_ransac_guided`, `points_foerstner`, `points_harris`

Nachfolger

`projective_trans_image`, `projective_trans_image_size`, `projective_trans_region`, `projective_trans_contour_xld`, `projective_trans_point_2d`, `projective_trans_pixel`

Alternativen

`hom_vector_to_proj_hom_mat2d`, `proj_match_points_ransac`, `proj_match_points_ransac_guided`

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2000.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Calibration

vector_to_proj_hom_mat2d_distortion (: : Points1Row, Points1Col, Points2Row, Points2Col, CovRR1, CovRC1, CovCC1, CovRR2, CovRC2, CovCC2, ImageWidth, ImageHeight, Method : HomMat2D, Kappa, Error)

Bestimmung der projektiven Transformationsmatrix und des radialen Verzeichnungskoeffizienten unter Verwendung vorgegebener Punktkorrespondenzen.

vector_to_proj_hom_mat2d_distortion ermittelt aus vorgegebenen Punktkorrespondenzen (Points1Row,Points1Col), (Points2Row,Points2Col) die projektive Transformationsmatrix HomMat2D und den radialen Verzeichnungskoeffizienten Kappa (κ), welche die folgende Gleichung am besten erfüllen:

$$\begin{pmatrix} r_2 \\ c_2 \\ 1 \end{pmatrix} = \texttt{HomMat2D} \cdot \begin{pmatrix} r_1 \\ c_1 \\ 1 \end{pmatrix}.$$

Hierbei sind (r_1, c_1) und (r_2, c_2) Bildpunkte, die durch Entzerrung der Eingabebildpunkte mit dem Divisionsmodell entstehen (siehe Kalibrierung):

$$r = \frac{\tilde{r}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)} \qquad c = \frac{\tilde{c}}{1 + \kappa(\tilde{r}^2 + \tilde{c}^2)}$$

Hierbei bezeichnen $(\tilde{r}_1, \tilde{c}_1) = (\texttt{Points1Row} - 0.5(\texttt{ImageHeight} - 1), \texttt{Points1Col} - 0.5(\texttt{ImageWidth} - 1))$ und $(\tilde{r}_2, \tilde{c}_2) = (\texttt{Points2Row} - 0.5(\texttt{ImageHeight} - 1), \texttt{Points2Col} - 0.5(\texttt{ImageWidth} - 1))$ die verzerrten Bildpunkte relativ zum Bildmittelpunkt. vector_to_proj_hom_mat2d_distortion nimmt also an, dass der Hauptpunkt, d.h. das Zentrum der radialen Verzeichnungen, im Bildmittelpunkt liegt. Die Punktkorrespondenzen (Points1Row,Points1Col) und (Points2Row,Points2Col) müssen aus Bildern extrahiert werden, die mit derselben Kamera aufgenommen worden sind.

Das zurückgelieferte Kappa kann dazu verwendet werden, Kameraparameter zu konstruieren, die zur Entzerrung von Bildern oder Bildpunkten verwendet werden können (siehe change_radial_distortion_cam_par, change_radial_distortion_image und change_radial_distortion_points):

$$\texttt{CamPar} = ['area_scan_telecentric_division', 1.0, \texttt{Kappa}, 1.0, 1.0, 0.5(\texttt{ImageWidth} - 1), 0.5(\texttt{ImageHeight} - 1), \texttt{ImageWidth}, \texttt{ImageHeight}]$$

Die minimale Anzahl an Punktkorrespondenzen ist fünf. vector_to_proj_hom_mat2d_distortion minimiert den Transformationsfehler basierend auf den obigen Gleichungen. Um dies zu erreichen, sind verschiedene Algorithmen verfügbar. Welches Verfahren verwendet werden soll, kann mittels des Parameters Method gesteuert werden. Für Method = *'linear'* wird ein lineares Verfahren verwendet, das einen algebraischen Fehler basierend auf den obigen Gleichungen minimiert. Dieses Verfahren ist sehr schnell und liefert genaue Ergebnisse für geringes bis mittleres Rauschen der Punktkoordinaten und für die meisten Verzeichnungen (außer kleinen Verzeichnungen). Für Method = *'gold_standard'* wird eine mathematisch optimale, dafür aber langsamere, Optimierung durchführt, die den geometrischen Reprojektionsfehler minimiert. Im allgemeinen sollte Method = *'gold_standard'* gewählt werden.

Falls *'gold_standard'* verwendet wird und die Punkte mit einem Operator wie points_foerstner, der die Kovarianzmatrix für jeden Punkt zurückliefert, extrahiert wurden, kann dies in der Berechnung berücksichtigt werden, indem die Kovarianzen in CovRR1, CovRC1, CovCC1 für die Punkte des ersten Bildes und in CovRR2, CovRC2, CovCC2 für die Punkte des zweiten Bildes übergeben werden. Sind die Kovarianzen unbekannt und wird die Methode *'gold_standard'* verwendet, so werden zur Berechnung intern Einheits-Kovarianzmatrizen angenommen. Falls *'linear'* verwendet wird oder die Kovarianzen unbekannt sind, können in den Kovarianzparametern leere Tupel übergeben werden.

Die Größe `Error` ist ein Gütemaß für die Schätzung der projektiven Transformationsmatrix und gibt den mittleren symmetrischen euklidischen Abstand in Pixeln zwischen den Punkten und ihren korrespondierenden transformierten Punkten an.

Falls die Korrespondenz zwischen den Punkten nicht bestimmt worden ist, sollten `proj_match_points_distortion_ransac` oder `proj_match_points_distortion_ransac_guided` zur Bestimmung der Korrespondenz sowie der Transformation verwendet werden.

Achtung

Es ist zu beachten, dass, falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, angenommen wird, dass der Ursprung des Koordinatensystems der homogenen Transformationsmatrix in der linken oberen Ecke des Pixels liegt. Die Bildverarbeitungsoperatoren, die Punktkoordinaten zurückliefern, nehmen hingegen ein Koordinatensystem an, in dem der Ursprung in der Mitte eines Pixels liegt. Daher muss, um eine konsistente homogene Transformationsmatrix zu erhalten, 0.5 zu den Punktkoordinaten addiert werden, bevor die Transformation berechnet wird.

Parameter

- ▷ **`Points1Row`** (input_control) . . . . . point.y-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Zeilenkoordinate).
  Restriktion: length(Points1Row) >= 5
- ▷ **`Points1Col`** (input_control) . . . . . point.x-array ⇝ *real* / integer
  Eingabepunkte in Bild 1 (Spaltenkoordinate).
  Restriktion: length(Points1Col) == length(Points1Row)
- ▷ **`Points2Row`** (input_control) . . . . . point.y-array ⇝ *real* / integer
  Eingabepunkte in Bild 2 (Zeilenkoordinate).
  Restriktion: length(Points2Row) == length(Points1Row)
- ▷ **`Points2Col`** (input_control) . . . . . point.x-array ⇝ *real* / integer
  Eingabepunkte in Bild 2 (Spaltenkoordinate).
  Restriktion: length(Points2Col) == length(Points1Row)
- ▷ **`CovRR1`** (input_control) . . . . . number-array ⇝ *real* / integer
  Varianz in Zeilenrichtung der Punkte in Bild 1.
  Defaultwert: []
- ▷ **`CovRC1`** (input_control) . . . . . number-array ⇝ *real* / integer
  Kovarianz der Punkte in Bild 1.
  Defaultwert: []
- ▷ **`CovCC1`** (input_control) . . . . . number-array ⇝ *real* / integer
  Varianz in Spaltenrichtung der Punkte in Bild 1.
  Defaultwert: []
- ▷ **`CovRR2`** (input_control) . . . . . number-array ⇝ *real* / integer
  Varianz in Zeilenrichtung der Punkte in Bild 2.
  Defaultwert: []
- ▷ **`CovRC2`** (input_control) . . . . . number-array ⇝ *real* / integer
  Kovarianz der Punkte in Bild 2.
  Defaultwert: []
- ▷ **`CovCC2`** (input_control) . . . . . number-array ⇝ *real* / integer
  Varianz in Spaltenrichtung der Punkte in Bild 2.
  Defaultwert: []
- ▷ **`ImageWidth`** (input_control) . . . . . integer ⇝ *integer*
  Breite der Bilder, aus denen die Punkte extrahiert wurden.
  Restriktion: ImageWidth > 0
- ▷ **`ImageHeight`** (input_control) . . . . . integer ⇝ *integer*
  Höhe der Bilder, aus denen die Punkte extrahiert wurden.
  Restriktion: ImageHeight > 0
- ▷ **`Method`** (input_control) . . . . . string ⇝ *string*
  Schätzalgorithmus.
  Defaultwert: 'gold_standard'
  Werteliste: `Method` ∈ {'linear', 'gold_standard'}

▷ **HomMat2D** (output_control) ..... hom_mat2d ⇝ *real*
Homogene projektive Transformationsmatrix.

▷ **Kappa** (output_control) ..... real ⇝ *real*
Berechneter Verzeichnungskoeffizient.

▷ **Error** (output_control) ..... real ⇝ *real*
Mittlerer quadratischer Transformationsfehler.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`proj_match_points_distortion_ransac`,
`proj_match_points_distortion_ransac_guided`, `points_foerstner`, `points_harris`

Nachfolger

`change_radial_distortion_cam_par`, `change_radial_distortion_image`,
`change_radial_distortion_points`, `projective_trans_image`,
`projective_trans_image_size`, `projective_trans_region`,
`projective_trans_contour_xld`, `projective_trans_point_2d`,
`projective_trans_pixel`

Alternativen

`vector_to_proj_hom_mat2d`, `hom_vector_to_proj_hom_mat2d`

Siehe auch

`calibrate_cameras`

Literatur

Richard Hartley, Andrew Zisserman: „Multiple View Geometry in Computer Vision“; Cambridge University Press, Cambridge; 2003.

Olivier Faugeras, Quang-Tuan Luong: „The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications“; MIT Press, Cambridge, MA; 2001.

Modul

Calibration

vector_to_rigid (: : Px, Py, Qx, Qy : HomMat2D)

Erzeugt eine Näherung für eine starre affine Transformation aus Punktkorrespondenzen.

`vector_to_rigid` erzeugt aus (mindestens zwei) Punktkorrespondenzen eine Näherung für eine starre affine Transformation, d.h. eine Transformation bestehend aus Rotation und Translation, und liefert sie in der homogenen Transformationsmatrix `HomMat2D` zurück. Diese beinhaltet 2 Komponenten: eine Rotationsmatrix $\mathbf{R}$ und einen Translationsvektor $\mathbf{T}$ (siehe auch `hom_mat2d_rotate` und `hom_mat2d_translate`):

$$\texttt{HomMat2D} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0\,0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{T} \\ 0 & 1 & \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & 0 \\ & 0 \\ 0\,0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T}) \cdot \mathbf{H}(\mathbf{R})$$

Die Punktkorrespondenzen werden in den Tupeln (`Px`,`Py`) und (`Qx`,`Qy`) übergeben, wobei korrespondierende Punkte an denselben Indexpositionen stehen müssen. Die Transformation ist durch die mindestens zwei Punktkorrespondenzen immer überbestimmt. Deswegen ist die zurückgelieferte Transformation diejenige, die die Abstände zwischen den Eingabepunkten (`Px`,`Py`) und den transformierten Zielpunkten (`Qx`,`Qy`) minimiert, wie in der folgenden Gleichung beschrieben (Punkte als homogene Vektoren dargestellt):

$$\sum_i \left\| \begin{pmatrix} \texttt{Qx}[i] \\ \texttt{Qy}[i] \\ 1 \end{pmatrix} - \texttt{HomMat2D} \cdot \begin{pmatrix} \texttt{Px}[i] \\ \texttt{Py}[i] \\ 1 \end{pmatrix} \right\|^2 = \text{minimal}$$

HomMat2D kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. affine_trans_image, verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Weiterhin ist zu beachten, dass, falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, angenommen wird, dass der Ursprung des Koordinatensystems der homogenen Transformationsmatrix in der linken oberen Ecke des Pixels liegt. Die Bildverarbeitungsoperatoren, die Punktkoordinaten zurückliefern, nehmen hingegen ein Koordinatensystem an, in dem der Ursprung in der Mitte eines Pixels liegt. Daher muss, um eine konsistente homogene Transformationsmatrix zu erhalten, 0.5 zu den Punktkoordinaten addiert werden, bevor die Transformation berechnet wird.

Parameter

- ▷ **Px** (input_control) . . . . . point.x-array ⇝ *real*
  X-Koordinaten der originalen Punkte.
- ▷ **Py** (input_control) . . . . . point.y-array ⇝ *real*
  Y-Koordinaten der originalen Punkte.
- ▷ **Qx** (input_control) . . . . . point.x-array ⇝ *real*
  X-Koordinaten der transformierten Punkte.
- ▷ **Qy** (input_control) . . . . . point.y-array ⇝ *real*
  Y-Koordinaten der transformierten Punkte.
- ▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

affine_trans_image, affine_trans_image_size, affine_trans_region, affine_trans_contour_xld, affine_trans_polygon_xld, affine_trans_point_2d

Alternativen

vector_to_hom_mat2d, vector_to_aniso, vector_to_similarity

Siehe auch

vector_field_to_hom_mat2d, point_line_to_hom_mat2d

Modul

Foundation

vector_to_similarity (: : Px, Py, Qx, Qy : HomMat2D)

Erzeugt eine Näherung für eine Ähnlichkeitstransformation aus Punktkorrespondenzen.

vector_to_similarity erzeugt aus (mindestens zwei) Punktkorrespondenzen eine Näherung für eine Ähnlichkeitstransformation, d.h. eine Transformation bestehend aus einer gleichförmigen Skalierung, Rotation und Translation, und liefert sie in der homogenen Transformationsmatrix HomMat2D zurück. Diese beinhaltet 3 Komponenten: eine Skalierungsmatrix $\mathbf{S}$ mit identischer Skalierung in x- und y-Richtung, eine Rotationsmatrix $\mathbf{R}$ und einen Translationsvektor $\mathbf{T}$ (siehe auch hom_mat2d_scale, hom_mat2d_rotate und hom_mat2d_translate):

$$\texttt{HomMat2D} = \begin{bmatrix} \mathbf{R}\cdot\mathbf{S} & \mathbf{T} \\ 0\ \ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{T} \\ 0 & 1 & \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & 0 \\ & 0 \\ 0\,0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{S} & 0 \\ & 0 \\ 0\,0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T}) \cdot \mathbf{H}(\mathbf{R}) \cdot \mathbf{H}(\mathbf{S})$$

Die Punktkorrespondenzen werden in den Tupeln (Px,Py) und (Qx,Qy) übergeben, wobei korrespondierende Punkte an denselben Indexpositionen stehen müssen. Die Transformation ist überbestimmt, falls mehr als zwei Punktkorrespondenzen übergeben werden. In diesem Fall ist die zurückgelieferte Transformation diejenige, die die Abstände zwischen den Eingabepunkten (Px,Py) und den Zielpunkten (Qx,Qy) minimiert, wie in der folgenden Gleichung beschrieben (Punkte als homogene Vektoren dargestellt):

$$\sum_i \left\| \begin{pmatrix} \texttt{Qx}[i] \\ \texttt{Qy}[i] \\ 1 \end{pmatrix} - \texttt{HomMat2D} \cdot \begin{pmatrix} \texttt{Px}[i] \\ \texttt{Py}[i] \\ 1 \end{pmatrix} \right\|^2 = \text{minimal}$$

HomMat2D kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. affine_trans_image, verwendet werden.

Achtung

Es ist zu beachten, dass homogene Transformationsmatrizen sich auf ein allgemeines rechtshändiges mathematisches Koordinatensystem beziehen. Falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, ist zu beachten, dass die Zeilenkoordinaten in den x-Koordinaten und die Spaltenkoordinaten in den y-Koordinaten übergeben werden müssen. Die Übergabereihenfolge von Zeilen- und Spaltenkoordinaten entspricht also der üblichen Reihenfolge (Row,Column). Diese Konvention ist unerlässlich, um bei der Transformation von Bilddaten ein rechtshändiges Koordinatensystem zu erhalten, so dass z.B. insbesondere Rotationen in der mathematisch korrekten Drehrichtung ausgeführt werden.

Weiterhin ist zu beachten, dass, falls eine homogene Transformationsmatrix zur Transformation von Bildern, Regionen, XLD-Konturen oder anderen Daten, die aus Bildern extrahiert wurden, verwendet werden soll, angenommen wird, dass der Ursprung des Koordinatensystems der homogenen Transformationsmatrix in der linken oberen Ecke des Pixels liegt. Die Bildverarbeitungsoperatoren, die Punktkoordinaten zurückliefern, nehmen hingegen ein Koordinatensystem an, in dem der Ursprung in der Mitte eines Pixels liegt. Daher muss, um eine konsistente homogene Transformationsmatrix zu erhalten, 0.5 zu den Punktkoordinaten addiert werden, bevor die Transformation berechnet wird.

Parameter

- ▷ **Px** (input_control) . . . . . point.x-array ⇝ *real*
  X-Koordinaten der originalen Punkte.
- ▷ **Py** (input_control) . . . . . point.y-array ⇝ *real*
  Y-Koordinaten der originalen Punkte.
- ▷ **Qx** (input_control) . . . . . point.x-array ⇝ *real*
  X-Koordinaten der transformierten Punkte.
- ▷ **Qy** (input_control) . . . . . point.y-array ⇝ *real*
  Y-Koordinaten der transformierten Punkte.
- ▷ **HomMat2D** (output_control) . . . . . hom_mat2d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

affine_trans_image, affine_trans_image_size, affine_trans_region, affine_trans_contour_xld, affine_trans_polygon_xld, affine_trans_point_2d

Alternativen

vector_to_hom_mat2d, vector_to_aniso, vector_to_rigid

Siehe auch

vector_field_to_hom_mat2d, point_line_to_hom_mat2d

Modul

Foundation

27.2 3D-Transformationen

affine_trans_point_3d (: : HomMat3D, Px, Py, Pz : Qx, Qy, Qz)

Wendet eine beliebige affine 3D-Transformation auf Punkte an.

affine_trans_point_3d wendet eine beliebige affine 3D-Transformation (Skalierung, Drehung, Verschiebung) auf die Eingabepunkte (Px,Py,Pz) an und liefert die Ergebnispunkte in (Qx,Qy,Qz) zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in HomMat3D übergeben wird. Dies entspricht der folgenden Gleichung (Ein- und Ausgabepunkte als homogene Vektoren dargestellt):

$$\begin{pmatrix} Qx \\ Qy \\ Qz \\ 1 \end{pmatrix} = HomMat3D \cdot \begin{pmatrix} Px \\ Py \\ Pz \\ 1 \end{pmatrix}$$

Die Transformationsmatrix kann mit Hilfe der Operatoren hom_mat3d_identity, hom_mat3d_scale, hom_mat3d_rotate, hom_mat3d_translate etc. aufgebaut werden oder das Resultat von pose_to_hom_mat3d sein.

Falls zum Beispiel HomMat3D eine starre Transformation ist, also aus Rotation und Translation besteht, werden die Punkte folgendermaßen transformiert:

$$\begin{pmatrix} Qx \\ Qy \\ Qz \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0\,0\,0 & 1 \end{bmatrix} \cdot \begin{pmatrix} Px \\ Py \\ Pz \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} \cdot \begin{pmatrix} Px \\ Py \\ Pz \end{pmatrix} + \mathbf{T} \\ 1 \end{pmatrix}$$

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Px** (input_control) . . . . . point3d.x(-array) ⇝ *real* / integer
  Eingabepunkt(e) (x-Koordinate).
  Defaultwert: 64
  Wertevorschläge: Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Py** (input_control) . . . . . point3d.y(-array) ⇝ *real* / integer
  Eingabepunkt(e) (y-Koordinate).
  Defaultwert: 64
  Wertevorschläge: Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Pz** (input_control) . . . . . point3d.z(-array) ⇝ *real* / integer
  Eingabepunkt(e) (z-Koordinate).
  Defaultwert: 64
  Wertevorschläge: Pz ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Qx** (output_control) . . . . . point3d.x(-array) ⇝ *real*
  Ausgabepunkt(e) (x-Koordinate).
- ▷ **Qy** (output_control) . . . . . point3d.y(-array) ⇝ *real*
  Ausgabepunkt(e) (y-Koordinate).
- ▷ **Qz** (output_control) . . . . . point3d.z(-array) ⇝ *real*
  Ausgabepunkt(e) (z-Koordinate).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `affine_trans_point_3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`hom_mat3d_translate`, `hom_mat3d_translate_local`, `hom_mat3d_scale`, `hom_mat3d_scale_local`, `hom_mat3d_rotate`, `hom_mat3d_rotate_local`

Nachfolger

`hom_mat3d_translate`, `hom_mat3d_translate_local`, `hom_mat3d_scale`, `hom_mat3d_scale_local`, `hom_mat3d_rotate`, `hom_mat3d_rotate_local`

Modul

Foundation

deserialize_hom_mat3d (: : SerializedItemHandle : HomMat3D)

Deserialisieren einer homogenen 3D-Transformationsmatrix.

`deserialize_hom_mat3d` deserialisiert eine homogene 3D-Transformationsmatrix, welche mit `serialize_hom_mat3d` serialisiert wurde (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte Transformationsmatrix wird in dem Handle `SerializedItemHandle` übergeben. Die deserialisierten Werte werden in einer dafür automatisch erzeugten Transformationsmatrix mit dem Handle `HomMat3D` gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **HomMat3D** (output_control) ..... hom_mat3d ⇝ *real*
  Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deserialize_hom_mat3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fread_serialized_item`, `receive_serialized_item`, `serialize_hom_mat3d`

Modul

Foundation

hom_mat3d_compose (: : HomMat3DLeft, HomMat3DRight : HomMat3DCompose)

Multipliziert zwei homogene 3D-Transformationsmatrizen.

`hom_mat3d_compose` generiert eine neue 3D-Transformationsmatrix durch Multiplikation der zwei Eingabe-Transformationsmatrizen:

$$\texttt{HomMat3DCompose} = \texttt{HomMat3DLeft} \cdot \texttt{HomMat3DRight}$$

Falls zum Beispiel die Eingabematrizen starre Transformationen darstellen, also Transformationen bestehend aus Rotation und Translation, wird die Ergebnismatrix wie folgt bestimmt:

$$\texttt{HomMat3DCompose} = \begin{bmatrix} \mathbf{R}_l & \mathbf{T}_l \\ 0\,0\,0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}_r & \mathbf{T}_r \\ 0\,0\,0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_l \cdot \mathbf{R}_r & \mathbf{R}_l \cdot \mathbf{T}_r + \mathbf{T}_l \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix}$$

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & rc & td \\ re & rf & rg & th \\ ri & rj & rk & tl \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl] gespeichert. Es ist aber auch möglich, volle 4×4 Matrizen, die eine projektive 3D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat3DLeft** (input_control) . . . . . hom_mat3d ⇝ *real*
  Linke Eingabe-Transformationsmatrix.
- ▷ **HomMat3DRight** (input_control) . . . . . hom_mat3d ⇝ *real*
  Rechte Eingabe-Transformationsmatrix.
- ▷ **HomMat3DCompose** (output_control) . . . . . hom_mat3d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `hom_mat3d_compose` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat3d_compose`, `hom_mat3d_translate`, `hom_mat3d_translate_local`, `hom_mat3d_scale`, `hom_mat3d_scale_local`, `hom_mat3d_rotate`, `hom_mat3d_rotate_local`, `pose_to_hom_mat3d`

Nachfolger

`hom_mat3d_compose`, `hom_mat3d_translate`, `hom_mat3d_translate_local`, `hom_mat3d_scale`, `hom_mat3d_scale_local`, `hom_mat3d_rotate`, `hom_mat3d_rotate_local`

Alternativen

`pose_compose`, `dual_quat_compose`

Siehe auch

`affine_trans_point_3d`, `hom_mat3d_identity`, `hom_mat3d_rotate`, `hom_mat3d_translate`, `pose_to_hom_mat3d`, `hom_mat3d_to_pose`

Modul

Foundation

hom_mat3d_determinant (: : HomMat3D : Determinant)

Berechnet die Determinante einer homogenen 3D-Transformationsmatrix.

hom_mat3d_determinant berechnet die Determinante der homogenen 3D-Transformationsmatrix, die in HomMat3D übergeben wird, und gibt die Determinante in Determinant zurück.

Parameter

▷ **HomMat3D** (input_control) ..... hom_mat3d ⇝ *real*
Eingabe-Transformationsmatrix.

▷ **Determinant** (output_control) ..... real ⇝ *real*
Determinante der Eingabematrix.

Ergebnis

hom_mat3d_determinant liefert immer den Wert TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat3d_translate, hom_mat3d_translate_local, hom_mat3d_scale, hom_mat3d_scale_local, hom_mat3d_rotate, hom_mat3d_rotate_local, vector_to_hom_mat3d

Modul

Foundation

hom_mat3d_identity (: : : HomMat3DIdentity)

Erzeugt die homogene Transformationsmatrix der identischen 3D-Abbildung.

hom_mat3d_identity erzeugt die homogene Transformationsmatrix HomMat3DIdentity, die die identische 3D-Abbildung beschreibt:

$$\text{HomMat3DIdentity} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Demnach wird die Matrix HomMat3DIdentity als das Tupel [1,0,0,0,0,1,0,0,0,0,1,0] abgespeichert.

Parameter

▷ **HomMat3DIdentity** (output_control) ..... hom_mat3d ⇝ *real*
Transformationsmatrix.

Ergebnis

hom_mat3d_identity liefert immer den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

hom_mat3d_translate, hom_mat3d_translate_local, hom_mat3d_scale, hom_mat3d_scale_local, hom_mat3d_rotate, hom_mat3d_rotate_local

Alternativen

pose_to_hom_mat3d

Modul

Foundation

hom_mat3d_invert (: : HomMat3D : HomMat3DInvert)

Invertiert eine homogene 3D-Transformationsmatrix.

hom_mat3d_invert invertiert die homogene 3D-Transformationsmatrix, die in HomMat3D übergeben wird. Die resultierende Matrix wird in HomMat3DInvert zurückgeliefert.

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & rc & td \\ re & rf & rg & th \\ ri & rj & rk & tl \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl] gespeichert. Es ist aber auch möglich, volle 4×4 Matrizen, die eine projektive 3D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat3D** (input_control) ..... hom_mat3d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **HomMat3DInvert** (output_control) ..... hom_mat3d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

hom_mat3d_invert liefert den Wert TRUE, falls die Parameterwerte korrekt sind und die Eingabematrix invertierbar ist. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat3d_translate, hom_mat3d_translate_local, hom_mat3d_scale, hom_mat3d_scale_local, hom_mat3d_rotate, hom_mat3d_rotate_local, pose_to_hom_mat3d

Nachfolger

hom_mat3d_translate, hom_mat3d_translate_local, hom_mat3d_scale, hom_mat3d_scale_local, hom_mat3d_rotate, hom_mat3d_rotate_local, hom_mat3d_to_pose

Alternativen

pose_invert, dual_quat_conjugate

Siehe auch

affine_trans_point_3d, hom_mat3d_identity, hom_mat3d_rotate, hom_mat3d_translate, pose_to_hom_mat3d, hom_mat3d_to_pose, hom_mat3d_compose

Modul

Foundation

hom_mat3d_rotate (: : HomMat3D, Phi, Axis, Px, Py, Pz : HomMat3DRotate)

Fügt eine Rotation zu einer homogenen 3D-Transformationsmatrix hinzu.

hom_mat3d_rotate fügt zur homogenen 3D-Transformationsmatrix HomMat3D eine Rotation um den Winkel Phi um die im Parameter Axis übergebene Achse hinzu. Die Achse kann dabei entweder durch die Strings 'x', 'y' oder 'z' oder als Vektor [x,y,z] in Form eines Tupels festgelegt werden.

Die Rotation wird beschrieben durch die 3×3 Rotationsmatrix $\mathbf{R}$. Sie wird relativ zum globalen (d.h. feststehenden) Koordinatensystem ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

Axis = *'x'*:

$$\texttt{HomMat3DRotate} = \begin{bmatrix} & & & 0 \\ & \mathbf{R}_x & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat3D} \qquad \mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\texttt{Phi}) & -\sin(\texttt{Phi}) \\ 0 & \sin(\texttt{Phi}) & \cos(\texttt{Phi}) \end{bmatrix}$$

Axis = *'y'*:

$$\texttt{HomMat3DRotate} = \begin{bmatrix} & & & 0 \\ & \mathbf{R}_y & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat3D} \qquad \mathbf{R}_y = \begin{bmatrix} \cos(\texttt{Phi}) & 0 & \sin(\texttt{Phi}) \\ 0 & 1 & 0 \\ -\sin(\texttt{Phi}) & 0 & \cos(\texttt{Phi}) \end{bmatrix}$$

Axis = *'z'*:

$$\texttt{HomMat3DRotate} = \begin{bmatrix} & & & 0 \\ & \mathbf{R}_z & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat3D} \qquad \mathbf{R}_z = \begin{bmatrix} \cos(\texttt{Phi}) & -\sin(\texttt{Phi}) & 0 \\ \sin(\texttt{Phi}) & \cos(\texttt{Phi}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Axis = *[x,y,z]*:

$$\texttt{HomMat3DRotate} = \begin{bmatrix} & & & 0 \\ & \mathbf{R}_a & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat3D} \qquad \mathbf{R}_a = \mathbf{u} \cdot \mathbf{u}^{\mathrm{T}} + cos(\texttt{Phi}) \cdot (\mathbf{I} - \mathbf{u} \cdot \mathbf{u}^{\mathrm{T}}) + sin(\texttt{Phi}) \cdot \mathbf{S}$$

$$\mathbf{u} = \frac{\texttt{Axis}}{\|\texttt{Axis}\|} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \qquad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{bmatrix}$$

Der Punkt (Px,Py,Pz) ist dabei der Fixpunkt der Rotation, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat3DRotate transformiert. Dieses Verhalten wird erreicht, indem zur Eingabe-Transformationsmatrix zuerst eine Translation hinzugefügt wird, die den Fixpunkt in den Ursprung des globalen Koordinatensystems verschiebt. Dann wird die Rotation hinzugefügt, und zuletzt eine Translation, die den Fixpunkt wieder zurück in seine Ausgangsposition verschiebt. Dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat3DRotate} = \begin{bmatrix} 1 & 0 & 0 & +\texttt{Px} \\ 0 & 1 & 0 & +\texttt{Py} \\ 0 & 0 & 1 & +\texttt{Pz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} & & & 0 \\ & \mathbf{R} & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -\texttt{Px} \\ 0 & 1 & 0 & -\texttt{Py} \\ 0 & 0 & 1 & -\texttt{Pz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat3D}$$

Um die Transformation im lokalen Koordinatensystem, das durch HomMat3D beschrieben ist, durchzuführen, kann hom_mat3d_rotate_local verwendet werden.

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird

im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & rc & td \\ re & rf & rg & th \\ ri & rj & rk & tl \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl] gespeichert. Es ist aber auch möglich, volle 4×4 Matrizen, die eine projektive 3D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Phi** (input_control) . . . . . angle.rad ⇝ *real* / integer
  Rotationswinkel.
  Defaultwert: 0.78
  Wertevorschläge: `Phi` ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
  Typischer Wertebereich: 0 ≤ `Phi` ≤ 6.28318530718
- ▷ **Axis** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
  Achse, um die gedreht wird.
  Defaultwert: 'x'
  Wertevorschläge: `Axis` ∈ {'x', 'y', 'z'}
- ▷ **Px** (input_control) . . . . . point3d.x ⇝ *real* / integer
  Fixpunkt der Transformation (x-Koordinate).
  Defaultwert: 0
  Wertevorschläge: `Px` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Py** (input_control) . . . . . point3d.y ⇝ *real* / integer
  Fixpunkt der Transformation (y-Koordinate).
  Defaultwert: 0
  Wertevorschläge: `Py` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Pz** (input_control) . . . . . point3d.z ⇝ *real* / integer
  Fixpunkt der Transformation (z-Koordinate).
  Defaultwert: 0
  Wertevorschläge: `Pz` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat3DRotate** (output_control) . . . . . hom_mat3d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `hom_mat3d_rotate` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat3d_identity`, `hom_mat3d_translate`, `hom_mat3d_scale`, `hom_mat3d_rotate`

Nachfolger

`hom_mat3d_translate`, `hom_mat3d_scale`, `hom_mat3d_rotate`

Siehe auch

`hom_mat3d_invert`, `hom_mat3d_identity`, `hom_mat3d_rotate_local`, `pose_to_hom_mat3d`, `hom_mat3d_to_pose`, `hom_mat3d_compose`

Modul

Foundation

hom_mat3d_rotate_local (: : HomMat3D, Phi, Axis : HomMat3DRotate)

Fügt eine Rotation zu einer homogenen 3D-Transformationsmatrix hinzu.

hom_mat3d_rotate_local fügt zur homogenen 3D-Transformationsmatrix HomMat3D eine Rotation um den Winkel Phi um die im Parameter Axis übergebene Achse hinzu. Die Achse kann dabei entweder durch die Strings 'x', 'y' oder 'z' oder als Vektor [x,y,z] in Form eines Tupels festgelegt werden.

Die Rotation wird beschrieben durch die 3×3 Rotationsmatrix $\mathbf{R}$. Im Gegensatz zu hom_mat3d_rotate wird sie relativ zum lokalen Koordinatensystem, das durch HomMat3D beschrieben ist, ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

Axis = *'x'*:

$$\text{HomMat3DRotate} = \text{HomMat3D} \cdot \begin{bmatrix} & & & 0 \\ & \mathbf{R}_x & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\text{Phi}) & -\sin(\text{Phi}) \\ 0 & \sin(\text{Phi}) & \cos(\text{Phi}) \end{bmatrix}$$

Axis = *'y'*:

$$\text{HomMat3DRotate} = \text{HomMat3D} \cdot \begin{bmatrix} & & & 0 \\ & \mathbf{R}_y & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R}_y = \begin{bmatrix} \cos(\text{Phi}) & 0 & \sin(\text{Phi}) \\ 0 & 1 & 0 \\ -\sin(\text{Phi}) & 0 & \cos(\text{Phi}) \end{bmatrix}$$

Axis = *'z'*:

$$\text{HomMat3DRotate} = \text{HomMat3D} \cdot \begin{bmatrix} & & & 0 \\ & \mathbf{R}_z & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R}_z = \begin{bmatrix} \cos(\text{Phi}) & -\sin(\text{Phi}) & 0 \\ \sin(\text{Phi}) & \cos(\text{Phi}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Axis = *[x,y,z]*:

$$\text{HomMat3DRotate} = \text{HomMat3D} \cdot \begin{bmatrix} & & & 0 \\ & \mathbf{R}_a & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R}_a = \mathbf{u} \cdot \mathbf{u}^{\mathrm{T}} + cos(\text{Phi}) \cdot (\mathbf{I} - \mathbf{u} \cdot \mathbf{u}^{\mathrm{T}}) + sin(\text{Phi}) \cdot \mathbf{S}$$

$$\mathbf{u} = \frac{\text{Axis}}{\|\text{Axis}\|} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \qquad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{bmatrix}$$

Der Fixpunkt der Rotation ist der Ursprung des lokalen Koordinatensystems, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat3DRotate transformiert.

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & rc & td \\ re & rf & rg & th \\ ri & rj & rk & tl \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl] gespeichert. Es ist aber auch möglich, volle 4×4 Matrizen, die eine projektive 3D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Phi** (input_control) . . . . . angle.rad ⇝ *real* / integer
  Rotationswinkel.
  Defaultwert: 0.78
  Wertevorschläge: Phi ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
  Typischer Wertebereich: 0 ≤ Phi ≤ 6.28318530718
- ▷ **Axis** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
  Achse, um die gedreht wird.
  Defaultwert: 'x'
  Wertevorschläge: Axis ∈ {'x', 'y', 'z'}
- ▷ **HomMat3DRotate** (output_control) . . . . . hom_mat3d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert hom_mat3d_rotate_local den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat3d_identity, hom_mat3d_translate_local, hom_mat3d_scale_local, hom_mat3d_rotate_local

Nachfolger

hom_mat3d_translate_local, hom_mat3d_scale_local, hom_mat3d_rotate_local

Siehe auch

hom_mat3d_invert, hom_mat3d_identity, hom_mat3d_rotate, pose_to_hom_mat3d, hom_mat3d_to_pose, hom_mat3d_compose

Modul

Foundation

hom_mat3d_scale (: : HomMat3D, Sx, Sy, Sz, Px, Py, Pz : HomMat3DScale)

Fügt eine Skalierung zu einer homogenen 3D-Transformationsmatrix hinzu.

hom_mat3d_scale fügt zur homogenen 3D-Transformationsmatrix HomMat3D eine Skalierung um die Skalierungsfaktoren Sx, Sy und Sz hinzu und liefert die resultierende Matrix in HomMat3DScale zurück. Die Skalierung wird beschrieben durch die 3×3 Skalierungsmatrix **S**. Sie wird relativ zum globalen (d.h. feststehenden) Koordinatensystem ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\mathtt{HomMat3DScale} = \begin{bmatrix} & & 0 \\ & \mathbf{S} & 0 \\ & & 0 \\ 0\,0\,0 & & 1 \end{bmatrix} \cdot \mathtt{HomMat3D} \qquad \mathbf{S} = \begin{bmatrix} \mathtt{Sx} & 0 & 0 \\ 0 & \mathtt{Sy} & 0 \\ 0 & 0 & \mathtt{Sz} \end{bmatrix}$$

Der Punkt (Px,Py,Pz) ist dabei der Fixpunkt der Skalierung, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat3DScale transformiert. Dieses Verhalten wird erreicht, indem zur Eingabe-Transformationsmatrix zuerst eine Translation hinzugefügt wird, die den Fixpunkt in den Ursprung des globalen Koordinatensystems verschiebt. Dann wird die Skalierung hinzugefügt, und zuletzt eine Translation, die den Fixpunkt wieder zurück in seine Ausgangsposition verschiebt. Dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat3DScale} = \begin{bmatrix} 1 & 0 & 0 & +\texttt{Px} \\ 0 & 1 & 0 & +\texttt{Py} \\ 0 & 0 & 1 & +\texttt{Pz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} & & & 0 \\ & \mathbf{S} & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -\texttt{Px} \\ 0 & 1 & 0 & -\texttt{Py} \\ 0 & 0 & 1 & -\texttt{Pz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat3D}$$

Um die Transformation im lokalen Koordinatensystem, das durch `HomMat3D` beschrieben ist, durchzuführen, kann `hom_mat3d_scale_local` verwendet werden.

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & rc & td \\ re & rf & rg & th \\ ri & rj & rk & tl \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl] gespeichert. Es ist aber auch möglich, volle 4×4 Matrizen, die eine projektive 3D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Sx** (input_control) . . . . . number ⇝ *real* / integer
  Skalierungsfaktor in x-Richtung.
  Defaultwert: 2
  Wertevorschläge: `Sx` ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  Restriktion: Sx != 0
- ▷ **Sy** (input_control) . . . . . number ⇝ *real* / integer
  Skalierungsfaktor in y-Richtung.
  Defaultwert: 2
  Wertevorschläge: `Sy` ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  Restriktion: Sy != 0
- ▷ **Sz** (input_control) . . . . . number ⇝ *real* / integer
  Skalierungsfaktor in z-Richtung.
  Defaultwert: 2
  Wertevorschläge: `Sz` ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  Restriktion: Sz != 0
- ▷ **Px** (input_control) . . . . . point3d.x ⇝ *real* / integer
  Fixpunkt der Transformation (x-Koordinate).
  Defaultwert: 0
  Wertevorschläge: `Px` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Py** (input_control) . . . . . point3d.y ⇝ *real* / integer
  Fixpunkt der Transformation (y-Koordinate).
  Defaultwert: 0
  Wertevorschläge: `Py` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Pz** (input_control) . . . . . point3d.z ⇝ *real* / integer
  Fixpunkt der Transformation (z-Koordinate).
  Defaultwert: 0
  Wertevorschläge: `Pz` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat3DScale** (output_control) . . . . . hom_mat3d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

`hom_mat3d_scale` liefert den Wert TRUE, falls alle drei Skalierungsfaktoren verschieden von 0 sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat3d_identity, hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Nachfolger

hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Siehe auch

hom_mat3d_invert, hom_mat3d_identity, hom_mat3d_scale_local, pose_to_hom_mat3d, hom_mat3d_to_pose, hom_mat3d_compose

Modul

Foundation

hom_mat3d_scale_local (: : HomMat3D, Sx, Sy, Sz : HomMat3DScale)

Fügt eine Skalierung zu einer homogenen 3D-Transformationsmatrix hinzu.

hom_mat3d_scale_local fügt zur homogenen 3D-Transformationsmatrix HomMat3D eine Skalierung um die Skalierungsfaktoren Sx, Sy und Sz hinzu und liefert die resultierende Matrix in HomMat3DScale zurück. Die Skalierung wird beschrieben durch die 3×3 Skalierungsmatrix **S**. Im Gegensatz zu hom_mat3d_scale wird sie relativ zum lokalen Koordinatensystem, das durch HomMat3D beschrieben ist, ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat3DScale} = \texttt{HomMat3D} \cdot \begin{bmatrix} & & 0 \\ & \mathbf{S} & 0 \\ & & 0 \\ 0\,0\,0 & & 1 \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} \texttt{Sx} & 0 & 0 \\ 0 & \texttt{Sy} & 0 \\ 0 & 0 & \texttt{Sz} \end{bmatrix}$$

Der Fixpunkt der Skalierung ist der Ursprung des lokalen Koordinatensystems, d.h. dieser Punkt bleibt unverändert, wenn man ihn mit HomMat3DScale transformiert.

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & rc & td \\ re & rf & rg & th \\ ri & rj & rk & tl \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl] gespeichert. Es ist aber auch möglich, volle 4×4 Matrizen, die eine projektive 3D-Transformation darstellen können, zu verarbeiten.

Parameter

▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
Eingabe-Transformationsmatrix.

▷ **Sx** (input_control) . . . . . number ⇝ *real* / integer
Skalierungsfaktor in x-Richtung.
Defaultwert: 2
Wertevorschläge: Sx ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
Restriktion: Sx != 0

▷ **Sy** (input_control) . . . . . number ⇝ *real* / integer
Skalierungsfaktor in y-Richtung.
Defaultwert: 2
Wertevorschläge: Sy ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
Restriktion: Sy != 0

▷ **Sz** (input_control) . . . . . number ⇝ *real* / integer
Skalierungsfaktor in z-Richtung.
Defaultwert: 2
Wertevorschläge: Sz ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
Restriktion: Sz != 0

▷ **HomMat3DScale** (output_control) . . . . . hom_mat3d ⇝ *real*
Ausgabe-Transformationsmatrix.

Ergebnis

hom_mat3d_scale_local liefert den Wert TRUE, falls alle drei Skalierungsfaktoren verschieden von 0 sind. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

hom_mat3d_identity, hom_mat3d_translate_local, hom_mat3d_scale_local, hom_mat3d_rotate_local

Nachfolger

hom_mat3d_translate_local, hom_mat3d_scale_local, hom_mat3d_rotate_local

Siehe auch

hom_mat3d_invert, hom_mat3d_identity, hom_mat3d_scale, pose_to_hom_mat3d, hom_mat3d_to_pose, hom_mat3d_compose

Modul

Foundation

hom_mat3d_to_pose (: : HomMat3D : Pose)

Konvertieren einer homogenen Transformationsmatrix in eine 3D-Lage (Pose).

hom_mat3d_to_pose konvertiert eine homogene Transformationsmatrix HomMat3D in eine 3D-Lage Pose mit Typ-Code 0. Details zu 3D-Lagen und den zugehörigen Transformationsmatrizen findet sich bei create_pose.

Ein typischer Anwendungsfall von hom_mat3d_to_pose ist, dass eine 3D-Lage zuvor mit pose_to_hom_mat3d in eine homogene Transformationsmatrix konvertiert wurde, um sie z.B. mit hom_mat3d_rotate oder hom_mat3d_translate weiter zu transformieren, und dann wieder zurück in ein 3D-Lage konvertiert werden soll, um die als Eingabe für Operatoren wie image_points_to_world_plane zu nutzen.

Achtung

hom_mat3d_to_pose unterstützt nur rigide Transformationen in HomMat3D.

Parameter

▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
Homogene Transformationsmatrix.

▷ **Pose** (output_control) . . . . . pose ⇝ *real* / integer
äquivalente 3D-Lage (Pose).
Parameteranzahl: 7

Beispiel

```
* Calibrate camera.
calibrate_cameras (CalibDataID,Error)
* Get reference pose (pose 2 of calibration object 0).
get_calib_data (CalibDataID, 'calib_obj_pose',\
```

```
                [0,2], 'pose', ObjInCameraPose)
*  Convert pose to homogeneous transformation matrix.
pose_to_hom_mat3d(ObjInCameraPose, cam_H_cal)
*  Rotate it 90 degrees around its y-axis to obtain a world coordinate system
*  whose y- and z-axis lie in the plane of the calibration plate while the
*  x-axis point 'upwards': cam_H_w = cam_H_cal * RotY(90).
hom_mat3d_identity(HomMat3DIdent)
hom_mat3d_rotate(HomMat3DIdent, rad(90), 'y', 0, 0, 0, \
                 HomMat3DRotateY)
hom_mat3d_compose(cam_H_cal, HomMat3DRotateY, cam_H_w)
* Convert transformed matrix back to pose.
hom_mat3d_to_pose (cam_H_w, Pose)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `hom_mat3d_to_pose` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat3d_rotate`, `hom_mat3d_translate`, `hom_mat3d_invert`

Nachfolger

`camera_calibration`, `write_pose`, `disp_caltab`, `sim_caltab`

Siehe auch

`create_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`, `write_pose`, `read_pose`, `pose_to_hom_mat3d`, `project_3d_point`, `get_line_of_sight`, `hom_mat3d_rotate`, `hom_mat3d_translate`, `hom_mat3d_invert`, `affine_trans_point_3d`

Modul

Foundation

hom_mat3d_translate (: : HomMat3D, Tx, Ty, Tz : HomMat3DTranslate)

Fügt eine Translation zu einer homogenen 3D-Transformationsmatrix hinzu.

`hom_mat3d_translate` fügt zur homogenen 3D-Transformationsmatrix `HomMat3D` eine Translation um den Vektor $\mathbf{T}$ = (`Tx`,`Ty`,`Tz`) hinzu und liefert die resultierende Matrix in `HomMat3DTranslate` zurück. Die Translation wird relativ zum globalen (d.h. feststehenden) Koordinatensystem ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat3DTranslate} = \begin{bmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & \mathbf{T} \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat3D} \qquad \mathbf{T} = \begin{pmatrix} \texttt{Tx} \\ \texttt{Ty} \\ \texttt{Tz} \end{pmatrix}$$

Um die Transformation im lokalen Koordinatensystem, das durch `HomMat3D` beschrieben ist, durchzuführen, kann `hom_mat3d_translate_local` verwendet werden.

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & rc & td \\ re & rf & rg & th \\ ri & rj & rk & tl \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl] gespeichert. Es ist aber auch möglich, volle 4×4 Matrizen, die eine projektive 3D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Tx** (input_control) . . . . . point3d.x ⇝ *real* / integer
  Verschiebung in x-Richtung.
  Defaultwert: 64
  Wertevorschläge: `Tx` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Ty** (input_control) . . . . . point3d.y ⇝ *real* / integer
  Verschiebung in y-Richtung.
  Defaultwert: 64
  Wertevorschläge: `Ty` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Tz** (input_control) . . . . . point3d.z ⇝ *real* / integer
  Verschiebung in z-Richtung.
  Defaultwert: 64
  Wertevorschläge: `Tz` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat3DTranslate** (output_control) . . . . . hom_mat3d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `hom_mat3d_translate` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat3d_identity`, `hom_mat3d_translate`, `hom_mat3d_scale`, `hom_mat3d_rotate`

Nachfolger

`hom_mat3d_translate`, `hom_mat3d_scale`, `hom_mat3d_rotate`

Siehe auch

`hom_mat3d_invert`, `hom_mat3d_identity`, `hom_mat3d_translate_local`, `pose_to_hom_mat3d`, `hom_mat3d_to_pose`, `hom_mat3d_compose`

Modul

Foundation

hom_mat3d_translate_local (: : HomMat3D, Tx, Ty, Tz : HomMat3DTranslate)

Fügt eine Translation zu einer homogenen 3D-Transformationsmatrix hinzu.

`hom_mat3d_translate_local` fügt zur homogenen 3D-Transformationsmatrix `HomMat3D` eine Translation um den Vektor **T** = (`Tx`,`Ty`,`Tz`) hinzu und liefert die resultierende Matrix in `HomMat3DTranslate` zurück. Im Gegensatz zu `hom_mat3d_translate` wird die Translation relativ zum lokalen Koordinatensystem, das durch `HomMat3D` beschrieben ist, ausgeführt; dies entspricht der folgenden Kette von Transformationsmatrizen:

$$\texttt{HomMat3DTranslate} = \texttt{HomMat3D} \cdot \begin{bmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & \mathbf{T} \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T} = \begin{pmatrix} \texttt{Tx} \\ \texttt{Ty} \\ \texttt{Tz} \end{pmatrix}$$

Achtung

Homogene Transformationsmatrizen werden zeilenweise in Form eines Tupels abgespeichert; die letzte Zeile wird im Normalfall nicht gespeichert, da sie für alle affinen Transformationsmatrizen identisch ist. Zum Beispiel wird die Matrix

$$\begin{bmatrix} ra & rb & rc & td \\ re & rf & rg & th \\ ri & rj & rk & tl \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

als das Tupel [ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl] gespeichert. Es ist aber auch möglich, volle 4×4 Matrizen, die eine projektive 3D-Transformation darstellen können, zu verarbeiten.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Eingabe-Transformationsmatrix.
- ▷ **Tx** (input_control) . . . . . point3d.x ⇝ *real* / integer
  Verschiebung in x-Richtung.
  Defaultwert: 64
  Wertevorschläge: `Tx` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Ty** (input_control) . . . . . point3d.y ⇝ *real* / integer
  Verschiebung in y-Richtung.
  Defaultwert: 64
  Wertevorschläge: `Ty` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **Tz** (input_control) . . . . . point3d.z ⇝ *real* / integer
  Verschiebung in z-Richtung.
  Defaultwert: 64
  Wertevorschläge: `Tz` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- ▷ **HomMat3DTranslate** (output_control) . . . . . hom_mat3d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `hom_mat3d_translate_local` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat3d_identity`, `hom_mat3d_translate_local`, `hom_mat3d_scale_local`, `hom_mat3d_rotate_local`

Nachfolger

`hom_mat3d_translate_local`, `hom_mat3d_scale_local`, `hom_mat3d_rotate_local`

Siehe auch

`hom_mat3d_invert`, `hom_mat3d_identity`, `hom_mat3d_translate`, `pose_to_hom_mat3d`, `hom_mat3d_to_pose`, `hom_mat3d_compose`

Modul

Foundation

hom_mat3d_transpose (: : HomMat3D : HomMat3DTranspose)

Transponiert eine homogene 3D-Transformationsmatrix.

`hom_mat3d_transpose` transponiert die homogene 3D-Transformationsmatrix, die in `HomMat3D` übergeben wird. Die Ergebnismatrix `HomMat3DTranspose` ist immer eine 4×4 Matrix, selbst wenn die Eingabematrix als 3×4 Matrix dargestellt ist.

Parameter

▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
Eingabe-Transformationsmatrix.

▷ **HomMat3DTranspose** (output_control) . . . . . hom_mat3d ⇝ *real*
Ausgabe-Transformationsmatrix.

Ergebnis

`hom_mat3d_transpose` liefert immer den Wert TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`hom_mat3d_translate`, `hom_mat3d_translate_local`, `hom_mat3d_scale`, `hom_mat3d_scale_local`, `hom_mat3d_rotate`, `hom_mat3d_rotate_local`, `vector_to_hom_mat3d`

Nachfolger

`hom_mat3d_compose`, `hom_mat3d_invert`

Modul

Foundation

pose_to_hom_mat3d (: : Pose : HomMat3D)

Konvertieren einer 3D-Lage (Pose) in eine homogene Transformationsmatrix.

`pose_to_hom_mat3d` konvertiert eine 3D-Lage `Pose`, z.B. die externen Kameraparameter, in die äquivalente homogene Transformationsmatrix `HomMat3D`. Details zu 3D-Lagen und den zugehörigen Transformationsmatrizen findet sich bei `create_pose`.

Ein typischer Anwendungsfall von `pose_to_hom_mat3d` ist, dass eine 3D-Lage weiter transformiert, z.B. mittels `hom_mat3d_rotate` oder `hom_mat3d_translate` rotiert oder verschoben werden soll. Im Falle der externen Kameraparameter kann dies nötig sein, wenn der Kalibrierkörper nicht so platziert werden kann, dass sein Koordinatensystem mit dem gewünschten Weltkoordinatensystem zusammenfällt.

Parameter

▷ **Pose** (input_control) . . . . . pose ⇝ *real* / integer
3D-Lage (Pose).
Parameteranzahl: 7

▷ **HomMat3D** (output_control) . . . . . hom_mat3d ⇝ *real*
Äquivalente homogene Transformationsmatrix.

Beispiel

```
* Calibrate camera.
calibrate_cameras (CalibDataID,Error)
* Get reference pose (pose 2 of calibration object 0).
get_calib_data (CalibDataID, 'calib_obj_pose',\
                [0,2], 'pose', ObjInCameraPose)
*  Convert pose to homogeneous transformation matrix.
pose_to_hom_mat3d(ObjInCameraPose, cam_H_cal)
*  Rotate it 90 degrees around its y-axis to obtain a world coordinate system
*  whose y- and z-axis lie in the plane of the calibration plate while the
*  x-axis point 'upwards': cam_H_w = cam_H_cal * RotY(90).
hom_mat3d_identity(HomMat3DIdent)
hom_mat3d_rotate(HomMat3DIdent, rad(90), 'y', 0, 0, 0, \
```

```
                HomMat3DRotateY)
hom_mat3d_compose(cam_H_cal, HomMat3DRotateY, cam_H_w)
* Convert transformed matrix back to pose.
hom_mat3d_to_pose (cam_H_w, Pose)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `pose_to_hom_mat3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`camera_calibration`, `read_pose`

Nachfolger

`affine_trans_point_3d`, `hom_mat3d_invert`, `hom_mat3d_translate`, `hom_mat3d_rotate`, `hom_mat3d_to_pose`

Alternativen

`pose_to_dual_quat`

Siehe auch

`create_pose`, `camera_calibration`, `write_pose`, `read_pose`, `hom_mat3d_to_pose`, `project_3d_point`, `get_line_of_sight`, `hom_mat3d_rotate`, `hom_mat3d_translate`, `hom_mat3d_invert`, `affine_trans_point_3d`

Modul

Foundation

projective_trans_hom_point_3d (: : HomMat3D, Px, Py, Pz, Pw : Qx, Qy, Qz, Qw)

Projiziert einen homogenen 3D-Punkt mittels einer projektiven Transformationsmatrix.

`projective_trans_hom_point_3d` wendet die homogene projektive Transformationsmatrix `HomMat3D` auf alle homogenen Eingabepunkte (`Px`,`Py`,`Pz`,`Pw`) an und liefert ein Tupel von homogenen Ergebnispunkten (`Qx`,`Qy`,`Qz`,`Qw`) zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in `HomMat3D` übergeben wird. Dies entspricht der folgenden Gleichung (Ein- und Ausgabepunkte als homogene Vektoren dargestellt):

$$\begin{pmatrix} Qx \\ Qy \\ Qz \\ Qw \end{pmatrix} = HomMat3D \cdot \begin{pmatrix} Px \\ Py \\ Pz \\ Pw \end{pmatrix}$$

Um die homogenen Koordinaten in euklidische Koordinaten zu transformieren, muss durch `Qw` geteilt werden:

$$\begin{pmatrix} Qx \\ Qy \\ Qz \\ Qw \end{pmatrix} = HomMat3D \cdot \begin{pmatrix} Px \\ Py \\ Pz \\ Pw \end{pmatrix}$$

Dies kann direkt mit `projective_trans_point_3d` erreicht werden. Daher ist `projective_trans_hom_point_3d` hauptsächlich in Situationen nützlich, in denen die Ergebinspunkte in der unendlich fernen Ebene liegen können, d.h. $Qw = 0$ besitzen können, und somit die obige Division nicht ausgeführt werden kann.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **Px** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (x-Koordinate).
- ▷ **Py** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (y-Koordinate).
- ▷ **Pz** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (z-Koordinate).
- ▷ **Pw** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (w-Koordinate).
- ▷ **Qx** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (x-Koordinate).
- ▷ **Qy** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (y-Koordinate).
- ▷ **Qz** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (z-Koordinate).
- ▷ **Qw** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (w-Koordinate).

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`vector_to_hom_mat3d`

Alternativen

`projective_trans_point_3d`

Modul

Foundation

projective_trans_point_3d (: : HomMat3D, Px, Py, Pz : Qx, Qy, Qz)

Projiziert einen 3D-Punkt mittels einer projektiven Transformationsmatrix.

`projective_trans_point_3d` wendet die homogene projektive Transformationsmatrix `HomMat3D` auf alle Eingabepunkte (`Px`,`Py`,`Pz`) an und liefert ein Tupel von Ergebnispunkten (`Qx`,`Qy`,`Qz`) zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in `HomMat3D` übergeben wird. Dies entspricht den folgenden Gleichungen (Ein- und Ausgabepunkte als homogene Vektoren dargestellt):

$$\begin{pmatrix} Tx \\ Ty \\ Tz \\ Tw \end{pmatrix} = \texttt{HomMat3D} \cdot \begin{pmatrix} \texttt{Px} \\ \texttt{Py} \\ \texttt{Pz} \\ 1 \end{pmatrix}$$

`projective_trans_point_3d` transformiert hierauf die homogenen Koordinaten in euklidische Koordinaten, indem durch Tw geteilt wird:

$$\begin{pmatrix} \texttt{Qx} \\ \texttt{Qy} \\ \texttt{Qz} \end{pmatrix} = \begin{pmatrix} \frac{Tx}{Tw} \\ \frac{Ty}{Tw} \\ \frac{Tz}{Tw} \end{pmatrix}$$

Falls bei dieser Transformation ein Punkt in der unendlich fernen Ebene erzeugt wird ($Tw = 0$), wird eine Fehlermeldung zurückgeliefert. Falls dies nicht erwünscht ist, kann `projective_trans_hom_point_3d` verwendet werden.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Homogene projektive Transformationsmatrix.
- ▷ **Px** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (x-Koordinate).
- ▷ **Py** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (y-Koordinate).
- ▷ **Pz** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabepunkt (z-Koordinate).
- ▷ **Qx** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (x-Koordinate).
- ▷ **Qy** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (y-Koordinate).
- ▷ **Qz** (output_control) . . . . . number(-array) ⇝ *real*
  Ausgabepunkt (z-Koordinate).

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Vorgänger

`vector_to_hom_mat3d`

Alternativen

`projective_trans_hom_point_3d`

Modul

Foundation

serialize_hom_mat3d (: : HomMat3D : SerializedItemHandle)

Serialisieren einer homogenen 3D-Transformationsmatrix.

`serialize_hom_mat3d` serialisiert die Daten einer homogenen 3D-Transformationsmatrix (siehe `fwrite_serialized_item` für eine Einführung in die Grundlagen der Serialisierung). Die Transformationsmatrix wird in dem Handle `HomMat3D` übergeben. Die serialisierte Transformationsmatrix wird in dem Handle `SerializedItemHandle` zurückgegeben und kann mit `deserialize_hom_mat3d` wieder deserialisiert werden.

Parameter

- ▷ **HomMat3D** (input_control) . . . . . hom_mat3d ⇝ *real*
  Transformationsmatrix.
- ▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_hom_mat3d` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_hom_mat3d`

Modul

Foundation

vector_to_hom_mat3d (: : TransformationType, Px, Py, Pz, Qx, Qy, Qz : HomMat3D)

Erzeugt eine Näherung für eine 3D-Transformation aus Punktkorrespondenzen.

`vector_to_hom_mat3d` erzeugt aus Punktkorrespondenzen eine Näherung für eine affine oder projektive 3D-Transformation und liefert sie in der homogenen Transformationsmatrix `HomMat3D` zurück.

Der Typ der zu berechnenden 3D-Transformation wird mit `TransformationType` festgelegt. Für `TransformationType` = *'rigid'* wird eine starre 3D-Transformation (eine Rotation und eine Translation), für `TransformationType` = *'similarity'* eine Ähnlichkeitsabbildung (eine gleichförmige Skalierung, eine Rotation und eine Translation), für `TransformationType` = *'affine'* eine allgemeine affine 3D-Transformation und für `TransformationType` = *'projective'* eine projektive 3D-Transformation berechnet.

Die minimal benötigte Anzahl an Punktkorrespondenzen ist 3 für `TransformationType` = *'rigid'*, 3 für `TransformationType` = *'similarity'*, 4 für `TransformationType` = *'affine'* und 5 für `TransformationType` = *'projective'*.

Die Punktkorrespondenzen werden in den Tupeln (`Px`,`Py`,`Pz`) und (`Qx`,`Qy`,`Qz`) übergeben, wobei korrespondierende Punkte an denselben Indexpositionen stehen müssen. Die Transformation ist überbestimmt, falls mehr als die minimale Anzahl an Punktkorrespondenzen übergeben wird. In diesem Fall ist die zurückgelieferte Transformation diejenige, die die Abstände zwischen den transformierten Eingabepunkten (`Px`,`Py`,`Pz`) und den Zielpunkten (`Qx`,`Qy`,`Qz`) minimiert, wie in der folgenden Gleichung beschrieben (Punkte als homogene Vektoren dargestellt):

$$\sum_i \left\| \begin{pmatrix} \mathtt{Qx}[i] \\ \mathtt{Qy}[i] \\ \mathtt{Qz}[i] \\ 1 \end{pmatrix} - \mathtt{HomMat3D} \cdot \begin{pmatrix} \mathtt{Px}[i] \\ \mathtt{Py}[i] \\ \mathtt{Pz}[i] \\ 1 \end{pmatrix} \right\|^2 = \text{minimal}$$

`HomMat3D` kann direkt mit den Operatoren, die Daten mit affinen 3D-Transformationen transformieren, z.B. `affine_trans_point_3d`, verwendet werden.

Parameter

- ▷ **TransformationType** (input_control) ..... string ⇝ *string*
  Typ der zu berechnenden Transformation.
  Defaultwert: 'rigid'
  Werteliste: `TransformationType` ∈ {'rigid', 'similarity', 'affine', 'projective'}
- ▷ **Px** (input_control) ..... point3d.x-array ⇝ *real*
  X-Koordinaten der originalen Punkte.
- ▷ **Py** (input_control) ..... point3d.y-array ⇝ *real*
  Y-Koordinaten der originalen Punkte.
- ▷ **Pz** (input_control) ..... point3d.z-array ⇝ *real*
  Z-Koordinaten der originalen Punkte.
- ▷ **Qx** (input_control) ..... point3d.x-array ⇝ *real*
  X-Koordinaten der transformierten Punkte.
- ▷ **Qy** (input_control) ..... point3d.x-array ⇝ *real*
  Y-Koordinaten der transformierten Punkte.
- ▷ **Qz** (input_control) ..... point3d.z-array ⇝ *real*
  Z-Koordinaten der transformierten Punkte.
- ▷ **HomMat3D** (output_control) ..... hom_mat3d ⇝ *real*
  Ausgabe-Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`hom_mat3d_to_pose`, `affine_trans_point_3d`

Modul

Foundation

27.3 Duale Quaternionen

Dieses Kapitel enthält Operatoren für die Verwendung von dualen Quaternionen.

Einführung in Duale Quaternionen

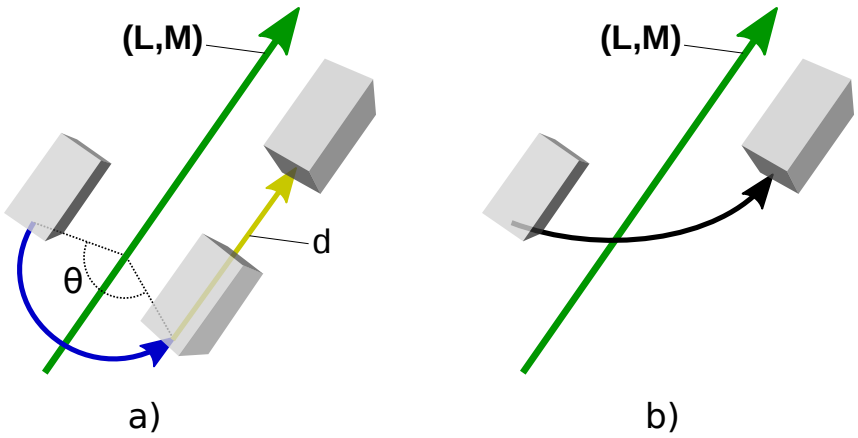
Eine duale Quaternion $\hat{q} = q_r + \varepsilon q_d$ besteht aus zwei Quaternionen q_r und q_d, wobei q_r den Realteil, q_d den Dualteil und ε die duale Einheit ($\varepsilon^2 = 0$) darstellt. Jede Quaternion $q = w + ix + jy + kz$ besteht wiederum aus dem Skalarteil w und dem Vektorteil $\mathbf{v} = (x, y, z)$, wobei $(1, i, j, k)$ die Basis-Elemente des Quaternionen-Vektorraums darstellen.

Darstellung von Dualen Quaternionen in HALCON

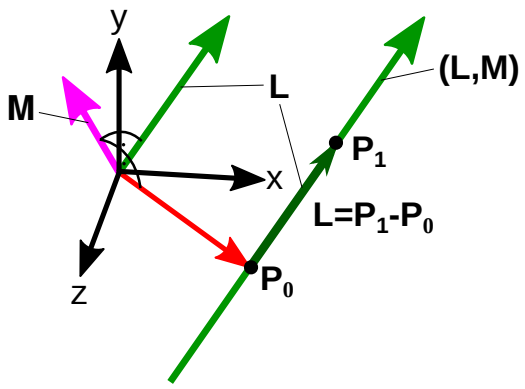
In HALCON werden duale Quaternionen als Tupel mit acht Elementen dargestellt $[w_r, x_r, y_r, z_r, w_d, x_d, y_d, z_d]$. Dabei sind w_r und $\mathbf{v}_r = (x_r, y_r, z_r)$ der Skalar- und Vektorteil des Realteils und w_d und $\mathbf{v}_d = (x_d, y_d, z_d)$ der Skalar- und Vektorteil des Dualteils.

Duale Einheitsquaternionen, Starre 3D-Abbildungen und Schraubungen

Im Unterschied zu einer Einheitsquaternion, die eine 3D-Rotation beschreibt, beschreibt eine duale Einheitsquaternion eine starre 3D-Abbildung, d.h. eine 3D-Rotation und eine 3D-Translation. Damit stellen duale Einheitsquaternionen neben 3D-Lagen und homogenen 3D-Transformationsmatrizen eine alternative Repräsentation für starre 3D-Abbildungen dar. Im Vergleich zu Transformationsmatrizen mit 12 Elementen stellen duale Quaternionen mit 8 Elementen eine kompaktere Repräsentation dar. Wie Transformationsmatrizen können auch duale Quaternionen sehr einfach kombiniert werden um mehrere Transformationen zu verketten. Darüber hinaus bieten sie die Möglichkeit, weich zwischen zwei starren 3D-Abbildungen zu interpolieren und Linien in 3D sehr effizient zu transformieren. Jede starre 3D-Abbildung kann als Schraubung dargestellt werden:



a) Eine starre 3D-Abbildung... b) kann durch eine Schraubung dargestellt werden.



Moment $\mathbf{M}$ der Schraubenachse.

Die Schraubungsparameter sind:

- Rotationswinkel der Schraubung θ
- Translation der Schraubung d
- Richtung $\mathbf{L} = (L_x, L_y, L_z)^T$ der Schraubenachse mit $\|\mathbf{L}\| = 1$
- Moment $\mathbf{M} = (M_x, M_y, M_z)^T$ der Schraubenachse mit $\mathbf{L} \cdot \mathbf{M} = 0$

Eine Schraubung setzt sich zusammen aus einer Rotation um die Schraubenachse, die durch $\mathbf{L}$ und $\mathbf{M}$ definiert ist, um den Winkel θ und einer Translation um d entlang dieser Achse. Die Position der Schraubenachse im Raum ist definiert durch ihr Moment $\mathbf{M}$ bezüglich des Ursprungs des Koordinatensystems. Der Vektor $\mathbf{M}$ steht senkrecht auf der Richtung der Schraubenachse $\mathbf{L}$ und senkrecht auf einem Vektor vom Koordinatensystemursprung zu einem Punkt $\mathbf{P}_0$ auf der Schraubenachse. Das Moment wird berechnet durch das Vektorprodukt

$$\mathbf{M} = \mathbf{P}_0 \times \mathbf{L}$$

Damit ist $\mathbf{M}$ der Normalenvektor der Ebene, die durch die Schraubungsachse und den Ursprung aufgespannt wird. Zu beachten ist, dass

$$\mathbf{P}_0 = \mathbf{L} \times \mathbf{M}$$

der Punkt auf der Schraubenachse $(\mathbf{L}, \mathbf{M})$ mit dem kürzesten Abstand zum Koordinatensystemursprung ist.

Die Elemente einer dualen Einheitsquaternion stehen mit den Schraubungsparametern einer starren 3D-Abbildung wie folgt in Beziehung:

$$\hat{q} = \begin{pmatrix} \cos\frac{\theta}{2} \\ \mathbf{L}\sin\frac{\theta}{2} \end{pmatrix} + \varepsilon \begin{pmatrix} -\frac{d}{2}\sin\frac{\theta}{2} \\ \mathbf{M}\sin\frac{\theta}{2} + \mathbf{L}\frac{d}{2}\cos\frac{\theta}{2} \end{pmatrix},$$

Zu beachten ist, dass $\hat{q}$ und $-\hat{q}$ die selbe starre 3D-Abbildung beschreiben. Außerdem ist die Inverse einer dualen Einheitsquaternion gleich ihrer Konjugierten, d.h. $\hat{q}^{-1} = \bar{\hat{q}}$ (eine Beschreibung der Konjugierten einer dualen Quaternion ist bei `dual_quat_conjugate` zu finden).

3D-Linien und Plücker-Koordinaten

Linien in 3D können durch duale Vektoren ausgedrückt werden. Ein dualer Vektor kann als duale Quaternion mit Skalarteil 0 angesehen werden. Eine starre 3D-Abbildung, die durch eine duale Einheitsquaternion dargestellt werden kann, steht in einfacher Beziehung mit den Parametern einer Schraubung um eine Schraubenachse. Wie oben beschrieben ist die Schraubenachse definiert durch ihre Richtung $\mathbf{L}$ mit $\|\mathbf{L}\| = 1$ und ihr Moment $\mathbf{M}$ bezüglich des Ursprungs des Koordinatensystem mit $\mathbf{L} \cdot \mathbf{M} = 0$. Auf die gleiche Weise kann eine 3D-Linie durch

ihre Richtung $\mathbf{L}$ mit $\|\mathbf{L}\| = 1$ und ihr Moment $\mathbf{M}$ dargestellt werden. Die sechs Parameter von $\mathbf{L}$ und $\mathbf{M}$ sind die Plücker-Koordinaten der 3D-Linie.

Folglich ist eine Linie $\hat{l}$, die durch eine duale Quaternion mit Skalarteil 0 repräsentiert wird,

$$\hat{l} = l_r + \varepsilon l_d = \begin{pmatrix} 0 \\ L_x \\ L_y \\ L_z \end{pmatrix} + \varepsilon \begin{pmatrix} 0 \\ M_x \\ M_y \\ M_z \end{pmatrix}.$$

deserialize_dual_quat (: : SerializedItemHandle : DualQuaternion)

Deserialisieren einer serialisierten dualen Quaternion.

deserialize_dual_quat deserialisiert eine duale Quaternion, welche mit serialize_dual_quat serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte duale Quaternion wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in DualQuaternion gespeichert.

Parameter

- ▷ **SerializedItemHandle** (input_control) . . . . . serialized_item ⇝ *handle*
  Handle des serialisierten Elements.
- ▷ **DualQuaternion** (output_control) . . . . . dual_quaternion ⇝ *real* / integer
  Duale Quaternion.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_dual_quat den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_quat

Nachfolger

dual_quat_to_hom_mat3d, dual_quat_to_pose, dual_quat_compose

Modul

Foundation

dual_quat_compose (: : DualQuaternionLeft,
DualQuaternionRight : DualQuaternionComposed)

Multiplikation zweier dualer Quaternionen.

Der Operator dual_quat_compose multipliziert die zwei dualen Quaternionen DualQuaternionLeft und DualQuaternionRight und gibt das Produkt in DualQuaternionComposed zurück.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Die Multiplikation der dualen Quaternionen $\hat{p}$ und $\hat{q}$ ist $\hat{p}\hat{q} = p_r q_r + \varepsilon(p_r q_d + p_d q_r)$.

Informationen zur Multiplikation von Quaternionen sind in der Beschreibung von quat_compose zu finden.

Zu beachten ist, dass die Multiplikation zweier dualer Quaternionen nicht kommutativ ist, d.h. $\hat{p}\hat{q} \neq \hat{q}\hat{p}$.

Falls DualQuaternionLeft und DualQuaternionRight duale Einheitsquaternionen sind und damit starre 3D-Abbildungen darstellen, entspricht ihre Multiplikation der Multiplikation ihrer entsprechenden homogenen

Transformationsmatrizen. Daher kann dual_quat_compose zur Hintereinanderschaltung zweier starrer 3D-Abbildungen verwendet werden analog zu pose_compose oder hom_mat3d_compose.

Parameter

▷ **DualQuaternionLeft** (input_control) . . . . . dual_quaternion(-array) ⇝ *real*
Linke duale Quaternion.

▷ **DualQuaternionRight** (input_control) . . . . . dual_quaternion(-array) ⇝ *real*
Rechte duale Quaternion.

▷ **DualQuaternionComposed** (output_control) . . . . . dual_quaternion(-array) ⇝ *real*
Produkt der dualen Quaternionen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

screw_to_dual_quat

Nachfolger

dual_quat_to_hom_mat3d, dual_quat_to_pose, dual_quat_to_screw

Alternativen

pose_compose, hom_mat3d_compose

Siehe auch

dual_quat_interpolate, pose_to_dual_quat, dual_quat_normalize, dual_quat_conjugate, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, quat_compose

Modul

Foundation

dual_quat_conjugate (: : DualQuaternion : DualQuaternionConjugate)

Konjugation einer dualen Quaternion.

Der Operator dual_quat_conjugate berechnet die Konjugierte DualQuaternionConjugate der dualen Eingabequaternion DualQuaternion.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Die Konjugierte einer dualen Quaternion $\hat{q} = q_r + \varepsilon q_d$ ist gegeben durch $\bar{\hat{q}} = \bar{q}_r + \varepsilon \bar{q}_d$, wobei $\bar{q}_r$ und $\bar{q}_d$ die konjugierten Quaternionen q_r und q_d darstellen.

Für weitere Informationen zur Konjugation von Quaternionen siehe quat_conjugate.

Falls es sich bei DualQuaternion um eine duale Einheitsquaternion handelt und DualQuaternion damit eine starre 3D-Abbildung darstellt, ist deren Inverse gleich ihrer Konjugierten, d.h. $\hat{q}^{-1} = \bar{\hat{q}}$. Folglich stellt DualQuaternionConjugate die inverse starre 3D-Abbildung von DualQuaternion dar. Daher kann dual_quat_conjugate analog zu pose_invert oder hom_mat3d_invert dazu verwendet werden, um eine starre 3D-Abbildung zu invertieren.

Parameter

▷ **DualQuaternion** (input_control) . . . . . dual_quaternion(-array) ⇝ *real*
Duale Quaternion.

▷ **DualQuaternionConjugate** (output_control) . . . . . dual_quaternion(-array) ⇝ *real*
Konjugierte der dualen Quaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

dual_quat_compose, pose_to_dual_quat, screw_to_dual_quat

Nachfolger

dual_quat_compose, dual_quat_to_hom_mat3d, dual_quat_to_screw, dual_quat_interpolate, dual_quat_to_pose

Alternativen

pose_invert, hom_mat3d_invert

Siehe auch

dual_quat_normalize, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, quat_conjugate

Modul

Foundation

dual_quat_interpolate (: : DualQuaternionStart, DualQuaternionEnd, InterpPos : DualQuaternionInterpolated)

Interpolation von zwei dualen Quaternionen.

Der Operator dual_quat_interpolate interpoliert zwischen den beiden dualen Quaternionen DualQuaternionStart und DualQuaternionEnd an der Position InterpPos.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Für Interpolationen muss InterpPos innerhalb des Intervalls $[0,1]$ liegen, wenngleich auch Werte außerhalb dieses Intervalls möglich sind. Letzteres entspricht dann einer Extrapolation. Im Fall InterpPos $=$ 0 entspricht die interpolierte duale Quaternion DualQuaternionInterpolated dem Startpunkt DualQuaternionStart der Interpolation. Im Fall InterpPos $=$ 1 entspricht DualQuaternionInterpolated dem Endpunkt DualQuaternionEnd der Interpolation.

Die Interpolation wird durch eine lineare Schraubungsinterpolation (screw linear interpolation, ScLERP) durchgeführt. Sind sowohl DualQuaternionStart als auch DualQuaternionEnd duale Einheitsquaternionen, so ist auch DualQuaternionInterpolated eine duale Einheitsquaternion. dual_quat_interpolate kann also dazu verwendet werden, um weich zwischen zwei starren 3D-Abbildungen zu interpolieren.

Die Interpolation des Rotationsanteils zwischen DualQuaternionStart und DualQuaternionEnd ist identisch zur Interpolation von Quaternionen, d.h. der Schraubungswinkel wird linear interpoliert (siehe quat_interpolate). Die Interpolation des Translationsanteils erfolgt durch die lineare Interpolation der Translation der Schraubung.

Es ist möglich, für InterpPos ein Tupel mit mehreren Werten zu übergeben. In diesem Fall wird in DualQuaternionInterpolated für jeden Wert in InterpPos eine interpolierte duale Quaternion zurückgegeben. Dies ist effizienter als dual_quat_interpolate mehrfach mit einem einzelnen Wert für InterpPos aufzurufen.

Parameter

▷ **DualQuaternionStart** (input_control) . . . . . dual_quaternion ⇝ *real*
Duale Quaternion als Startpunkt der Interpolation.

▷ **DualQuaternionEnd** (input_control) . . . . . dual_quaternion ⇝ *real*
Duale Quaternion als Endpunkt der Interpolation.

▷ **InterpPos** (input_control) . . . . . real(-array) ⇝ *real*
Interpolationsparameter.
Defaultwert: 0.5
Wertevorschläge: InterpPos ∈ {0.0, 0.25, 0.5, 0.75, 1.0}

▷ **DualQuaternionInterpolated** (output_control) . . . . . dual_quaternion(-array) ⇝ *real*
Interpolierte duale Quaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

pose_to_dual_quat, screw_to_dual_quat, dual_quat_normalize

Nachfolger

dual_quat_to_hom_mat3d, dual_quat_to_pose, dual_quat_to_screw

Siehe auch

dual_quat_compose, dual_quat_conjugate, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, quat_interpolate

Modul

Foundation

dual_quat_normalize (: : DualQuaternion : DualQuaternionNormalized)

Normalisierung einer dualen Quaternion.

Der Operator dual_quat_normalize normalisiert die duale Quaternion DualQuaternion und liefert die normalisierte duale Quaternion zurück, die auch als duale Einheitsquaternion bezeichnet wird.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Die Norm einer dualen Einheitsquaternion $\hat{q} = q_r + \varepsilon q_d$ ist 1, d.h. $\hat{q}\bar{\hat{q}} = 1$. Dies ist äquivalent zu den folgenden beiden Bedingungen:

$$\begin{aligned} \|q_r\| = q_r\bar{q}_r &= 1 \\ q_r\bar{q}_d + \bar{q}_r q_d &= 0, \end{aligned}$$

wobei $\bar{q}$ die konjugierte Quaternion von q darstellt (siehe quat_conjugate).

Eine starre 3D-Abbildung kann durch eine duale Einheitsquaternion repräsentiert werden.

Achtung

Für den Fall, dass die Norm des Realteils von DualQuaternion gleich 0 ist, liefert dual_quat_normalize den Fehlercode 9310 (H_ERR_DQ_ZERO_NORM) zurück, da in diesem Fall keine Normalisierung möglich ist.

Parameter

▷ **DualQuaternion** (input_control) . . . . . dual_quaternion(-array) ⇝ *real*
Duale Quaternion.

▷ **DualQuaternionNormalized** (output_control) . . . . . dual_quaternion(-array) ⇝ *real*
Duale Einheitsquaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

dual_quat_compose

Nachfolger

dual_quat_to_hom_mat3d

Siehe auch

dual_quat_interpolate, pose_to_dual_quat, screw_to_dual_quat, dual_quat_to_screw, dual_quat_to_pose, dual_quat_conjugate, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, quat_normalize

Modul

Foundation

dual_quat_to_hom_mat3d (: : DualQuaternion : HomMat3D)

Umwandlung einer dualen Einheitsquaternion in eine homogene Transformationsmatrix.

Der Operator dual_quat_to_hom_mat3d wandelt die duale Einheitsquaternion DualQuaternion, die eine starre 3D-Abbildung darstellt, in eine entsprechende homogene Transformationsmatrix HomMat3D um.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Der Rotationsanteil von HomMat3D wird aus dem Realteil der dualen Quaternion berechnet, wie in quat_to_hom_mat3d beschrieben. Der Translationsanteil $\mathbf{T}$ von HomMat3D wird aus dem Realteil und dem Dualteil von $\hat{q} = q_r + \varepsilon q_d$ berechnet:

$$\begin{aligned} p &= q_d q_r \\ \mathbf{T} &= 2.0\mathbf{v}_p, \end{aligned}$$

wobei $\mathbf{v}_p$ der Vektorteil der Quaternion p ist.

Achtung

dual_quat_to_hom_mat3d nimmt an, dass es sich bei DualQuaternion um eine duale Einheitsquaternion handelt und diese daher eine starre 3D-Abbildung darstellt. Andernfalls ist eine Umwandlung in eine homogene Transformationsmatrix nicht sinnvoll möglich.

Parameter

- ▷ **DualQuaternion** (input_control) ..... dual_quaternion ⇝ *real*
  Duale Einheitsquaternion.
- ▷ **HomMat3D** (output_control) ..... hom_mat3d ⇝ *real*
  Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

screw_to_dual_quat, dual_quat_interpolate, dual_quat_compose, dual_quat_normalize

Nachfolger

affine_trans_point_3d

Alternativen

dual_quat_to_pose, dual_quat_to_screw

Siehe auch

dual_quat_compose, pose_to_dual_quat, dual_quat_conjugate, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, quat_to_hom_mat3d

Modul

Foundation

dual_quat_to_screw (: : DualQuaternion, ScrewFormat : AxisDirectionX, AxisDirectionY, AxisDirectionZ, AxisMomentOrPointX, AxisMomentOrPointY, AxisMomentOrPointZ, Rotation, Translation)

Umwandlung einer dualen Einheitsquaternion in eine Schraubung.

Der Operator dual_quat_to_screw wandelt die duale Einheitsquaternion DualQuaternion, die eine starre 3D-Abbildung repräsentiert, in die Parameter einer Schraubung um.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Eine Schraubung kann beschrieben werden durch die Richtung ihrer Schraubenachse $\mathbf{L} = (L_x, L_y, L_z)^T$ mit $\|\mathbf{L}\| = 1$, das Moment der Schraubenachse $\mathbf{M} = (M_x, M_y, M_z)^T$ mit $\mathbf{L} \cdot \mathbf{M} = 0$, den Rotationswinkel der Schraubung θ und die Translation der Schraubung d.

Wird für ScrewFormat *'moment'* gewählt, werden die beschriebenen Schraubungsparameter in den entsprechenden Parametern AxisDirectionX, AxisDirectionY, AxisDirectionZ, AxisMomentOrPointX, AxisMomentOrPointY, AxisMomentOrPointZ, Rotation und Translation zurückgegeben.

Alternativ ist es auch möglich, anstatt des Moments der Schraubenachse den Punkt auf der Schraubenachse zurückzugeben, der den kürzesten Abstand zum Ursprung hat. Dazu muss für ScrewFormat *'point'* gewählt werden. In diesem Fall werden die Koordinaten des Punktes in AxisMomentOrPointX, AxisMomentOrPointY und AxisMomentOrPointZ zurückgegeben.

Achtung

dual_quat_to_screw nimmt an, dass es sich bei DualQuaternion um eine duale Einheitsquaternion handelt und diese daher eine starre 3D-Abbildung darstellt. Andernfalls ist eine Umwandlung in eine Schraubung nicht sinnvoll möglich. Zu beachten ist auch, dass die Schraubenachse für die Identitätsabbildung (keine Rotation und keine Translation), nicht definiert ist. In diesem Fall wird L auf $(1, 0, 0)^T$ gesetzt.

Parameter

- ▷ **DualQuaternion** (input_control) ..... dual_quaternion ⇝ *real*
  Duale Einheitsquaternion.
- ▷ **ScrewFormat** (input_control) ..... string ⇝ *string*
  Format der Schraubungsparameter.
  Defaultwert: 'moment'
  Werteliste: ScrewFormat ∈ {'moment', 'point'}
- ▷ **AxisDirectionX** (output_control) ..... point3d.x ⇝ *real*
  X-Komponente des Richtungsvektors der Schraubenachse.
- ▷ **AxisDirectionY** (output_control) ..... point3d.y ⇝ *real*
  Y-Komponente des Richtungsvektors der Schraubenachse.
- ▷ **AxisDirectionZ** (output_control) ..... point3d.z ⇝ *real*
  Z-Komponente des Richtungsvektors der Schraubenachse.
- ▷ **AxisMomentOrPointX** (output_control) ..... point3d.x ⇝ *real*
  X-Komponente des Momentenvektors oder eines Punktes der Schraubenachse.
- ▷ **AxisMomentOrPointY** (output_control) ..... point3d.y ⇝ *real*
  Y-Komponente des Momentenvektors oder eines Punktes der Schraubenachse.
- ▷ **AxisMomentOrPointZ** (output_control) ..... point3d.z ⇝ *real*
  Z-Komponente des Momentenvektors oder eines Punktes der Schraubenachse.
- ▷ **Rotation** (output_control) ..... angle.rad ⇝ *real*
  Rotationswinkel im Bogenmaß.
- ▷ **Translation** (output_control) ..... real ⇝ *real*
  Translation.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

dual_quat_compose, dual_quat_conjugate, dual_quat_interpolate

Alternativen

dual_quat_to_pose, dual_quat_to_hom_mat3d

Siehe auch

pose_to_dual_quat, dual_quat_normalize, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, screw_to_dual_quat

Modul

Foundation

dual_quat_trans_line_3d (: : DualQuaternion, LineFormat, LineDirectionX, LineDirectionY, LineDirectionZ, LineMomentOrPointX, LineMomentOrPointY, LineMomentOrPointZ : TransLineDirectionX, TransLineDirectionY, TransLineDirectionZ, TransLineMomentOrPointX, TransLineMomentOrPointY, TransLineMomentOrPointZ)

Transformieren einer 3D-Linie mit einer dualen Einheitsquaternion.

Der Operator dual_quat_trans_line_3d transformiert eine in Plücker-Koordinaten dargestellte 3D-Line mit einer starren 3D-Abbildung, die durch die duale Einheitsquaternion DualQuaternion gegeben ist.

Eine kurze Einführung in duale Quaternionen, Plücker-Koordinaten und die verwendete Notation findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Eine Linie in Plücker-Koordinaten ist gegeben durch ihre Richtung $\mathbf{L}$ mit $\|\mathbf{L}\| = 1$ und ihr Moment $\mathbf{M}$ mit $\mathbf{L} \cdot \mathbf{M} = 0$. Sie kann als duale Quaternion $\hat{l}$ mit Skalarteil 0 repräsentiert werden. Die Linie $\hat{l}$ kann durch die starre 3D-Abbildung, die durch eine duale Einheitsquaternion $\hat{q}$ repräsentiert ist, sehr einfach transformiert werden:

$$\hat{k} = \hat{q}\hat{l}\bar{\hat{q}}$$

Die resultierende duale Quaternion $\hat{k}$ hat ebenfalls einen Skalarteil von 0 und enthält direkt die Richtung und das Moment der transformierten Linie in ihren Vektorteilen.

Die Linienrichtung muss in LineDirectionX, LineDirectionY und LineDirectionZ übergeben werden. Zu beachten ist, dass die Länge des übergebenen Richtungsvektors 1 sein muss, um sinnvolle Ergebnisse zu erhalten, d.h. $L_x^2 + L_y^2 + L_z^2 = 1$.

Wird für LineFormat *'moment'* gewählt, muss in LineMomentOrPointX, LineMomentOrPointY und LineMomentOrPointZ das Moment der Linie übergeben werden.

Alternativ ist es auch möglich, anstatt des Moments der Linie einen beliebigen Punkt auf der Linie zu übergeben. Dazu muss für LineFormat *'point'* gewählt werden und in LineMomentOrPointX, LineMomentOrPointY und LineMomentOrPointZ müssen die Koordinaten des Punktes übergeben werden.

Die Parameter der transformierten Linie werden in TransLineDirectionX, TransLineDirectionY, TransLineDirectionZ, TransLineMomentOrPointX, TransLineMomentOrPointY und TransLineMomentOrPointZ im Format, das in LineFormat angegeben ist, zurückgegeben. Wird für LineFormat *'point'* gewählt, wird der Punkt auf der Linie mit dem kürzesten Abstand zum Koordinatensystemursprung zurückgegeben.

Achtung

dual_quat_trans_line_3d liefert nur dann sinnvolle Ergebnisse, wenn in DualQuaternion eine duale Einheitsquaternion übergeben wird und die Länge des Richtungsvektors der Linie 1 ist.

Parameter

▷ **DualQuaternion** (input_control) . . . . . dual_quaternion ⇝ *real* / integer
Duale Einheitsquaternion, die die Transformation darstellt.

▷ **LineFormat** (input_control) ..... string ⇝ *string*
Format der Linienparameter.
Defaultwert: 'moment'
Werteliste: LineFormat ∈ {'moment', 'point'}

▷ **LineDirectionX** (input_control) ..... point3d.x(-array) ⇝ *real* / integer
X-Komponente des Richtungsvektors der Linie.

▷ **LineDirectionY** (input_control) ..... point3d.y(-array) ⇝ *real* / integer
Y-Komponente des Richtungsvektors der Linie.

▷ **LineDirectionZ** (input_control) ..... point3d.z(-array) ⇝ *real* / integer
Z-Komponente des Richtungsvektors der Linie.

▷ **LineMomentOrPointX** (input_control) ..... point3d.x(-array) ⇝ *real* / integer
X-Komponente des Momentenvektors oder eines Punktes der Linie.

▷ **LineMomentOrPointY** (input_control) ..... point3d.y(-array) ⇝ *real* / integer
Y-Komponente des Momentenvektors oder eines Punktes der Linie.

▷ **LineMomentOrPointZ** (input_control) ..... point3d.z(-array) ⇝ *real* / integer
Z-Komponente des Momentenvektors oder eines Punktes der Linie.

▷ **TransLineDirectionX** (output_control) ..... point3d.x(-array) ⇝ *real* / integer
X-Komponente des Richtungsvektors der transformierten Linie.

▷ **TransLineDirectionY** (output_control) ..... point3d.y(-array) ⇝ *real* / integer
Y-Komponente des Richtungsvektors der transformierten Linie.

▷ **TransLineDirectionZ** (output_control) ..... point3d.z(-array) ⇝ *real* / integer
Z-Komponente des Richtungsvektors der transformierten Linie.

▷ **TransLineMomentOrPointX** (output_control) ..... point3d.x(-array) ⇝ *real* / integer
X-Komponente des Momentenvektors der transformierten Linie oder des zum Ursprung nächstliegenden Punktes auf der transformierten Linie.

▷ **TransLineMomentOrPointY** (output_control) ..... point3d.y(-array) ⇝ *real* / integer
Y-Komponente des Momentenvektors der transformierten Linie oder des zum Ursprung nächstliegenden Punktes auf der transformierten Linie.

▷ **TransLineMomentOrPointZ** (output_control) ..... point3d.z(-array) ⇝ *real* / integer
Z-Komponente des Momentenvektors der transformierten Linie oder des zum Ursprung nächstliegenden Punktes auf der transformierten Linie.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

dual_quat_compose, dual_quat_conjugate, pose_to_dual_quat

Alternativen

affine_trans_point_3d

Siehe auch

dual_quat_to_hom_mat3d, dual_quat_to_screw, dual_quat_to_pose, dual_quat_normalize, serialize_dual_quat, deserialize_dual_quat, quat_rotate_point_3d

Modul

Foundation

screw_to_dual_quat (: : ScrewFormat, AxisDirectionX, AxisDirectionY, AxisDirectionZ, AxisMomentOrPointX, AxisMomentOrPointY, AxisMomentOrPointZ, Rotation, Translation : DualQuaternion)

Umwandlung einer Schraubung in eine duale Einheitsquaternion.

Der Operator screw_to_dual_quat wandelt die Parameter einer Schraubung in die duale Einheitsquaternion DualQuaternion um, die eine starre 3D-Abbildung repräsentiert.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Eine Schraubung kann beschrieben werden durch die Richtung ihrer Schraubenachse $\mathbf{L} = (L_x, L_y, L_z)^T$ mit $\|\mathbf{L}\| = 1$, das Moment der Schraubenachse $\mathbf{M} = (M_x, M_y, M_z)^T$ mit $\mathbf{L} \cdot \mathbf{M} = 0$, den Rotationswinkel der Schraubung θ und die Translation der Schraubung d.

Wird für ScrewFormat *'moment'* gewählt, müssen die beschriebenen Schraubungsparameter in den entsprechenden Parametern AxisDirectionX, AxisDirectionY, AxisDirectionZ, AxisMomentOrPointX, AxisMomentOrPointY, AxisMomentOrPointZ, Rotation und Translation übergeben werden.

Alternativ ist es auch möglich, anstatt des Moments der Schraubenachse einen beliebigen Punkt auf der Schraubenachse zu übergeben. Dazu muss für ScrewFormat *'point'* gewählt werden und in AxisMomentOrPointX, AxisMomentOrPointY und AxisMomentOrPointZ müssen die Koordinaten des Punktes übergeben werden.

Achtung

screw_to_dual_quat nimmt an, dass der übergebene Richtungsvektor der Schraubenachse die Länge 1 hat, d.h. $\sqrt{L_x^2 + L_y^2 + L_z^2} = 1$. Andernfalls ist das resultierende duale Quaternion nicht sinnvoll.

Parameter

- ▷ **ScrewFormat** (input_control) . . . . . string ⇝ *string*
  Format der Schraubungsparameter.
  Defaultwert: 'moment'
  Werteliste: ScrewFormat ∈ {'moment', 'point'}
- ▷ **AxisDirectionX** (input_control) . . . . . point3d.x ⇝ *real* / integer
  X-Komponente des Richtungsvektors der Schraubenachse.
- ▷ **AxisDirectionY** (input_control) . . . . . point3d.y ⇝ *real* / integer
  Y-Komponente des Richtungsvektors der Schraubenachse.
- ▷ **AxisDirectionZ** (input_control) . . . . . point3d.z ⇝ *real* / integer
  Z-Komponente des Richtungsvektors der Schraubenachse.
- ▷ **AxisMomentOrPointX** (input_control) . . . . . point3d.x ⇝ *real* / integer
  X-Komponente des Momentenvektors oder eines Punktes der Schraubenachse.
- ▷ **AxisMomentOrPointY** (input_control) . . . . . point3d.y ⇝ *real* / integer
  Y-Komponente des Momentenvektors oder eines Punktes der Schraubenachse.
- ▷ **AxisMomentOrPointZ** (input_control) . . . . . point3d.z ⇝ *real* / integer
  Z-Komponente des Momentenvektors oder eines Punktes der Schraubenachse.
- ▷ **Rotation** (input_control) . . . . . angle.rad ⇝ *real* / integer
  Rotationswinkel im Bogenmaß.
- ▷ **Translation** (input_control) . . . . . real ⇝ *real* / integer
  Translation.
- ▷ **DualQuaternion** (output_control) . . . . . dual_quaternion ⇝ *real* / integer
  Duale Quaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

dual_quat_compose, dual_quat_conjugate, dual_quat_interpolate

Alternativen

pose_to_dual_quat

Siehe auch

dual_quat_to_hom_mat3d, pose_to_dual_quat, dual_quat_to_screw,

dual_quat_to_pose, dual_quat_normalize, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, axis_angle_to_quat

Modul

Foundation

serialize_dual_quat (: : DualQuaternion : SerializedItemHandle)

Serialisieren einer dualen Quaternion.

serialize_dual_quat serialisiert die Daten der in DualQuaternion übergebenen dualen Quaternion (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte duale Quaternion wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_dual_quat wieder deserialisiert werden.

Parameter

- ▷ **DualQuaternion** (input_control) ..... dual_quaternion ⇝ *real / integer*
  Duale Quaternion.
- ▷ **SerializedItemHandle** (output_control) ..... serialized_item ⇝ *handle*
  Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_dual_quat den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

axis_angle_to_quat

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_quat

Modul

Foundation

27.4 Posen

Eine 3D-Lage (Pose) beschreibt eine starre Transformation, d.h., eine Transformation die aus einer beliebigen Translation und Rotation besteht. In HALCON ist eine 3D-Lage ein Tupel mit 7 Parametern: 3 Parameter legen die Translation fest (TransX, TransY und TransZ), 3 Parameter beschreiben die Rotation (RotX, RotY und RotZ). Der letzte Parameter verschlüsselt die Reihenfolge der Translationen und Rotationen (sowie auch die Rotationsrichtung). Für weitere Informationen zu diesen Parametern wird auf create_pose und den „Solution Guide III-C - 3D Vision" verwiesen.

3D-Lagen werden typischerweise für zwei Zwecke verwendet: Erstens zur Beschreibung der Position und Orientierung eines Koordinatensystems relativ zu einem anderen (z.B. die 3D-Lage eines Objektkoordinatensystems relativ zum Kamerakoordinatensystem - kurz, die 3D-Lage des Objekts relativ zur Kamera) und zweitens zur Beschreibung der Transformation von Koordinaten zwischen Koordinatensystemen (z.B. die Transformation von Objektkoordinaten in Kamerakoordinaten).

Eine 3D-Lage, welche Punktkoordinaten von einem Koordinatensystem 1 ($cs1$) in ein Koordinatensystem 2 ($cs2$) transformiert, wird als ${}^{cs2}\mathbf{P}_{cs1}$ bezeichnet. Die entsprechende Transformation der Koordinaten eines Punktes in $cs1$ ($\mathbf{p}^{cs1}$) in $cs2$ ($\mathbf{p}^{cs2}$) wird wie folgt bezeichnet:

$$\mathbf{p}^{cs2} = {}^{cs2}\mathbf{P}_{cs1} \cdot \mathbf{p}^{cs1}.$$

Es wird darauf hingewiesen, dass nicht die 3D-Lage (als Tupel) sondern die damit beschriebene Transformation für die Multiplikation verwendet wird und die oben verwendete Notation bloß der Lesbarkeit dient. Die 3D-Lage ${}^{cs2}\mathbf{P}_{cs1}$ beschreibt die starre Transformation, welche durch die homogene Transformationsmatrix $\mathbf{H}({}^{cs2}\mathbf{P}_{cs1}) = {}^{cs2}\mathbf{H}_{cs1}$ verkörpert wird (siehe `create_pose` für weitere Informationen):

$$\begin{pmatrix} \mathbf{p}^{cs2} \\ 1 \end{pmatrix} = {}^{cs2}\mathbf{H}_{cs1} \cdot \begin{pmatrix} \mathbf{p}^{cs1} \\ 1 \end{pmatrix}.$$

Somit beschreibt die 3D-Lage ${}^{cs2}\mathbf{P}_{cs1}$ die Transformation von Punktkoordinaten von $cs1$ nach $cs2$. Zudem beschreibt die 3D-Lage wie oben erwähnt die Transformation des Koordinatensystems selbst, jedoch in umgekehrter Reihenfolge. ${}^{cs2}\mathbf{P}_{cs1}$ beschreibt, wie Koordinatensystem 2 transformiert werden muss, um Koordinatensystem 1 zu erhalten und damit die 3D-Lage von Koordinatensystem 2 relativ zu Koordinatensystem 1.

Mit dieser Notation können Posen wie homogene Matrizen verkettet werden, z.B.

$${}^{cs2}\mathbf{P}_{cs0} = {}^{cs2}\mathbf{P}_{cs1} \cdot {}^{cs1}\mathbf{P}_{cs0}.$$

Eine solche Verkettung kann z.B. mittels `pose_compose` durchgeführt werden.

convert_pose_type (: : PoseIn, OrderOfTransform, OrderOfRotation, ViewOfTransform : PoseOut)

Verändern des Darstellungstyps einer 3D-Lage (Pose).

`convert_pose_type` dient zum Konvertieren einer 3D-Lage `PoseIn` in die 3D-Lage `PoseOut` mit einem anderen Darstellungstypen. Siehe `create_pose` für eine detaillierte Beschreibung von 3D-Lagen, ihren Darstellungstypen und die Bedeutung der Parameter `OrderOfTransform`, `OrderOfRotation` und `ViewOfTransform`.

Beachten Sie, dass `convert_pose_type` nur die Darstellung einer 3D-Lage ändert, nicht jedoch die zu Grunde liegende starre Transformation.

Parameter

- ▷ **PoseIn** (input_control) ..... pose ⇝ *real* / integer
  Ursprüngliche 3D-Lage (Pose).
  Parameteranzahl: 7
- ▷ **OrderOfTransform** (input_control) ..... string ⇝ *string*
  Reihenfolge von Rotation und Translation.
  Defaultwert: 'Rp+T'
  Wertevorschläge: `OrderOfTransform` ∈ {'Rp+T', 'R(p-T)'}
- ▷ **OrderOfRotation** (input_control) ..... string ⇝ *string*
  Bedeutung der Rotationswerte.
  Defaultwert: 'gba'
  Wertevorschläge: `OrderOfRotation` ∈ {'gba', 'abg', 'rodriguez'}
- ▷ **ViewOfTransform** (input_control) ..... string ⇝ *string*
  Sichtweise der Transformation.
  Defaultwert: 'point'
  Wertevorschläge: `ViewOfTransform` ∈ {'point', 'coordinate_system'}
- ▷ **PoseOut** (output_control) ..... pose ⇝ *real* / integer
  3D-Transformationsvorschrift.
  Parameteranzahl: 7

Beispiel

```
* Define a pose.
create_pose (0.1, 0.1, 0.1, 90, 90, 90, 'Rp+T', 'gba', 'point', Pose)
* Convert pose to a pose with desired semantic.
convert_pose_type (Pose, 'Rp+T', 'abg', 'point', Pose2)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `convert_pose_type` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_pose`, `hom_mat3d_to_pose`, `camera_calibration`, `hand_eye_calibration`

Nachfolger

`write_pose`

Siehe auch

`create_pose`, `get_pose_type`, `write_pose`, `read_pose`

Modul

Foundation

create_pose (: : TransX, TransY, TransZ, RotX, RotY, RotZ, OrderOfTransform, OrderOfRotation, ViewOfTransform : Pose)

Erzeugen einer 3D-Lage (Pose).

`create_pose` erzeugt eine 3D-Lage `Pose`. Eine 3D-Lage beschreibt eine starre 3D-Transformation, d.h. eine Transformation bestehend aus einer beliebigen Translation und Rotation, mit 6 Parametern: `TransX`, `TransY` und `TransZ` beschreiben die Translation entlang der x-, y- und z-Achse, während `RotX`, `RotY` und `RotZ` die Rotation beschreiben.

3D-Lagen werden typischerweise für zwei Zwecke verwendet: Erstens zur Beschreibung der Position und Orientierung eines Koordinatensystems relativ zu einem anderen (z.B. die 3D-Lage eines Objektkoordinatensystems relativ zum Kamerakoordinatensystem - kurz, die 3D-Lage des Objekts relativ zur Kamera) und zweitens zur Beschreibung der Transformation von Koordinaten zwischen Koordinatensystemen (z.B. die Transformation von Objektkoordinaten in Kamerakoordinaten).

Darstellung der Orientierung (Rotation)

Eine 3D-Rotation um eine beliebige Achse kann mit drei Parametern in verschiedener Weise dargestellt werden. HALCON stellt drei davon über den Parameter `OrderOfRotation` zur Auswahl: Falls der Wert *'gba'* übergeben wird, wird die Rotation durch die folgende Kette von Rotationen beschrieben (siehe `hom_mat3d_rotate` für den Inhalt der Rotationsmatrizen $\mathbf{R}_x$, $\mathbf{R}_y$ und $\mathbf{R}_z$):

$$\mathbf{R}_{gba} = \mathbf{R}_x(\texttt{RotX}) \cdot \mathbf{R}_y(\texttt{RotY}) \cdot \mathbf{R}_z(\texttt{RotZ})$$

$\mathbf{R}_{gba}$ wird in der Fachliteratur auch Yaw-Pitch-Roll Konvention genannt. Bitte beachten Sie, dass diese Kette auf zwei Arten „gelesen“ werden kann: Wenn man von rechts startet, werden die Rotationen immer relativ zum globalen (unveränderten oder „alten“) Koordinatensystem ausgeführt. D.h., $\mathbf{R}_{gba}$ kann wie folgt gelesen werden: Erst wird um die z-Achse rotiert, dann um die „alte“ y-Achse und schließlich um die „alte“ x-Achse. Falls die Kette dagegen von links nach rechts gelesen wird, werden die Rotationen jeweils relativ zum lokalen („neuen“) Koordinatensystem ausgeführt. Dann liest sich $\mathbf{R}_{gba}$ wie folgt: Erst wird um die x-Achse rotiert, dann um die „neue“ y-Achse und schließlich um die „neu(st)e“ z-Achse.

Von rechts nach links gelesen, entspricht $\mathbf{R}_{gba}$ der folgenden Sequenz von Operatoraufrufen:

```
hom_mat3d_identity (HomMat3DIdent)
hom_mat3d_rotate (HomMat3DIdent, RotZ, 'z', 0, 0, 0, HomMat3DRotZ)
hom_mat3d_rotate (HomMat3DRotZ, RotY, 'y', 0, 0, 0, HomMat3DRotYZ)
hom_mat3d_rotate (HomMat3DRotYZ, RotX, 'x', 0, 0, 0, HomMat3DXYZ)
```

Dagegen entspricht die folgende Sequenz der Leserichtung von links nach rechts:

```
hom_mat3d_identity (HomMat3DIdent)
hom_mat3d_rotate_local (HomMat3DIdent, RotX, 'x', HomMat3DRotX)
hom_mat3d_rotate_local (HomMat3DRotX, RotY, 'y', HomMat3DRotXY)
hom_mat3d_rotate_local (HomMat3DRotXY, RotZ, 'z', HomMat3DXYZ)
```

$\mathbf{R}_{abg}$ wird in der Fachliteratur auch Roll-Pitch-Yaw Konvention genannt.

Falls in `OrderOfRotation` der Wert *'abg'* übergeben wird, wird die Rotation durch die folgende Transformationskette dargestellt:

$$\mathbf{R}_{abg} = \mathbf{R}_z(\texttt{RotZ}) \cdot \mathbf{R}_y(\texttt{RotY}) \cdot \mathbf{R}_x(\texttt{RotX})$$

Falls der Wert *'rodriguez'* in `OrderOfRotation` übergeben wird, werden die Rotationsparameter `RotX`, `RotY` und `RotZ` als die x-, y- und z-Komponente des so genannten Rodriguez-Rotationsvektors interpretiert. Die Richtung dieses Vektors definiert die (beliebige) Rotationsachse, seine Länge den Rotationswinkel (mathematisch positive Orientierung). HALCON verwendet eine Variation des Rodriguez-Vektors, in der die Länge des Vektors gleich dem Tangens des halben Winkels ist:

$$\mathbf{R}_{rodriguez} \quad = \quad rotate\ around \quad \begin{pmatrix} \texttt{RotX} \\ \texttt{RotY} \\ \texttt{RotZ} \end{pmatrix} \quad by \quad 2 \cdot \arctan(\sqrt{\texttt{RotX}^2 + \texttt{RotY}^2 + \texttt{RotZ}^2})$$

Bitte beachten Sie, dass diese Posen-Darstellung nicht immer eindeutig ist, d.h. mehrere Posen-Darstellungen beschreiben dieselbe homogene Transformationsmatrix. Beispielsweise für $\mathbf{R}_{gba}$ mit $b = \pm 90$ haben die folgenden zwei Posen die gleiche Darstellung als homogene Transformationsmatrix:

```
create_pose(0, 0, 0, 30 , 90, 54, 'Rp+T', 'gba', 'point', Pose1)
create_pose(0, 0, 0, 17, 90, 67, 'Rp+T', 'gba', 'point', Pose2)
```

Alternativ können Rotationen auch durch homogene Transformationsmatrizen oder Quaternionen (`axis_angle_to_quat`) dargestellt werden.

Entsprechende homogene Transformationsmatrix

Mit dem Operator `pose_to_hom_mat3d` erhält man die der 3D-Lage entsprechende homogene Transformationsmatrix. Gemäß der Standard-Definition ist dies die folgende Transformationsmatrix, welche in zwei separate Matrizen aufgespaltet werden kann, eine für die Translation ($\mathbf{H}(\mathbf{T})$) und eine für die Rotation ($\mathbf{H}(\mathbf{R})$):

$$\mathbf{H}_{pose} \quad = \quad \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0\,0\,0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}(\texttt{RotX}, \texttt{RotY}, \texttt{RotZ}) & \begin{matrix} \texttt{TransX} \\ \texttt{TransY} \\ \texttt{TransZ} \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & 0 & \texttt{TransX} \\ 0 & 1 & 0 & \texttt{TransY} \\ 0 & 0 & 1 & \texttt{TransZ} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}(\texttt{RotX}, \texttt{RotY}, \texttt{RotZ}) & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} = \mathbf{H}(\mathbf{T}) \cdot \mathbf{H}(\mathbf{R})$$

Koordinatentransformation

Die folgende Gleichung zeigt, wie ein Punkt mit Hilfe einer 3D-Lage (bzw. der entsprechenden homogenen Transformationsmatrix ${}^{cs2}\mathbf{H}_{cs1}$) von Koordinatensystem 1 (*cs*1) in Koordinatensystem 2 (*cs*2) transformiert werden kann (Ein- und Ausgabepunkte als homogene Vektoren dargestellt, vgl. `affine_trans_point_3d`). Beachten Sie, dass zur Transformation von *cs*1 in *cs*2 die homogene Transformationsmatrix verwendet wird, die die 3D-Lage von *cs*1 relativ zu *cs*2 beschreibt:

$$\begin{pmatrix} \mathbf{p}^{cs2} \\ 1 \end{pmatrix} = {}^{cs2}\mathbf{H}_{cs1} \cdot \begin{pmatrix} \mathbf{p}^{cs1} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R}(\texttt{RotX}, \texttt{RotY}, \texttt{RotZ}) \cdot \mathbf{p}^{cs1} + \begin{pmatrix} \texttt{TransX} \\ \texttt{TransY} \\ \texttt{TransZ} \end{pmatrix} \\ 1 \end{pmatrix}$$

Die Gleichung entspricht der folgenden Sequenz von Operatoraufrufen:

```
pose_to_hom_mat3d(PoseOf1In2, HomMat3DFrom1In2)
affine_trans_point_3d(HomMat3DFrom1In2, P1X, P1Y, P1Z, P2X, P2Y, P2Z)
```

Nicht dem Standard entsprechende Darstellungen von 3D-Lagen

Bisher wurde die Standard-Definition von 3D-Lagen beschrieben. Um diese zu erzeugen, übergibt man in den Parametern `OrderOfTransform` und `ViewOfTransform` die (Standard-)Werte *'Rp+T'* bzw. *'point'*. Für andere Werte werden nicht dem Standard entsprechende Typen von 3D-Lagen erzeugt, die kurz im Folgenden beschrieben werden. Bitte beachten Sie, dass diese Darstellungstypen nur zur Abwärtskompatibilität unterstützt werden; wir empfehlen nachdrücklich, die Standard-Typen zu verwenden.

Wenn *'R(p-T)'* in `OrderOfTransform` übergeben wird, entspricht die erzeugte 3D-Lage der folgenden Transformationskette, d.h. die Reihenfolge von Translation und Rotation ist vertauscht und die Translation ist negiert:

$$\mathbf{H}_{R(p-T)} = \begin{bmatrix} & \mathbf{R}(\texttt{RotX},\texttt{RotY},\texttt{RotZ}) & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -\texttt{TransX} \\ 0 & 1 & 0 & -\texttt{TransY} \\ 0 & 0 & 1 & -\texttt{TransZ} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{R}) \cdot \mathbf{H}(-\mathbf{T})$$

Wenn in *'coordinate_system'* in `ViewOfTransform` übergeben wird, bleibt die Reihenfolge der Transformationen gleich, aber die Rotationswinkel werden negiert. Bitte beachten Sie, dass dies nicht der Transformation eines Koordinatensystems entspricht, auch wenn der Parameterwert dies suggeriert!

$$\mathbf{H}_{coordinate_system} = \begin{bmatrix} 1 & 0 & 0 & \texttt{TransX} \\ 0 & 1 & 0 & \texttt{TransY} \\ 0 & 0 & 1 & \texttt{TransZ} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} & \mathbf{R}(-\texttt{RotX},-\texttt{RotY},-\texttt{RotZ}) & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Zurückgelieferte Datenstruktur

Die erzeugte 3D-Lage wird in `Pose` zurückgeliefert, in Form eines Tupels mit sieben Elementen. Die ersten drei Elemente enthalten die Translationsparameter `TransX`, `TransY` und `TransZ`, gefolgt von den Rotationsparametern `RotX`, `RotY` und `RotZ`. Das letzte Element kodiert den Darstellungstyp der 3D-Lage, der mit den Parametern `OrderOfTransform`, `OrderOfRotation` und `ViewOfTransform` gewählt wurde. Die folgende Tabelle listet die möglichen Kombinationen. Wie schon erwähnt empfehlen wir, nur die Typen mit `OrderOfTransform` = *'Rp+T'* und `ViewOfTransform` = *'point'* zu verwenden (Codes 0, 2, and 4).

OrderOfTransform	OrderOfRotation	ViewOfTransform	Code
'Rp+T'	*'gba'*	*'point'*	0
'Rp+T'	*'abg'*	*'point'*	2
'Rp+T'	*'rodriguez'*	*'point'*	4
'Rp+T'	*'gba'*	*'coordinate_system'*	1
'Rp+T'	*'abg'*	*'coordinate_system'*	3
'Rp+T'	*'rodriguez'*	*'coordinate_system'*	5
'R(p-T)'	*'gba'*	*'point'*	8
'R(p-T)'	*'abg'*	*'point'*	10
'R(p-T)'	*'rodriguez'*	*'point'*	12
'R(p-T)'	*'gba'*	*'coordinate_system'*	9
'R(p-T)'	*'abg'*	*'coordinate_system'*	11
'R(p-T)'	*'rodriguez'*	*'coordinate_system'*	13

3D-Lagen können mit `convert_pose_type` in andere Darstellungstypen konvertiert werden; mit `get_pose_type` kann der Darstellungstyp einer 3D-Lage abgefragt werden.

Parameter

▷ **TransX** (input_control) . . . . . real ⇝ *real*
Translation in Richtung der x-Achse (in [m]).
Defaultwert: 0.1
Wertevorschläge: TransX ∈ {-1.0, -0.75, -0.5, -0.25, -0.2, -0.1, -0.5, -0.25, -0.125, -0.01, 0.0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0}

▷ **TransY** (input_control) . . . . . real ⇝ *real*
Translation in Richtung der y-Achse (in [m]).
Defaultwert: 0.1
Wertevorschläge: TransY ∈ {-1.0, -0.75, -0.5, -0.25, -0.2, -0.1, -0.5, -0.25, -0.125, -0.01, 0.0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0}

▷ **TransZ** (input_control) . . . . . real ⇝ *real*
Translation in Richtung der z-Achse (in [m]).
Defaultwert: 0.1
Wertevorschläge: TransZ ∈ {-1.0, -0.75, -0.5, -0.25, -0.2, -0.1, -0.5, -0.25, -0.125, -0.01, 0.0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0}

▷ **RotX** (input_control) . . . . . real ⇝ *real*
Rotation um x-Achse bzw. x-Komponente des Rodriguez-Vektors (in [°] bzw. ohne Einheit).
Defaultwert: 90.0
Wertevorschläge: RotX ∈ {0.0, 90.0, 180.0, 270.0}
Typischer Wertebereich: $0 \leq$ RotX ≤ 360

▷ **RotY** (input_control) . . . . . real ⇝ *real*
Rotation um y-Achse bzw. y-Komponente des Rodriguez-Vektors (in [°] bzw. ohne Einheit).
Defaultwert: 90.0
Wertevorschläge: RotY ∈ {0.0, 90.0, 180.0, 270.0}
Typischer Wertebereich: $0 \leq$ RotY ≤ 360

▷ **RotZ** (input_control) . . . . . real ⇝ *real*
Rotation um z-Achse bzw. z-Komponente des Rodriguez-Vektors (in [°] bzw. ohne Einheit).
Defaultwert: 90.0
Wertevorschläge: RotZ ∈ {0.0, 90.0, 180.0, 270.0}
Typischer Wertebereich: $0 \leq$ RotZ ≤ 360

▷ **OrderOfTransform** (input_control) . . . . . string ⇝ *string*
Reihenfolge von Rotation und Translation.
Defaultwert: 'Rp+T'
Wertevorschläge: OrderOfTransform ∈ {'Rp+T', 'R(p-T)'}

▷ **OrderOfRotation** (input_control) . . . . . string ⇝ *string*
Bedeutung der Rotationswerte.
Defaultwert: 'gba'
Wertevorschläge: OrderOfRotation ∈ {'gba', 'abg', 'rodriguez'}

▷ **ViewOfTransform** (input_control) . . . . . string ⇝ *string*
Sichtweise der Transformation.
Defaultwert: 'point'
Wertevorschläge: ViewOfTransform ∈ {'point', 'coordinate_system'}

▷ **Pose** (output_control) . . . . . pose ⇝ *real* / integer
3D-Lage (Pose).
Parameteranzahl: 7

Beispiel

```
* Create a pose.
create_pose (0.1, 0.2, 0.3, 40, 50, 60, 'Rp+T', 'gba', 'point', Pose)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_pose den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

pose_to_hom_mat3d, write_pose, camera_calibration, hand_eye_calibration

Alternativen

read_pose, hom_mat3d_to_pose

Siehe auch

hom_mat3d_rotate, hom_mat3d_translate, convert_pose_type, get_pose_type, hom_mat3d_to_pose, pose_to_hom_mat3d, write_pose, read_pose

Modul

Foundation

deserialize_pose (: : SerializedItemHandle : Pose)

Deserialisieren einer serialisierten 3D-Lage (Pose).

deserialize_pose deserialisiert eine 3D-Lage (Pose), welche mit serialize_pose serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte Pose wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in einer dafür automatisch erzeugten Pose mit dem Handle Pose gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **Pose** (output_control) ..... pose ⇝ *real* / integer
3D-Lage (Pose).
Parameteranzahl: 7

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_pose

Modul

Foundation

dual_quat_to_pose (: : DualQuaternion : Pose)

Umwandeln einer dualen Quaternion in eine entsprechende 3D-Lage.

Der Operator dual_quat_to_pose wandelt die duale Quaternion DualQuaternion in ihre entsprechende 3D-Lage Pose um.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Werden in DualQuaternion mehrere duale Quaternionen übergeben, so wird jede duale Quaternion eine Pose zurückgegeben.

Achtung

dual_quat_to_pose nimmt an, dass es sich bei DualQuaternion um eine duale Einheitsquaternion handelt und diese daher eine starre 3D-Abbildung darstellt. Andernfalls ist eine Umwandlung in eine Pose nicht sinnvoll möglich.

Parameter

- ▷ **DualQuaternion** (input_control) . . . . . dual_quaternion(-array) ⇝ *real*
  Duale Einheitsquaternion.
- ▷ **Pose** (output_control) . . . . . pose(-array) ⇝ *real* / integer
  3D-Lage.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

dual_quat_compose, dual_quat_interpolate, screw_to_dual_quat, dual_quat_conjugate

Nachfolger

camera_calibration, hand_eye_calibration, write_pose, disp_caltab, sim_caltab

Alternativen

dual_quat_to_screw, dual_quat_to_hom_mat3d

Siehe auch

pose_to_dual_quat, dual_quat_normalize, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, quat_to_pose

Modul

Foundation

get_circle_pose (Contour : : CameraParam, Radius, OutputType : Pose1, Pose2)

Bestimmen der 3D-Lage (Pose) eines Kreises aus seiner perspektivischen Projektion in 2D.

Jede Ellipse im Bild kann als die perspektivische Projektion eines Kreises interpretiert werden. Tatsächlich existieren für einen vorgegebenen Radius des Kreises zwei unterschiedlich orientierte Kreise im Raum, die dieselbe Projektion zur Folge haben. get_circle_pose bestimmt die 3D-Positionen und -Orientierungen dieser beiden Kreise. Zuerst wird jede Contour durch eine Ellipse approximiert. Dann werden, basierend auf den internen Kameraparametern (CameraParam) und dem Radius des Kreises in 3D (Radius), die 3D-Positionen und -Orientierungen (Pose1,Pose2) in Kamerakoordinaten bestimmt.

In Abhängigkeit vom Wert des Parameters OutputType werden Position und Orientierung als 3D-Lage (Pose) (OutputType = *'pose'*) oder in Form des Mittelpunktes des Kreises und des Normalenvektors der Ebene, in der der Kreis liegt (OutputType = *'center_normal'*), zurückgegeben. Im ersteren Fall wird der Winkel für die Rotation um die z-Achse auf Null gesetzt, da er nicht bestimmt werden kann. Im letzteren Fall enthalten die ersten drei Elemente der Ausgabeparameter Pose1 und Pose2 jeweils die Kamerakoordinaten des Mittelpunktes. Die folgenden drei Elemente enthalten den Normalenvektor. Der Normalenvektor ist normalisiert und zeigt auf die Seite der Ebene, die das Projektionszentrum, also den Ursprung des Kamerakoordinatensystems, nicht enthält. Ist der Parameter OutputType auf den Wert *'center_normal'* gesetzt, enthalten die Ausgabeparameter Pose1 und Pose2 nur jeweils sechs Elemente, die die Position und die Orientierung der Kreise beschreiben. Ist hingegen OutputType = *'pose'*, enthalten die Ausgabeparameter Pose1 und Pose2 jeweils die sieben Elemente der Pose.

Wird mehr als eine Kontur in Contour übergeben, muss in Radius entweder ein Tupel übergeben werden, das für jede Kontur einen Wert enthält, oder Radius enthält nur einen Wert, der dann für alle Konturen verwendet wird. Die Ausgabeparameter Pose1 und Pose2 enthalten die Posen bzw. die Positionen und Orientierungen

hintereinander, d.h. Pose1 und Pose2 enthalten zuerst die Pose bzw. die Position und den Normalenvektor für die erste Kontur, dann die Werte für die zweite Kontur, usw.

Achtung

Die Genauigkeit der Posen hängt stark von der Genauigkeit der extrahierten Konturen ab. Die Extraktion von gekrümmten Kanten mit relativ großen Filtermasken hat zur Folge, dass die Kantenposition leicht verschoben wird. Kantenextraktionsverfahren, die auf der ersten Ableitung der Bildfunktion basieren (z.B. edges_sub_pix) verschieben die Kantenposition in Richtung des Krümmungsmittelpunkts, d.h. extrahierte Ellipsen werden geringfügig kleiner. Extraktionsverfahren, die auf der zweiten Ableitung der Bildfunktion basieren (laplace_of_gauss gefolgt von zero_crossing_sub_pix), verschieben die Kantenposition vom Krümmungsmittelpunkt weg, d.h. extrahierte Ellipsen werden geringfügig größer.

Diese Effekte sind umso stärker, je stärker die Kante gekrümmt ist und je größer die verwendete Filtermaske ist. Um hohe Genauigkeiten erzielen zu können, sollten daher die Ellipsen im Bild groß sein und der Filterparameter sollte so gewählt werden, dass kleine Filtermasken verwendet werden (siehe info_edges).

Parameter

- ▷ **Contour** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen.
- ▷ **CameraParam** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **Radius** (input_control) ..... number(-array) ⇝ *real*
  Radius des Kreises im Objektraum.
  Parameteranzahl: Radius == Contour || Radius == 1
  Restriktion: Radius > 0.0
- ▷ **OutputType** (input_control) ..... string ⇝ *string*
  Typ der Rückgabewerte.
  Defaultwert: 'pose'
  Werteliste: OutputType ∈ {'pose', 'center_normal'}
- ▷ **Pose1** (output_control) ..... real-array ⇝ *real* / integer
  3D-Lage (Pose) des ersten Kreises.
  Parameteranzahl: Pose1 == 7 * Contour || Pose1 == 6 * Contour
- ▷ **Pose2** (output_control) ..... real-array ⇝ *real* / integer
  3D-Lage (Pose) des zweiten Kreises.
  Parameteranzahl: Pose2 == 7 * Contour || Pose2 == 6 * Contour

Ergebnis

Sind die Parameterwerte korrekt und konnte die Position des Kreises bestimmt werden, dann liefert get_circle_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

edges_sub_pix

Alternativen

find_marks_and_pose, camera_calibration

Siehe auch

get_rectangle_pose, fit_ellipse_contour_xld

Modul

3D Metrology

get_pose_type (: : Pose : OrderOfTransform, OrderOfRotation, ViewOfTransform)

Abfragen des Darstellungstyps einer 3D-Lage (Pose).

Mit get_pose_type wird der Darstellungstyp der 3D-Lage Pose abgefragt. Siehe create_pose für eine detaillierte Beschreibung von 3D-Lagen, ihren Darstellungstypen und die Bedeutung der Parameter OrderOfTransform, OrderOfRotation und ViewOfTransform.

Parameter

▷ **Pose** (input_control) . . . . . pose ⇝ *real* / integer
3D-Lage (Pose).
Parameteranzahl: 7

▷ **OrderOfTransform** (output_control) . . . . . string ⇝ *string*
Reihenfolge von Rotation und Translation.

▷ **OrderOfRotation** (output_control) . . . . . string ⇝ *string*
Bedeutung der Rotationswerte.

▷ **ViewOfTransform** (output_control) . . . . . string ⇝ *string*
Sichtweise der Transformation.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_pose_type den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_pose, hom_mat3d_to_pose, camera_calibration, hand_eye_calibration

Nachfolger

convert_pose_type

Siehe auch

create_pose, convert_pose_type, write_pose, read_pose

Modul

Foundation

get_rectangle_pose (Contour : : CameraParam, Width, Height, WeightingMode, ClippingFactor : Pose, CovPose, Error)

Bestimmen der 3D-Lage (Pose) eines Rechtecks aus seiner perspektivischen Projektion in 2D.

Ein Rechteck im Raum wird als ein allgemeines Viereck in Bild projiziert. get_rectangle_pose bestimmt die 3D-Lage (Pose) des Rechtecks aus seiner Projektionskontur (Contour).

Der Algorithmus arbeitet auf folgende Weise: Erst wird die Kontur in vier Linien segmentiert und ihre Schnittpunkte werden als Eckpunkte der Kontur genommen. Die Eckpunkte, die internen Kameraparameter (CameraParam) und die Größe des Rechtecks in Meter(Width, Height) werden für die Bestimmung der Initialpose verwendet. Die Pose wird zum Schluss durch eine nichtlineare Optimierung verfeinert, wobei der Abstand von den Konturpunkten zu der Wiederprojektion des Rechtecks im Bild minimiert wird.

Der Operator unterstützt nur (projektive) Lochkamera mit Flächensensor. Falls CameraParam eine Lochkamera mit Zeilensensor oder eine telezentrische Kamera darstellt (siehe Kalibrierung), wird ein Fehler geliefert.

Width und Height beschreiben die Größe des Rechtecks in $X-$ bzw. $Y-$Richtung in seinem Koordinatensystem. Der Ursprung dieses Koordinatensystem liegt in dem Mittelpunkt des Rechtecks. Die $Z-$Achse zeigt weg von der Kamera.

Die Parameter WeightingMode und ClippingFactor können verwendet werden, um die Einwirkung von Ausreißern im Algorithmus zu vermindern. Wenn WeightingMode auf *'tukey'* oder *'huber'* gesetzt ist, werden die Konturpunkte nach den Verfahren von Tukey bzw. Huber gewichtet. In diesem Fall wird die Standardabweichung der Abstände der Konturpunkte zu dem rückprojizierten Rechteck durch robuste Fehlerstatistik bestimmt. Der Parameter ClippingFactor (Skalierungsfaktor für die Standardabweichung) kontrolliert das Dämpfen von Ausreißern: Je kleiner der Wert von ClippingFactor ist, desto mehr Ausreißer werden erkannt. Siehe

fit_line_contour_xld für eine Diskussion über die Eigenschaften der verschiedenen Verfahren. Bitte beachten Sie, dass hier, zum Unterschied von fit_line_contour_xld, das Verfahren von Huber empfohlen wird.

Ausgabe

Die gelieferte Pose hat Code-0 (siehe create_pose) und stellt die Position des Mittelpunkts des Rechtecks dar. Die Positionen der Eckpunkte des Rechtecks kann man wie folgt berechnen:

```
set_origin_pose (Pose, Width/2, -Height/2, 0, PoseCorner1)
set_origin_pose (Pose, Width/2, Height/2, 0, PoseCorner2)
set_origin_pose (Pose, -Width/2, Height/2, 0, PoseCorner3)
set_origin_pose (Pose, -Width/2, -Height/2, 0, PoseCorner4)
```

Ein Rechteck ist symmetrisch um seine X-, Y- und Z-Achse und deswegen kann eine viereckige Kontur ein Rechteck in 4 verschiedenen Posen darstellen. Die Winkel in Pose sind normalisiert, so dass sie in dem Intervall $[-90; 90]$ Grad liegen. Die 3 verbliebenen möglichen Posen können durch eine Kombination von Umdrehungen, die von den folgenden Formel dargestellt sind, berechnet werden:

```
* Achtung: Die folgende Code gilt nur für Pose von Code-0,
*          wie von get_rectangle_pose zurückgeliefert
*
* flip around z-axis
PoseFlippedZ := Pose
PoseFlippedZ[5] := PoseFlippedZ[5]+180
* flip around y-axis
PoseFlippedY := Pose
PoseFlippedY[4] := PoseFlippedY[4]+180
PoseFlippedY[5] := -PoseFlippedY[5]
* flip around x-axis
PoseFlippedX := Pose
PoseFlippedX[3] := PoseFlippedX[3]+180
PoseFlippedX[4] := -PoseFlippedX[4]
PoseFlippedX[5] := -PoseFlippedX[5]
```

Wenn das Rechteck ein Quadrat ist (Width == Height), dann ist die Anzahl der möglichen Posen 8.

Wenn mehr als eine Kontur in Contour übergeben wird, dann muss ein entsprechend großes Tupel von Werten auch in Width und bzw. Height übergeben werden. Wird nur ein Wert für jeweils beide Parameter übergeben, wird dieser Wert für alle verarbeiteten Konturen verwendet. Für jede Kontur wird eine Pose bestimmt und alle in Pose hintereinander gehängt zurückgeliefert (siehe das Beispiel).

Genauigkeit der Pose

Die Genauigkeit der berechneten Pose hängt von den folgenden Faktoren ab:

- Seitenverhältnis Width/Height
- Länge der projizierten Kontur
- Grad der perspektivische Verzerrung

Um eine genaue Pose berechnen zu können, gibt es drei entsprechende Kriterien:

Erstens sollte das **Seitenverhältnis** Width/Height die folgenden Beschränkungen erfüllen:

$$\frac{1}{3} < \texttt{Width}/\texttt{Height} < 3$$

Hat das Rechteck ein von dieser Regel abweichendes Seitenverhältnis, dominieren seine längere Seite die Posenbestimmung. Dabei entsteht eine Instabilität bei der Bestimmung des Winkels um die längere Achse des Rechtecks. Im Extremfall, in dem eine der Dimensionen des Rechtecks gleich 0 ist, ist das Rechteck ein Liniensegment und seine Lage kann nicht bestimmt werden.

Zweitens muss jede **Konturseite mindestens 20 Pixel lang** sein. Falls eine Seite weniger als 5 Pixel lang ist, wird ein Fehler geliefert.

Drittens, je mehr **perspektivische Verzerrung** bei der Projektion der Kontur auftritt, desto stabiler arbeitet der Algorithmus. Deswegen kann die Pose eines gekippten Rechtecks besser bestimmt werden als die Pose eines parallel

zur Bildebene liegenden Rechtecks. Das wird weiter im nächsten Absatz diskutiert. Außerdem gilt die Faustregel, dass das Rechteck so im Raum positioniert sein soll, dass seine Größe in $X-$ und $Y-$Kamerakoordinate nicht kleiner als 1/10tel seines Abstands von der Kamera in $Z-$Richtung sein darf.

`get_rectangle_pose` liefert zwei Maße für die Genauigkeit der bestimmten `Pose`. `Error` ist der durchschnittliche Pixelabstand zwischen den Konturpunkten und dem rückprojizierten Rechteck. Falls `Error` den Wert von 0.5 überschreitet, hat der Algorithmus nicht richtig konvergiert. In diesem Fall sollte die gelieferte `Pose` nicht verwendet werden. `CovPose` enthält 36 Werte, die die 6×6 Kovarianzmatrix der 6 ersten Posenwerte darstellen. Die oben genannte Instabilität des Winkels um die längere Rechteckachse kann daran erkannt werden, dass die Absolutwerte der (Ko-)Varianzen der Rotationen um die X- bzw. Y-Achse größer als 0.05 sind (`CovPose`[21], `CovPose`[28], `CovPose`[22] und `CovPose`[28]). Erhöhte Werte von allen anderen Kovarianzen und insbesondere von den Varianzen (die 6 Werte auf der Diagonale von `CovPose` mit Index 0, 7, 14, 21, 28 und 35) deuten ebenfalls auf eine schlechte Qualität der bestimmten `Pose`.

Parameter

- ▷ **`Contour`** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen.
- ▷ **`CameraParam`** (input_control) ..... campar ⇝ *real* / integer / string
  Interne Kameraparameter.
- ▷ **`Width`** (input_control) ..... number(-array) ⇝ *real*
  Breite des Rechtecks in Meter.
  Restriktion: Width > 0
- ▷ **`Height`** (input_control) ..... number(-array) ⇝ *real*
  Höhe des Rechtecks im Meter.
  Restriktion: Height > 0
- ▷ **`WeightingMode`** (input_control) ..... string ⇝ *string*
  Gewichtungsart für die Optimierungsphase.
  Defaultwert: 'nonweighted'
  Werteliste: `WeightingMode` ∈ {'nonweighted', 'huber', 'tukey'}
- ▷ **`ClippingFactor`** (input_control) ..... number ⇝ *real*
  Abschnittsfaktor für das Dämpfen von Ausreißern (normalerweise: *1.0* für *'huber'* und *3.0* für *'tukey'*).
  Defaultwert: 2.0
  Wertevorschläge: `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
  Restriktion: ClippingFactor > 0
- ▷ **`Pose`** (output_control) ..... pose(-array) ⇝ *real* / integer
  3D-Lage (Pose) des Rechtecks
  Parameteranzahl: Pose == 7 * Contour
- ▷ **`CovPose`** (output_control) ..... number-array ⇝ *real*
  Kovarianzen der 6 Posenwerte.
  Parameteranzahl: CovPose == 36 * Contour
- ▷ **`Error`** (output_control) ..... number-array ⇝ *real*
  Durchschnittsabstand zwischen dem rückprojizierten Rechteck und der Kontur.
  Parameteranzahl: Error == Contour

Beispiel

```
* Process an image with several rectangles of the same size appearing
* as light objects.
RectWidth := 0.04
RectHeight := 0.025
gen_cam_par_area_scan_division (0.014, 830, 4.58e-006, 4.65e-006, \
                                509, 488, 800, 800, CameraParam)
read_image (Image, 'barcode/ean13/tea_box_01')
* Find light objects in the image.
mean_image (Image, ImageMean, 201, 201)
dyn_threshold (Image, ImageMean, Region, 5, 'light')
* Fill gaps in the objects.
fill_up (Region, RegionFillUp)
* Extract rectangular contours.
```

```
connection (RegionFillUp, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, ['area','rectangularity'], \
              'and', [50000,0.9], [640000,1])
boundary (SelectedRegions, RegionBorder, 'inner')
dilation_circle (RegionBorder, RegionDilation, 3.5)
* NOTE: for a real application, this step might require some additional
*       pre- or postprocessing.
reduce_domain (Image, RegionDilation, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'canny', 1, 20, 40)
* Get the pose of all contours found.
get_rectangle_pose (Edges, CameraParam, RectWidth, RectHeight, 'huber', 2, \
                    Poses, CovPose, Error)
NumPoses := |Poses|/7
for I := 0 to NumPoses-1 by 1
  Pose := Poses[I*7:I*7+6]
  * Use the Pose here.
  * ...
endfor
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Position des Rechtecks bestimmt werden, dann liefert `get_rectangle_pose` den Wert TRUE. Falls die übergebene Kontur nicht wie ein Viereck segmentiert werden kann, liefert `get_rectangle_pose` den Wert H_ERR_FIT_QUADRANGLE. Gegebenenfalls wird eine weiter Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`edges_sub_pix`

Siehe auch

`get_circle_pose`, `set_origin_pose`, `camera_calibration`

Literatur

G.Schweighofer and A.Pinz: „Robust Pose Estimation from a Planar Target“; Transactions on Pattern Analysis and Machine Intelligence (PAMI), 28(12):2024-2030, 2006

Modul

3D Metrology

pose_average (: : Poses, Weights, Mode, SigmaT, SigmaR : AveragePose, Quality)

Berechnet den Mittelwert einer Menge von Posen.

`pose_average` mittelt die Rotation und die Translation der in `Poses` übergebenen Posen und liefert die gemittelte Pose in `AveragePose` zurück. Die Posen können über den Parameter `Weights` gewichtet werden. Wird ein leeres Tupel übergeben, so haben alle Posen den gleichen Einfluss und werden gleich behandelt. Andernfalls muss ein Tupel übergeben werden dessen Länge der Anzahl der Posen entspricht und welches für jede Pose ein positives Gewicht enthält. Ein höheres Gewicht entspricht einem stärkeren Einfluss der Pose auf den Mittelwert.

`pose_average` unterstützt zwei Verfahren zur Mittelung der Posen, welche über den Parameter `Mode` ausgewählt werden können. *'direct'* wählt ein direktes Lösungsverfahren, bei dem Translation und Rotation unabhängig gemittelt werden und das Ergebnis unmittelbar zurückgegeben wird. *'iterative'* verwendet die mit *'direct'* ermittelte Pose als Startwert für ein iteratives Verfahren, welches den lokalen Modus der Posen bestimmt. Das iterative Verfahren ist langsamer als das direkte Verfahren, liefert aber vor allem beim Vorhandensein von Ausreißern genauere Ergebnisse zurück. Dabei werden Posen, die von der mittleren Pose stark abweichen, geringer gewichtet.

Für das iterative Verfahren kann die relative Gewichtung der Rotation und der Translation über die Parameter SigmaT und SigmaR gesetzt werden. Jeder der beiden Werte kann auf *'auto'* gesetzt werden, wodurch er automatisch geschätzt wird. Für SigmaT wird dabei die Streuung der Translationen der übergebenen Posen verwendet, für SigmaR wird eine fester Wert verwendet. Die Werte beschreiben die erwartete Streuung der Translation bzw. Rotation. Für das direkte Verfahren werden die beiden Parameter ignoriert.

In Quality wird ein Maß für die Qualität der gefundenen Pose zurückgegeben. Quality ist stets ein vierelementiges Tupel, welches die mittlere und maximale Abweichung der Translation und Rotation der gemittelten Pose von den übergebenen Posen zurückgibt. Die Rotation wird dabei in Grad angegeben. Die Reihenfolge der Werte ist: Mittlerer quadratischer Translationsfehler, mittlerer quadratischer Rotationsfehler, maximale Translations- und maximale Rotationsabweichung. Evtl. in Weights übergebene Gewichte der Posen werden für die mittlere quadratische Translations- und Rotationsfehler berücksichtigt, aber nicht für die andere zwei Werte des Qualitätsschätzers.

Parameter

- ▷ **Poses** (input_control) . . . . . pose-array ⇝ *integer*
  Liste von Posen deren Mittelwert berechnet wird.
- ▷ **Weights** (input_control) . . . . . number-array ⇝ *real* / integer
  Gewichte der Posen, oder ein leeres Tupel.
  Defaultwert: []
  Restriktion: Weights > 0 && length(Weights) == 0 || length(Weights) == length(Poses) / 7
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Art der Mittelwertbildung.
  Defaultwert: 'iterative'
  Werteliste: Mode ∈ {'direct', 'iterative'}
- ▷ **SigmaT** (input_control) . . . . . number ⇝ *real* / integer / string
  Gewicht der Translation.
  Defaultwert: 'auto'
  Wertevorschläge: SigmaT ∈ {'auto', 0.1, 1, 100}
- ▷ **SigmaR** (input_control) . . . . . number ⇝ *real* / integer / string
  Gewicht der Rotation.
  Defaultwert: 'auto'
  Wertevorschläge: SigmaR ∈ {'auto', 0.1, 1, 10}
- ▷ **AveragePose** (output_control) . . . . . pose ⇝ *real* / integer
  Gewichteter Mittelwert der Posen.
- ▷ **Quality** (output_control) . . . . . number-array ⇝ *real*
  Abweichung des Mittelwerts von den Eingabeposen.
  Zusicherung: length(Quality) == 4

Ergebnis

Sind alle Parameter gültig, dann liefert pose_average den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_pose, hom_mat3d_to_pose

Siehe auch

read_pose, hom_mat3d_to_pose, pose_to_hom_mat3d

Modul

Foundation

pose_compose (: : PoseLeft, PoseRight : PoseCompose)

Komposition von 3D-Lagen aus zwei Tupeln.

pose_compose verbindet jeweils zwei Posen aus den Eingabetupeln PoseLeft und PoseRight. Falls beide Tupel dieselbe Anzahl Posen enthalten, werden die entsprechenden Posen der beiden Tupel multipliziert. Ansonsten darf entweder PoseLeft oder PoseRight nur eine Pose enthalten. In diesem Fall wird die Multiplikation für jede Pose des längeren Tupels mit der einzigen Pose des anderen Tupels ausgeführt. Für jede Komposition sind die Posen als Transformationen von Koordinatensystemen zu verstehen. Jede Pose in Ausgabetupel PoseCompose ist folglich das Ergebnis der sequentiellen Anwendung der entsprechenden Posen der Eingabetupel PoseLeft und PoseRight.

Dazu werden zunächst die zu verbindenden Posen aus PoseLeft und PoseRight in die entsprechenden homogenen Transformationsmatrizen $\mathbf{H}_1$ und $\mathbf{H}_2$ umgewandelt. Dann wird $\mathbf{H}_1$ mit $\mathbf{H}_2$ multipliziert. Die resultierende Matrix $\mathbf{H}_1 \cdot \mathbf{H}_2$ wird dann in eine Pose umgerechnet und an entsprechender Stelle im Ausgabetupel PoseCompose zurückgegeben.

Falls die zu verbindenden Posen aus PoseLeft und PoseRight verschiedene Posentypen haben, hat die Ergebnispose den Type 0 ('Rp+T', 'gba', 'point'). Andernfalls hat die Ergebnispose den gleichen Typ wie die Eingangsposen.

Parameter

- ▷ **PoseLeft** (input_control) ..... pose(-array) ⇝ *real* / integer
  Tupel der linken Posen.
- ▷ **PoseRight** (input_control) ..... pose(-array) ⇝ *real* / integer
  Tupel der rechten Posen.
- ▷ **PoseCompose** (output_control) ..... pose(-array) ⇝ *real* / integer
  Tupel der Ausgabeposen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert pose_compose den Wert TRUE. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_pose, hom_mat3d_to_pose, create_pose, convert_pose_type, pose_invert

Nachfolger

convert_pose_type

Alternativen

hom_mat3d_compose, dual_quat_compose

Siehe auch

pose_to_hom_mat3d, hom_mat3d_to_pose, hom_mat3d_compose

Modul

Foundation

pose_invert (: : Pose : PoseInvert)

Invertierung von als Tupel gegebenen 3D-Lagen.

pose_invert invertiert die Posen des Tupels Pose. Hierzu wird jede Pose in die entsprechende homogene Transformationsmatrix $\mathbf{H}$ umgewandelt und diese wird invertiert. Die resultierende Matrix wird dann in eine Pose umgerechnet. Diese wird an der entsprechenden Stelle im Tupel PoseInvert zurückgegeben. Die zurückgegebenen Poses haben die gleichen Typen wie die Originalposen.

Parameter

▷ **Pose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
Tupel von 3D-Lagen.

▷ **PoseInvert** (output_control) . . . . . pose(-array) ⇝ *real* / integer
Tupel von invertierten 3D-Lage.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `pose_invert` den Wert TRUE. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`read_pose`, `hom_mat3d_to_pose`, `create_pose`, `convert_pose_type`, `pose_compose`

Nachfolger

`convert_pose_type`

Alternativen

`hom_mat3d_invert`, `dual_quat_conjugate`

Siehe auch

`pose_to_hom_mat3d`, `hom_mat3d_to_pose`

Modul

Foundation

pose_to_dual_quat (: : Pose : DualQuaternion)

Umwandeln einer 3D-Lage in eine entsprechende duale Einheitsquaternion.

Der Operator `pose_to_dual_quat` wandelt die `Pose` in ihre entsprechende duale Einheitsquaternion `DualQuaternion` um.

Eine kurze Einführung in duale Quaternionen, die verwendete Notation und die Beziehung zwischen dualen Quaternionen und Schraubungen findet sich in der Dokumentation zu diesem Kapitel (Transformationen / Duale Quaternionen).

Da $\hat{q}$ und $-\hat{q}$ die selbe starre 3D-Abbildung repräsentieren, wird das Vorzeichen der resultierenden dualen Quaternion `DualQuaternion` so gewählt, dass der Skalarteil ihres Realteils ≥ 0 ist.

Werden in `Pose` mehrere Posen übergeben, so wird jede Pose eine duale Quaternion zurückgegeben.

Parameter

▷ **Pose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
3D-Lage.

▷ **DualQuaternion** (output_control) . . . . . dual_quaternion(-array) ⇝ *real*
Duale Einheitsquaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_pose`, `read_pose`

Nachfolger

dual_quat_compose, dual_quat_interpolate, dual_quat_to_screw, dual_quat_conjugate

Alternativen

pose_to_hom_mat3d, screw_to_dual_quat

Siehe auch

dual_quat_to_hom_mat3d, dual_quat_to_pose, dual_quat_normalize, serialize_dual_quat, deserialize_dual_quat, dual_quat_trans_line_3d, pose_to_quat

Modul

Foundation

pose_to_quat (: : Pose : Quaternion)

Wandelt die Rotation einer 3D-Lage in eine entsprechende Quaternion um.

Der Operator quat_to_pose wandelt die Rotation der 3D-Lage Pose in die entsprechende Quaternion Quaternion um. Die Translation von Pose wird ignoriert. Werden in Pose mehrere Posen übergeben, so wird für jede davon eine Quaternion erzeugt.

Parameter

- ▷ **Pose** (input_control) . . . . . pose(-array) ⇝ *real* / integer
  3D-Lage.
- ▷ **Quaternion** (output_control) . . . . . quaternion(-array) ⇝ *real*
  Rotationsquaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

create_pose, read_pose

Nachfolger

quat_compose, quat_rotate_point_3d, quat_interpolate

Alternativen

axis_angle_to_quat

Siehe auch

quat_to_pose, quat_to_hom_mat3d, quat_compose, quat_rotate_point_3d, pose_to_dual_quat

Modul

Foundation

proj_hom_mat2d_to_pose (: : Homography, CameraMatrix, Method : Pose)

Berechnen einer Pose anhand der Homographie von Weltkoordinaten zu Bildkoordinaten.

proj_hom_mat2d_to_pose berechnet eine Pose anhand der Homography von 2D-Weltkoordinaten (Einheit Meter) zu 2D Bildkoordinaten. Die Homography kann z.B. durch Verwendung von vector_to_proj_hom_mat2d erstellt werden und darf keinesfalls mit Homographien aus 2D Matching Operatoren (z.B. find_uncalib_descriptor_model) verwechselt werden. Zur Berechnung der Pose ist eine Kameramatrix (z.B. aus cam_par_to_cam_mat) nötig, die in CameraMatrix übergeben wird.

Durch den Parameter Method kann gewählt werden, welches Vorgehen zur Berechnung der Pose verwendet wird. Im Augenblick werden eine direkte Zerlegung (Method = 'decomposition') und eine auf Singulärwertzerlegung basierende (Method = 'decomposition_svd') unterstützt. Die direkte Zerlegung ('decomposition') liefert aufgrund numerischer Gründe im Allgemeinen robustere Ergebnisse.

Die Abfolge von Aufrufen der Operatoren vector_to_proj_hom_mat2d und proj_hom_mat2d_to_pose ist vector_to_pose für 2D Objekte ähnlich. Jedoch ist hier keine Verfeinerung aufgrund von Redundanz möglich.

Parameter

- ▷ **Homography** (input_control) . . . . . hom_mat2d ⇝ *real*
  Die Homographie zwischen Welt- und Bildkoordinaten.
- ▷ **CameraMatrix** (input_control) . . . . . hom_mat2d ⇝ *real*
  Die Kamera Kalibriermatrix K.
- ▷ **Method** (input_control) . . . . . string ⇝ *string*
  Art der Posenberechnung.
  Defaultwert: 'decomposition'
  Werteliste: Method ∈ {'decomposition', 'decomposition_svd'}
- ▷ **Pose** (output_control) . . . . . pose ⇝ *real* / integer
  Pose des 2D-Objekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert proj_hom_mat2d_to_pose den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

image_points_to_world_plane, vector_to_proj_hom_mat2d

Siehe auch

vector_to_proj_hom_mat2d, image_points_to_world_plane, vector_to_pose, camera_calibration

Modul

Calibration

quat_to_pose (: : Quaternion : Pose)

Wandelt eine Quaternion in die entsprechende 3D-Lage um.

Der Operator quat_to_pose wandelt die Eingabequaternion Quaternion in deren entsprechende 3D-Lage Pose um. Die Translation von Pose ist in jedem Fall Null.

Parameter

- ▷ **Quaternion** (input_control) . . . . . quaternion ⇝ *real*
  Rotationsquaternion.
- ▷ **Pose** (output_control) . . . . . pose ⇝ *real* / integer
  3D-Lage.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

axis_angle_to_quat, quat_compose, quat_conjugate

Nachfolger

camera_calibration, write_pose, disp_caltab, sim_caltab

Siehe auch

pose_to_quat, axis_angle_to_quat, quat_compose, quat_rotate_point_3d, quat_to_hom_mat3d, quat_normalize, dual_quat_to_pose

Modul

Foundation

read_pose (: : PoseFile : Pose)

Lesen einer 3D-Lage (Pose) aus einer Textdatei.

read_pose dient zum Einlesen einer 3D-Lage Pose aus einer Textdatei mit dem Namen PoseFile. Die Dateiendung für die 3D-Lage ist in HALCON 'dat'.

Eine 3D-Lage (Pose) beschreibt eine starre 3D-Transformation, d.h. eine Transformation bestehend aus einer beliebigen Translation und Rotation, mit 6 Parametern, drei für die Translation, drei für die Rotation. Mit Hilfe eines siebten Parameters kann zwischen verschiedenen Pose-Typen unterschieden werden (siehe create_pose).

Eine solche Datei kann durch den Operator write_pose erzeugt werden und hat z.B. folgendes Aussehen:

```
# 3D POSE PARAMETERS: rotation and translation

# Used representation type:
f 0

# Rotation angles [deg] or Rodriguez vector:
r -17.8134 1.83816 0.288092

# Translation vector (x y z [m]):
t 0.280164 0.150644 1.7554
```

Parameter

▷ **PoseFile** (input_control) ..... filename.read ⇝ *string*
Dateiname der Kameraparameterdatei.
Defaultwert: 'campose.dat'
Wertevorschläge: PoseFile ∈ {'campose.dat', 'campose_initial.dat', 'campose_final.dat'}
Dateiendung: .dat

▷ **Pose** (output_control) ..... pose ⇝ *real* / integer
3D-Lage (Pose).
Parameteranzahl: 7

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich gelesen werden, dann liefert read_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_cam_par

Nachfolger

pose_to_hom_mat3d, camera_calibration, disp_caltab, sim_caltab

Siehe auch

create_pose, find_marks_and_pose, camera_calibration, disp_caltab, sim_caltab, write_pose, pose_to_hom_mat3d, hom_mat3d_to_pose

Modul

Foundation

serialize_pose (: : Pose : SerializedItemHandle)

Serialisieren einer 3D-Lage (Pose).

serialize_pose serialisiert die Daten einer 3D-Lage (Pose) (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Dabei werden dieselben Daten in ein serialisiertes Element überführt, die von write_pose in eine Datei geschrieben werden. Die Pose wird in dem Handle Pose übergeben. Die serialisierte Pose wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_pose wieder deserialisiert werden.

Parameter

▷ **Pose** (input_control) . . . . . pose ⇝ *real / integer*
3D-Lage (Pose).
Parameteranzahl: 7

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert serialize_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

fwrite_serialized_item, send_serialized_item, deserialize_pose

Modul

Foundation

set_origin_pose (: : PoseIn, DX, DY, DZ : PoseNewOrigin)

Verschieben des Ursprungs einer 3D-Lage (Pose).

set_origin_pose verschiebt den Ursprung der 3D-Lage PoseIn um den Vektor, der durch die Parameter DX, DY und DZ gegeben ist, und liefert das Ergebnis in PoseNewOrigin zurück. Beachten Sie, dass diese Translation relativ zum lokalen Koordinatensystem der 3D-Lage selbst durchgeführt wird. Wenn z.B. PoseIn die Lage eines Objekts im Kamerakoordinatensystem beschreibt, erhält man PoseNewOrigin durch Verschiebung des Objektkoordinatensystems um DX entlang seiner eigenen x-Achse (und so weiter für die anderen Achsen), und nicht entlang der x-Achse des Kamerakoordinatensystems. Dies entspricht der folgenden Transformationskette:

$$\mathtt{PoseNewOrigin} = \mathtt{PoseIn} \cdot \begin{bmatrix} 1 & 0 & 0 & \begin{pmatrix} \mathtt{DX} \\ \mathtt{DY} \\ \mathtt{DZ} \end{pmatrix} \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

set_origin_pose ist also eine Abkürzung für die folgende Sequenz von Operatoren:

```
pose_to_hom_mat3d (PoseIn, HomMat3DIn)
hom_mat3d_translate_local (HomMat3DIn, DX, DY, DZ, HomMat3DNewOrigin)
hom_mat3d_to_pose (HomMat3DNewOrigin, PoseNewOrigin)
```

Eine typische Anwendung dieses Operators ist, wenn man ein Weltkoordinatensystem definiert, indem man die Standard-Kalibrierplatte auf die Messebene legt. In diesem Fall entsprechen die von `camera_calibration` zurückgelieferten externen Kameraparameter `PoseIn` einem Koordinatensystem, das über der Messebene liegt, da das Koordinatensystem der Kalibrierplatte auf deren Oberfläche liegt und die Platte eine gewisse Dicke hat. Zur Korrektur ruft man `set_origin_pose` mit dem Translationsvektor (0,0,D) auf, wobei D der Dicke der Kalibrierplatte entspricht.

Parameter

- ▷ **`PoseIn`** (input_control) . . . . . pose ⇝ *real* / integer
  Ursprüngliche 3D-Lage (Pose).
  Parameteranzahl: 7
- ▷ **`DX`** (input_control) . . . . . real ⇝ *real*
  Verschiebung des Ursprungs in x-Richtung.
  Defaultwert: 0
- ▷ **`DY`** (input_control) . . . . . real ⇝ *real*
  Verschiebung des Ursprungs in y-Richtung.
  Defaultwert: 0
- ▷ **`DZ`** (input_control) . . . . . real ⇝ *real*
  Verschiebung des Ursprungs in z-Richtung.
  Defaultwert: 0
- ▷ **`PoseNewOrigin`** (output_control) . . . . . pose ⇝ *real* / integer
  Neue 3D-Lage nach Verschiebung des Ursprungs.
  Parameteranzahl: 7

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_origin_pose` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_pose`, `hom_mat3d_to_pose`, `camera_calibration`, `hand_eye_calibration`

Nachfolger

`write_pose`, `pose_to_hom_mat3d`, `image_points_to_world_plane`, `contour_to_world_plane_xld`

Siehe auch

`hom_mat3d_translate_local`

Modul

Foundation

`vector_to_pose` (: : WorldX, WorldY, WorldZ, ImageRow, ImageColumn, CameraParam, Method, QualityType : Pose, Quality)

Berechnen einer absoluten Pose aus Punktkorrespondenzen von Welt- zu Bildkoordinaten.

Der Operator `vector_to_pose` berechnet eine Pose aus mindestens drei oder vier (abhängig von `Method`) Punktkorrespondenzen von 3D-Weltkoordinaten (`WorldX`, `WorldY`, `WorldZ`), gegeben in Metern, und 2D-Bildkoordinaten (`ImageRow`, `ImageColumn`), gegeben in Pixeln sowie den internen Kameraparametern (`CameraParam`). Dabei ist die Pose (die externen Kameraparameter) in der Form ${}^{ccs}\mathbf{P}_{wcs}$. Dabei steht *ccs* für das Kamerakoordinatensystem und *wcs* für das Weltkoordinatensystem, siehe auch Transformationen / Posen und „Solution Guide III-C - 3D Vision".

Parameter Method

Je nach Wahl des Parameters `Method` kann gewählt werden, welcher Algorithmus für die Berechnung der Pose verwendet werden soll.

Methoden, die für **Lochkameras** unterstützt werden:

`Method`	*Wann verwenden*	*Minimale Anzahl von Punktkorrespondenzen*
'analytic' [1]	Standardmethode für allgemeine Verwendung	4
'iterative' [2]	Falls nur drei oder vier Punktkorrespondenzen verwendet werden oder falls die Weltpunkte fast planar sind	3
'planar_analytic' [4]	Falls die Weltpunkte in einer horizontalen Ebene liegen (`WorldZ` $= 0$)	4

Die Zahlen in den eckigen Klammern in der oben stehenden Tabelle verweisen auf die Veröffentlichungen, auf denen die Implementierungen der entsprechenden Methoden basieren.

Methoden, die für **telezentrische Kameras** unterstützt werden:

`Method`	*Wann verwenden*	*Minimale Anzahl von Punktkorrespondenzen*
'telecentric' [3]	Standardmethode für allgemeine Verwendung	4
'telecentric_robust' [3]	Für sehr schlecht gestellte Punktkonfigurationen, falls `Quality` einen unerwartet hohen Wert besitzt	4
'telecentric_planar' [3]	Falls die Weltpunkte in einer horizontalen Ebene liegen (`WorldZ` $= 0$)	3
'telecentric_planar_robust' [3]	Für sehr schlecht gestellte Punktkonfigurationen, in denen die Weltpunkte in einer horizontalen Ebene liegen (`WorldZ` $= 0$) und für die `Quality` einen unerwartet hohen Wert besitzt	3

Die Zahlen in den eckigen Klammern in der oben stehenden Tabelle verweisen auf die Veröffentlichungen, auf denen die Implementierungen der entsprechenden Methoden basieren.

Parameter CameraParam und Quality

Zur Lösung des Problems benötigen alle Algorithmen die internen Kameraparameter aus der Kamerakalibrierung, die in `CameraParam` übergeben werden.

Zusätzlich kann der Benutzer durch `QualityType` eine oder mehrere Qualitätsmaß-Bewertungen der Pose bestimmen, die mitberechnet werden. Die resultierenden Qualitätsmaße werden in `Quality` aneinandergehängt zurückgeliefert. Derzeit wird nur *'error'* unterstützt. Es entspricht dem quadratischen Mittelwert der Fehler in Pixeln der projizierten 3D-Weltkoordinaten.

Allgemeine Anmerkungen

Falls eine Methode für planare Weltpunkte gewählt wird, wird angenommen, dass alle Weltpunkte in der Ebene `WorldZ` $= 0$ liegen. Daher muss für planare Weltpunkte die z-Komponente der Weltpunkte nicht gesetzt werden (`WorldZ` = *[]*), da nur 2D-Korrespondenzen verwendet werden.

Für telezentrische Kameras kann offensichtlicher Weise die Translation in z-Richtung nicht bestimmt werden. Sie wird in `Pose` auf *0* gesetzt.

Für planare Weltpunkte und telezentrische Kameras existieren immer zwei mögliche äquivalente Posen. Diese Mehrdeutigkeit kann nur durch Zusatzwissen aufgelöst werden. `vector_to_pose` gibt in diesem Fall eine beliebige der zwei möglichen Lösungen zurück. Die andere Lösung kann leicht dadurch berechnet werden, dass die Werte von α und β in `Pose` durch $360 - \alpha$ und $360 - \beta$ ersetzt werden.

Achtung

Die Methode *'analytic'* erlaubt maximal 32767 Punktkorrespondenzen als Eingabe.

Parameter

▷ **WorldX** (input_control) . . . . . number-array ⇝ *real* / integer
X-Koordinaten der Weltpunkte.
Parameteranzahl: WorldX >= 4

▷ **WorldY** (input_control) . . . . . number-array ⇝ *real* / integer
Y-Koordinaten der Weltpunkte.
Parameteranzahl: WorldY == WorldX

▷ **WorldZ** (input_control) . . . . . number-array ⇝ *real* / integer
Z-Koordinaten der Weltpunkte.
Parameteranzahl: WorldZ == WorldX || WorldZ == 0

▷ **ImageRow** (input_control) . . . . . number-array ⇝ *real* / integer
Zeilen-Koordinaten der Bildpunkte.
Parameteranzahl: ImageRow == WorldX

▷ **ImageColumn** (input_control) . . . . . number-array ⇝ *real* / integer
Spalten-Koordinaten der Bildpunkte.
Parameteranzahl: ImageColumn == WorldX

▷ **CameraParam** (input_control) . . . . . campar ⇝ *real* / integer / string
Die internen Kameraparameter aus der Kamerakalibrierung.

▷ **Method** (input_control) . . . . . string ⇝ *string*
Art des Algorithmus
Defaultwert: 'iterative'
Werteliste: `Method` ∈ {'iterative', 'analytic', 'planar_analytic', 'telecentric', 'telecentric_robust', 'telecentric_planar', 'telecentric_planar_robust'}

▷ **QualityType** (input_control) . . . . . string(-array) ⇝ *string*
Art des mit Quality zurückgegebenen Qualitätsmaßes.
Defaultwert: 'error'
Werteliste: `QualityType` ∈ {'error'}

▷ **Pose** (output_control) . . . . . pose ⇝ *real* / integer
Pose.

▷ **Quality** (output_control) . . . . . number(-array) ⇝ *real* / integer
Posenqualitätsmaß.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `vector_to_pose` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`proj_hom_mat2d_to_pose`, `vector_to_rel_pose`, `camera_calibration`

Literatur

[1] Francesc Moreno-Noguer, Vincent Lepetit, and Pascal Fua: „Accurate Non-Iterative O(n) Solution to the PnP Problem“; Eleventh IEEE International Conference on Computer Vision, 2007.

[2] Gerald Schweighofer, and Axel Pinz: „Robust Pose Estimation from a Planar Target“; Transactions on Pattern Analysis and Machine Intelligence (PAMI), 28(12):2024-2030, 2006.

[3] Carsten Steger: „Algorithms for the Orthographic-*n*-Point Problem“; Journal of Mathematical Imaging and Vision, vol. 60, no. 2, pp. 246-266, 2018.

[4] Zhengyou Zhang: „A flexible new technique for camera calibration.“; Transactions on Pattern Analysis and Machine Intelligence (PAMI), 22(11):1330-1334, 2000.

Modul

Calibration

write_pose (: : Pose, PoseFile :)

Speichern einer 3D-Lage (Pose) in eine Textdatei.

write_pose dient zum Abspeichern einer 3D-Lage Pose in eine Textdatei mit dem Namen PoseFile. Die Dateiendung für die 3D-Lage ist in HALCON 'dat'.

Eine 3D-Lage (Pose) beschreibt eine starre 3D-Transformation, d.h. eine Transformation bestehend aus einer beliebigen Translation und Rotation, mit 6 Parametern, drei für die Translation, drei für die Rotation. Mit Hilfe eines siebten Parameters kann zwischen verschiedenen Pose-Typen unterschieden werden (siehe create_pose).

Eine mit write_pose erzeugte Datei hat z.B. folgendes Aussehen:

```
# 3D POSE PARAMETERS: rotation and translation

# Used representation type:
f 0

# Rotation angles [deg] or Rodriguez vector:
r -17.8134 1.83816 0.288092

# Translation vector (x y z [m]):
t 0.280164 0.150644 1.7554
```

Parameter

▷ **Pose** (input_control) ..... pose ⇝ *real* / integer
3D-Lage (Pose).
Parameteranzahl: 7

▷ **PoseFile** (input_control) ..... filename.write ⇝ *string*
Dateiname der Kameraparameterdatei.
Defaultwert: 'campose.dat'
Wertevorschläge: PoseFile ∈ {'campose.dat', 'campose_initial.dat', 'campose_final.dat'}
Dateiendung: .dat

Beispiel

```
* Perform hand-eye calibration.
calibrate_hand_eye (CalibDataID, Errors)
* Query the camera parameters and the poses.
get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)
* Get poses computed by the hand eye calibration.
get_calib_data (CalibDataID, 'camera', 0, 'base_in_cam_pose', BaseInCamPose)
get_calib_data (CalibDataID, 'calib_obj', 0,\
                'obj_in_tool_pose', ObjInToolPose)
* Store the camera parameters to file.
write_cam_par (CamParam, DataNameStart + 'final_campar.dat')
* Save the hand eye calibration results to file.
write_pose (BaseInCamPose, DataNameStart + 'final_pose_cam_base.dat')
write_pose (ObjInToolPose, DataNameStart + 'final_pose_tool_calplate.dat')
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich geschrieben werden, dann liefert write_pose den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

camera_calibration, get_calib_data, hom_mat3d_to_pose

Siehe auch

create_pose, find_marks_and_pose, camera_calibration, disp_caltab, sim_caltab, read_pose, pose_to_hom_mat3d, hom_mat3d_to_pose

Modul

Foundation

27.5 Quaternionen

axis_angle_to_quat (: : AxisX, AxisY, AxisZ, Angle : Quaternion)

Generiert eine Rotationsquaternion.

Quaternionen sind eine Erweiterung der komplexen Zahlen. Die Menge der Quaternionen ist gegeben durch $\{x_0 + x_1 * i + x_2 * j + x_3 * k | x_0, x_1, x_2, x_3 \in R\}$. In HALCON werden Quaternionen durch Tupel der Länge vier dargestellt: $[x_0, x_1, x_2, x_3]$.

Einheitsquaternionen können verwendet werden, um Rotationen zu beschreiben. Eine Quaternion ist eine Einheitsquaternion, wenn ihre Norm 1 beträgt. Eine Rotation gegen den Uhrzeigersinn um einen Einheitsvektor v (AxisX,AxisY,AxisZ) um den Winkel a (Angle) kann durch die Einheitsquaternion $q = (cos(a/2), sin(a/2)v)$ beschrieben werden.

Der Operator axis_angle_to_quat kann genutzt werden, um solch eine Quaternion zu erzeugen. Um einen Punkt mittels Quaternion zu rotieren, kann der Operator quat_rotate_point_3d verwendet werden.

Zwei Rotation können mit quat_compose multipliziert werden. Außerdem können Rotationsquaternionen dazu verwendet werden, in set_paint die Orientierung des 3D-Plots zu bestimmen.

Achtung

Der Operator axis_angle_to_quat überprüft nicht, ob der Vektor (AxisX, AxisY, AxisZ) ein Einheitsvektor ist (d.h. Länge 1 hat). Falls dies nicht der Fall ist, ist Quaternion keine gültige Rotationsquaternion.

Parameter

- ▷ **AxisX** (input_control) ..... real ⇝ *real* / integer
  X-Komponente der Rotationsachse.
- ▷ **AxisY** (input_control) ..... real ⇝ *real* / integer
  Y-Komponente der Rotationsachse.
- ▷ **AxisZ** (input_control) ..... real ⇝ *real* / integer
  Z-Komponente der Rotationsachse.
- ▷ **Angle** (input_control) ..... real ⇝ *real* / integer
  Rotationswinkel im Bogenmaß.
- ▷ **Quaternion** (output_control) ..... quaternion ⇝ *real* / integer
  Rotationsquaternion.

Beispiel

```
** Normalize a vector and create a rotation quaternion
Length := sqrt(AxisX*AxisX+AxisY*AxisY+AxisZ*AxisZ)
AxisX := AxisX/Length
AxisY := AxisY/Length
AxisZ := AxisZ/Length
axis_angle_to_quat (AxisX, AxisY, AxisZ, rad(90), Quaternion)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Nachfolger

quat_to_hom_mat3d, quat_to_pose, quat_rotate_point_3d, quat_compose

Siehe auch

quat_normalize, quat_conjugate, quat_interpolate, screw_to_dual_quat

Modul

Foundation

deserialize_quat (: : SerializedItemHandle : Quaternion)

Deserialisieren einer serialisierten Quaternion.

deserialize_quat deserialisiert eine Quaternion, welche mit serialize_quat serialisiert wurde (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte Quaternion wird in dem Handle SerializedItemHandle übergeben. Die deserialisierten Werte werden in Quaternion gespeichert.

Parameter

▷ **SerializedItemHandle** (input_control) ..... serialized_item ⇝ *handle*
Handle des serialisierten Elements.

▷ **Quaternion** (output_control) ..... quaternion ⇝ *real* / integer
Quaternion.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert deserialize_quat den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fread_serialized_item, receive_serialized_item, serialize_quat

Nachfolger

quat_to_hom_mat3d, quat_to_pose, quat_rotate_point_3d, quat_compose

Modul

Foundation

quat_compose (: : QuaternionLeft,
QuaternionRight : QuaternionComposed)

Multiplikation zweier Quaternionen.

Der Operator quat_compose multipliziert die zwei Quaternionen QuaternionLeft und QuaternionRight und gibt das Ergebnis als QuaternionComposed zurück.

Eine Quaternion x ist gegeben durch $x = x_0 + x_1 i + x_2 j + x_3 k$. In HALCON wird eine Quaternion durch ein Tupel mit vier Werten repräsentiert: $[x_0, x_1, x_2, x_3]$.

Das Produkt zweier Quaternionen x and y ist wie folgt definiert:

$$\begin{aligned} xy \quad = \quad & (x_0y_0 - x_1y_1 - x_2y_2 - x_3y_3) + \\ & (x_0y_1 + x_1y_0 + x_2y_3 - x_3y_2)i + \\ & (x_0y_2 - x_1y_3 + x_2y_0 + x_3y_1)j + \\ & (x_0y_3 + x_1y_2 - x_2y_1 + x_3y_0)k \end{aligned}$$

Daraus folgt, dass die Multiplikation zweier Quaternionen nicht kommutativ ist, d.h. $xy \neq yx$.

Parameter

▷ **QuaternionLeft** (input_control) . . . . . quaternion ⇝ *real*
Linke Quaternion.

▷ **QuaternionRight** (input_control) . . . . . quaternion ⇝ *real*
Rechte Quaternion.

▷ **QuaternionComposed** (output_control) . . . . . quaternion ⇝ *real*
Produkt der Eingabequaternionen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

axis_angle_to_quat

Nachfolger

quat_to_hom_mat3d, quat_to_pose, quat_rotate_point_3d

Siehe auch

axis_angle_to_quat, quat_to_hom_mat3d, quat_rotate_point_3d, quat_normalize, quat_conjugate, dual_quat_compose

Modul

Foundation

quat_conjugate (: : Quaternion : ConjugatedQuaternion)

Erzeuge die Konjugation einer Quaternion.

Der Operator quat_conjugate erzeugt die Konjugation ConjugatedQuaternion der Eingabequaternion Quaternion.

Die Konjugation einer Quaternion $q = x_0 + x_1 i + x_2 j + x_3 k$ ist gegeben durch $\bar{q} = x_0 - x_1 i - x_2 j - x_3 k$.

Parameter

▷ **Quaternion** (input_control) . . . . . quaternion ⇝ *real*
Eingabequaternion.

▷ **ConjugatedQuaternion** (output_control) . . . . . quaternion ⇝ *real*
Konjugierte Quaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

axis_angle_to_quat

Nachfolger

quat_to_hom_mat3d, quat_rotate_point_3d

Siehe auch

axis_angle_to_quat, quat_to_hom_mat3d, quat_rotate_point_3d, quat_normalize, quat_compose, quat_interpolate, dual_quat_conjugate

Modul

Foundation

quat_interpolate (: : QuaternionStart, QuaternionEnd, InterpPos : QuaternionInterpolated)

Interpolation von zwei Quaternionen.

Der Operator quat_interpolate interpoliert die beiden Quaternionen QuaternionStart und QuaternionEnd an der Interpolationsposition InterpPos. Diese Position muss im Intervall $[0, 1]$ liegen. Für denn Fall InterpPos $= 0$ entspricht QuaternionInterpolated der Eingabequaternion QuaternionStart. Für den Fall InterpPos $= 1$ entspricht QuaternionInterpolated der Eingabequaternion QuaternionEnd.

Die Interpolation erfolgt durch sphärische lineare Interpolation. Sind QuaternionStart und QuaternionEnd Rotationsquaternionen, so ist auch QuaternionInterpolated eine Rotationsquaternion. Wird InterpPos konstant inkrementiert, so wandert ein Punkt auf der Einheitskugel, der mit QuaternionInterpolated rotiert wird, mit konstanter Geschwindigkeit auf einem Bogen auf der Einheitskugel.

Parameter

- ▷ **QuaternionStart** (input_control) . . . . . quaternion ⇝ *real*
  Startquaternion.
- ▷ **QuaternionEnd** (input_control) . . . . . quaternion ⇝ *real*
  Endquaternion.
- ▷ **InterpPos** (input_control) . . . . . real-array ⇝ *real*
  Interpolationsparameter.
  Defaultwert: 0.5
  Wertevorschläge: InterpPos ∈ {0.0, 0.25, 0.5, 0.75, 1.0}
- ▷ **QuaternionInterpolated** (output_control) . . . . . quaternion ⇝ *real*
  Interpolierte Quaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

axis_angle_to_quat, quat_normalize

Nachfolger

quat_to_hom_mat3d, quat_rotate_point_3d, quat_to_pose

Siehe auch

quat_rotate_point_3d, quat_conjugate, quat_compose, dual_quat_interpolate

Modul

Foundation

quat_normalize (: : Quaternion : NormalizedQuaternion)

Normalisiert eine Quaternion.

Der Operator quat_normalize skaliert die Eingabequaternion Quaternion so, dass deren Norm 1 beträgt. Das Ergebnis wird in NormalizedQuaternion zurückgegeben.

Parameter

- ▷ **Quaternion** (input_control) . . . . . quaternion ⇝ *real*
  Eingabequaternion.
- ▷ **NormalizedQuaternion** (output_control) . . . . . quaternion ⇝ *real*
  Normalisierte Quaternion.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`axis_angle_to_quat`

Nachfolger

`quat_to_hom_mat3d`, `quat_rotate_point_3d`

Siehe auch

`axis_angle_to_quat`, `quat_interpolate`, `quat_compose`, `quat_conjugate`, `dual_quat_to_hom_mat3d`

Modul

Foundation

quat_rotate_point_3d (: : Quaternion, Px, Py, Pz : Qx, Qy, Qz)

Rotation eines Punktes mit einer Einheitsquaternion.

Der Operator `quat_rotate_point_3d` rotiert den Punkt (`Px`, `Py`, `Pz`) mit der Rotationsquaternion `Quaternion` (z.B. erzeugt durch `axis_angle_to_quat`). Der rotierte Punkt wird als (`Qx`, `Qy`, `Qz`) zurückgegeben.

Die Rotation eines Punktes x durch eine Einheitsquaternion q ist definiert durch

$x' = qx\bar{q}$.

$\bar{q}$ entspricht der Konjugation von q (siehe `quat_conjugate`). Der Punkt x wird intern in Quaternionenform umgewandelt, damit obige Gleichung angewandt werden kann. Dabei wird der Realteil der Quaternion auf Null gesetzt und die Komponenten von x den drei imaginären Komponenten der Quaternion zugewiesen.

Achtung

Der Operator `quat_rotate_point_3d` überprüft nicht, ob `Quaternion` eine Einheitsquaternion ist. Falls `Quaternion` keine Einheitsquaternion ist, ist das Ergebnis dieses Operators nicht definiert.

Parameter

- ▷ **Quaternion** (input_control) . . . . . quaternion ⇝ *real*
  Rotationsquaternion.
- ▷ **Px** (input_control) . . . . . real ⇝ *real*
  X-Koordinate des zu drehenden Punktes.
- ▷ **Py** (input_control) . . . . . real ⇝ *real*
  Y-Koordinate des zu drehenden Punktes.
- ▷ **Pz** (input_control) . . . . . real ⇝ *real*
  Z-Koordinate des zu drehenden Punktes.
- ▷ **Qx** (output_control) . . . . . real ⇝ *real*
  X-Koordinate des gedrehten Punktes.
- ▷ **Qy** (output_control) . . . . . real ⇝ *real*
  Y-Koordinate des gedrehten Punktes.
- ▷ **Qz** (output_control) . . . . . real ⇝ *real*
  Z-Koordinate des gedrehten Punktes.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

axis_angle_to_quat

Siehe auch

axis_angle_to_quat, quat_to_hom_mat3d, quat_compose, quat_normalize, quat_conjugate, quat_interpolate, dual_quat_trans_line_3d

Modul

Foundation

quat_to_hom_mat3d (: : Quaternion : RotationMatrix)

Wandelt eine Quaternion in die entsprechende Rotationsmatrix um.

Der Operator quat_to_hom_mat3d wandelt eine Einheitsquaternion Quaternion in die entsprechende Rotationsmatrix RotationMatrix um.

Die Rotationsmatrix für eine Quaternion $x = x_0 + x_1 i + x_2 j + x_3 k$ ist gegeben durch:

$$\begin{pmatrix} x_0^2 + x_1^2 - x_2^2 - x_3^2 & 2x_1x_2 - 2x_0x_3 & 2x_0x_2 + 2x_1x_3 \\ 2x_0x_3 + 2x_1x_2 & x_0^2 - x_1^2 + x_2^2 - x_3^2 & 2x_2x_3 - 2x_0x_1 \\ 2x_1x_3 - 2x_0x_2 & 2x_0x_1 + 2x_2x_3 & x_0^2 - x_1^2 - x_2^2 + x_3^2 \end{pmatrix}$$

Achtung

RotationMatrix ist nur dann eine Rotationsmatrix, wenn Quaternion eine Einheitsquaternion ist.

Parameter

- ▷ **Quaternion** (input_control) ..... quaternion ⇝ *real*
  Rotationsquaternion.
- ▷ **RotationMatrix** (output_control) ..... hom_mat3d ⇝ *real*
  Rotationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

axis_angle_to_quat, quat_compose

Nachfolger

affine_trans_point_3d

Siehe auch

axis_angle_to_quat, quat_rotate_point_3d, quat_to_pose, quat_normalize, quat_conjugate, quat_interpolate

Modul

Foundation

serialize_quat (: : Quaternion : SerializedItemHandle)

Serialisieren einer Quaternion.

serialize_quat serialisiert die Daten der in Quaternion übergebenen Quaternion (siehe fwrite_serialized_item für eine Einführung in die Grundlagen der Serialisierung). Die serialisierte Quaternion wird in dem Handle SerializedItemHandle zurückgegeben und kann mit deserialize_quat wieder deserialisiert werden.

Parameter

▷ **Quaternion** (input_control) . . . . . quaternion ⇝ *real* / integer
Quaternion.

▷ **SerializedItemHandle** (output_control) . . . . . serialized_item ⇝ *handle*
Handle des serialisierten Elements.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `serialize_quat` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`axis_angle_to_quat`

Nachfolger

`fwrite_serialized_item`, `send_serialized_item`, `deserialize_quat`

Modul

Foundation

27.6 Sonstiges

convert_point_3d_cart_to_spher (: : X, Y, Z, EquatPlaneNormal, ZeroMeridian : Longitude, Latitude, Radius)

Konvertieren kartesischer Koordinaten eines 3D-Punktes in sphärische Koordinaten.

Der Operator `convert_point_3d_cart_to_spher` konvertiert die kartesischen Koordinaten eines 3D-Punktes, die in `X`, `Y` und `Z` gegeben sind, in sphärische Koordinaten. Die sphärischen Koordinaten werden in `Longitude` (geographische Länge), `Latitude` (geographische Breite) und `Radius` zurückgegeben. Der Wertebereich von `Longitude` ist $[-\pi, +\pi]$, der von `Latitude` ist $[-\pi/2, +\pi/2]$. Dabei hat der Nordpol die Breite $\pi/2$, der Südpol entsprechend die Breite $-\pi/2$.

Die Lage des sphärischen Koordinatensystems bezüglich des kartesischen Koordinatensystems kann mit den Parametern `EquatPlaneNormal` und `ZeroMeridian` festgelegt werden.

`EquatPlaneNormal` bestimmt dabei die Normale der Äquatorebene (Länge == 0) in Richtung Nordpol (positive Breite) und kann die folgenden Werte annehmen:

'x': Die Äquatorebene ist die yz-Ebene. Die positive x-Achse zeigt zum Nordpol.

'-x': Die Äquatorebene ist die yz-Ebene. Die positive x-Achse zeigt zum Südpol.

'y': Die Äquatorebene ist die xz-Ebene. Die positive y-Achse zeigt zum Nordpol.

'-y': Die Äquatorebene ist die xz-Ebene. Die positive y-Achse zeigt zum Südpol.

'z': Die Äquatorebene ist die xy-Ebene. Die positive z-Achse zeigt zum Nordpol.

'-z': Die Äquatorebene ist die xy-Ebene. Die positive z-Achse zeigt zum Südpol.

Die Lage des Nullmeridians kann mit dem Parameter `ZeroMeridian` festgelegt werden. Dazu muss die (in der Äquatorebene liegende) Koordinatenachse angegeben werden, die zum Nullmeridian zeigen soll. Folgende Werte für `ZeroMeridian` sind zulässig:

'x': Die *positive* x-Achse zeigt in Richtung des Nullmeridians.

'-x': Die *negative* x-Achse zeigt in Richtung des Nullmeridians.

'y': Die *positive* y-Achse zeigt in Richtung des Nullmeridians.

'-y': Die *negative* y-Achse zeigt in Richtung des Nullmeridians.

'z': Die *positive* z-Achse zeigt in Richtung des Nullmeridians.

'-z': Die *negative* z-Achse zeigt in Richtung des Nullmeridians.

Es sind nur sinnvolle Kombinationen von `EquatPlaneNormal` und `ZeroMeridian` zulässig, d.h. die Normale der Äquatorebene darf nicht parallel zur Richtung des Nullmeridians sein. Demnach ist beispielsweise die Kombination `EquatPlaneNormal`=*'y'* und `ZeroMeridian`=*'-y'* ungültig.

Beachte: Um eine konsistente Rückkonvertierung von sphärischen in kartesische Koordinaten mit `convert_point_3d_spher_to_cart` zu gewährleisten, müssen für `EquatPlaneNormal` und `ZeroMeridian` die selben Werte angegeben werden wie bei `convert_point_3d_cart_to_spher`.

Der Operator `convert_point_3d_cart_to_spher` kann z.B. dazu genutzt werden, eine gegebene Kameraposition in sphärische Koordinaten umzurechnen. Werden mehrere Kamerapositionen umgerechnet, erhält man somit einen Pose-Bereich (in sphärischen Koordinaten), der in `create_shape_model_3d` zur Erzeugung eines 3D-Formmodells übergeben werden kann.

Parameter

- ▷ **X** (input_control) ..... real(-array) ⇝ *real*
  X-Koordinate des 3D-Punktes.
- ▷ **Y** (input_control) ..... real(-array) ⇝ *real*
  Y-Koordinate des 3D-Punktes.
- ▷ **Z** (input_control) ..... real(-array) ⇝ *real*
  Z-Koordinate des 3D-Punktes.
- ▷ **EquatPlaneNormal** (input_control) ..... string ⇝ *string*
  Normalenvektor der Äquatorebene (zeigt zum Nordpol).
  Defaultwert: '-y'
  Werteliste: `EquatPlaneNormal` ∈ {'x', 'y', 'z', '-x', '-y', '-z'}
- ▷ **ZeroMeridian** (input_control) ..... string ⇝ *string*
  Koordinatenachse in der Äquatorebene, die zum Nullmeridian zeigt.
  Defaultwert: '-z'
  Werteliste: `ZeroMeridian` ∈ {'x', 'y', 'z', '-x', '-y', '-z'}
- ▷ **Longitude** (output_control) ..... angle.rad(-array) ⇝ *real*
  Geographische Länge des 3D-Punktes.
- ▷ **Latitude** (output_control) ..... angle.rad(-array) ⇝ *real*
  Geographische Breite des 3D-Punktes.
- ▷ **Radius** (output_control) ..... real(-array) ⇝ *real*
  Radius des 3D-Punktes.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `convert_point_3d_cart_to_spher` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`create_shape_model_3d`, `find_shape_model_3d`

Siehe auch

`convert_point_3d_spher_to_cart`

Modul

3D Metrology

convert_point_3d_spher_to_cart (: : Longitude, Latitude, Radius, EquatPlaneNormal, ZeroMeridian : X, Y, Z)

Konvertieren sphärischer Koordinaten eines 3D-Punktes in kartesische Koordinaten.

Der Operator convert_point_3d_spher_to_cart konvertiert die sphärischen Koordinaten eines 3D-Punktes, die in Longitude (geographische Länge), Latitude (geographische Breite) und Radius gegeben sind, in kartesische Koordinaten X, Y und Z. Die sphärischen Koordinaten Longitude und Latitude müssen im Bogenmaß übergeben werden. Der Parameter Latitude muss dabei im Bereich $[-\pi/2, +\pi/2]$ liegen, wobei der Nordpol die Breite $\pi/2$ und der Südpol entsprechend die Breite $-\pi/2$ hat.

Die Lage des sphärischen Koordinatensystems bezüglich des kartesischen Koordinatensystems kann mit den Parametern EquatPlaneNormal und ZeroMeridian festgelegt werden.

EquatPlaneNormal bestimmt dabei die Normale der Äquatorebene (Länge == 0) in Richtung Nordpol (positive Breite) und kann die folgenden Werte annehmen:

'x': Die Äquatorebene ist die yz-Ebene. Die positive x-Achse zeigt zum Nordpol.

'-x': Die Äquatorebene ist die yz-Ebene. Die positive x-Achse zeigt zum Südpol.

'y': Die Äquatorebene ist die xz-Ebene. Die positive y-Achse zeigt zum Nordpol.

'-y': Die Äquatorebene ist die xz-Ebene. Die positive y-Achse zeigt zum Südpol.

'z': Die Äquatorebene ist die xy-Ebene. Die positive z-Achse zeigt zum Nordpol.

'-z': Die Äquatorebene ist die xy-Ebene. Die positive z-Achse zeigt zum Südpol.

Die Lage des Nullmeridians kann mit dem Parameter ZeroMeridian festgelegt werden. Dazu muss die (in der Äquatorebene liegende) Koordinatenachse angegeben werden, die zum Nullmeridian zeigen soll. Folgende Werte für ZeroMeridian sind zulässig:

'x': Die *positive* x-Achse zeigt in Richtung des Nullmeridians.

'-x': Die *negative* x-Achse zeigt in Richtung des Nullmeridians.

'y': Die *positive* y-Achse zeigt in Richtung des Nullmeridians.

'-y': Die *negative* y-Achse zeigt in Richtung des Nullmeridians.

'z': Die *positive* z-Achse zeigt in Richtung des Nullmeridians.

'-z': Die *negative* z-Achse zeigt in Richtung des Nullmeridians.

Es sind nur sinnvolle Kombinationen von EquatPlaneNormal und ZeroMeridian zulässig, d.h. die Normale der Äquatorebene darf nicht parallel zur Richtung des Nullmeridians sein. Demnach ist beispielsweise die Kombination EquatPlaneNormal='*y*' und ZeroMeridian='*-y*' ungültig.

Beachte: Um eine konsistente Rückkonvertierung von kartesischen in sphärische Koordinaten mit convert_point_3d_cart_to_spher zu gewährleisten, müssen für EquatPlaneNormal und ZeroMeridian die selben Werte angegeben werden wie bei convert_point_3d_spher_to_cart.

Der Operator convert_point_3d_spher_to_cart kann z.B. dazu genutzt werden, eine in sphärischen Koordinaten gegebene Kameraposition in kartesische Koordinaten umzurechnen. Das Ergebnis kann dann mit create_cam_pose_look_at_point in eine vollständige Kamera-Pose umgerechnet werden.

Parameter

▷ **Longitude** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Geographische Länge des 3D-Punktes.

▷ **Latitude** (input_control) . . . . . angle.rad(-array) ⇝ *real*
Geographische Breite des 3D-Punktes.
Restriktion: - pi / 2 <= Latitude && Latitude <= pi / 2

▷ **Radius** (input_control) . . . . . real(-array) ⇝ *real*
Radius des 3D-Punktes.

▷ **EquatPlaneNormal** (input_control) . . . . . string ⇝ *string*
Normalenvektor der Äquatorebene (zeigt zum Nordpol).
Defaultwert: '-y'
Werteliste: EquatPlaneNormal ∈ {'x', 'y', 'z', '-x', '-y', '-z'}

▷ **ZeroMeridian** (input_control) ..... string ⇝ *string*
Koordinatenachse in der Äquatorebene, die zum Nullmeridian zeigt.
Defaultwert: '-z'
Werteliste: `ZeroMeridian` ∈ {'x', 'y', 'z', '-x', '-y', '-z'}

▷ **X** (output_control) ..... real(-array) ⇝ *real*
X-Koordinate des 3D-Punktes.

▷ **Y** (output_control) ..... real(-array) ⇝ *real*
Y-Koordinate des 3D-Punktes.

▷ **Z** (output_control) ..... real(-array) ⇝ *real*
Z-Koordinate des 3D-Punktes.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `convert_point_3d_spher_to_cart` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_shape_model_3d_params`

Siehe auch

`convert_point_3d_cart_to_spher`

Modul

3D Metrology

Kapitel 28

Tupel

28.1 Arithmetik

tuple_abs (: : T : Abs)

Berechnen des Absolutbetrags eines Tupels.

tuple_abs berechnet den Absolutbetrag des Eingabetupels T. Der Absolutbetrag einer ganzen Zahl ist wieder eine ganze Zahl. Der Absolutbetrag einer Gleitpunktzahl ist eine Gleitpunktzahl. Der Absolutbetrag von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_abs, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Abs := abs(T)
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Abs** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Absolutbetrag des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_fabs

Modul

Foundation

tuple_acos (: : T : ACos)

Berechnen des Arkuskosinus eines Tupels.

tuple_acos berechnet den Arkuskosinus des Eingabetupels T. Der Arkuskosinus wird in jedem Fall als Gleitpunktzahl in ACos zurückgeliefert. Die Winkel in ACos werden im Bogenmaß gespeichert. Der Arkuskosinus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_acos`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
ACos := acos(T)
```

Parameter

- ▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel.
  Restriktion: -1 <= T <= 1
- ▷ **ACos** (output_control) ..... angle.rad(-array) ⇝ *real*
  Arkuskosinus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_asin`, `tuple_atan`, `tuple_atan2`

Siehe auch

`tuple_cos`, `tuple_cosh`, `tuple_acosh`

Modul

Foundation

tuple_acosh (: : T : Acosh)

Berechnen des Areakosinus hyperbolicus eines Tupels.

`tuple_acosh` berechnet den Areakosinus hyperbolicus des Eingabetupels `T`. Der Areakosinus hyperbolicus wird in jedem Fall als Gleitpunktzahl in `Acosh` zurückgeliefert. Der Areakosinus hyperbolicus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_acosh`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Acosh := acosh(T)
```

Parameter

- ▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Acosh** (output_control) ..... number(-array) ⇝ *real*
  Areakosinus hyperbolicus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_asinh, tuple_atanh

Siehe auch

tuple_cosh, tuple_cos, tuple_acos

Modul

Foundation

tuple_add (: : S1, S2 : Sum)

Addieren zweier Tupel.

tuple_add berechnet die Summe der Eingabetupel S1 und S2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel addiert. Ansonsten muss entweder S1 oder S2 die Länge 1 haben. In diesem Fall wird die Addition für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Falls zwei ganze Zahlen addiert werden, ist das Ergebnis wieder eine ganze Zahl. Falls eine Gleitpunktzahl mit einer anderen Zahl addiert wird, ist das Ergebnis eine Gleitpunktzahl. Falls zwei Strings addiert werden, entspricht die Addition einer String-Konkatenation. Falls eine Zahl und ein String addiert werden, wird die Zahl in einen String umgewandelt. Die Addition entspricht so auch der String-Konkatenation.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_add, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Sum := S1 + S2
```

Die + Operation entspricht einer String-Konkatenation, falls S1 und S2 Strings enthalten.

Parameter

- ▷ **S1** (input_control) ..... tuple(-array) ⇝ *real* / integer / string
  Eingabetupel 1.
- ▷ **S2** (input_control) ..... tuple(-array) ⇝ *real* / integer / string
  Eingabetupel 2.
- ▷ **Sum** (output_control) ..... tuple(-array) ⇝ *real* / integer / string
  Summe der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sub

Siehe auch

tuple_cumul

Modul

Foundation

tuple_asin (: : T : ASin)

Berechnen des Arkussinus eines Tupels.

tuple_asin berechnet den Arkussinus des Eingabetupels T. Der Arkussinus wird in jedem Fall als Gleitpunktzahl in ASin zurückgeliefert. Die Winkel in ASin werden im Bogenmaß gespeichert. Der Arkussinus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_asin`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
ASin := asin(T)
```

Parameter

▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
Eingabetupel.
Restriktion: -1 <= T <= 1

▷ **ASin** (output_control) ..... angle.rad(-array) ⇝ *real*
Arkussinus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_acos`, `tuple_atan`, `tuple_atan2`

Siehe auch

`tuple_sin`, `tuple_sinh`, `tuple_asinh`

Modul

Foundation

tuple_asinh (: : T : Asinh)

Berechnen des Areasinus hyperbolicus eines Tupels.

`tuple_asinh` berechnet den Areasinus hyperbolicus des Eingabetupels `T`. Der Areasinus hyperbolicus wird in jedem Fall als Gleitpunktzahl in `Asinh` zurückgeliefert. Der Areasinus hyperbolicus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_asinh`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Asinh := asinh(T)
```

Parameter

▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Asinh** (output_control) ..... number(-array) ⇝ *real*
Areasinus hyperbolicus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_acosh, tuple_atanh

Siehe auch

tuple_asin, tuple_sin, tuple_asin

Modul

Foundation

tuple_atan (: : T : ATan)

Berechnen des Arkustangens eines Tupels.

tuple_atan berechnet den Arkustangens des Eingabetupels T. Der Arkustangens wird in jedem Fall als Gleitpunktzahl in ATan zurückgeliefert. Die Winkel in ATan werden im Bogenmaß gespeichert. Der Arkustangens von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_atan, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
ATan := atan(T)
```

Parameter

- ▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **ATan** (output_control) ..... angle.rad(-array) ⇝ *real*
  Arkustangens des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_atan2, tuple_asin, tuple_acos

Siehe auch

tuple_tan, tuple_tanh, tuple_atanh

Modul

Foundation

tuple_atan2 (: : Y, X : ATan)

Berechnen des Arkustangens eines Tupels für alle vier Quadranten.

tuple_atan2 berechnet den Arkustangens der Eingabetupel Y/X mit richtiger Behandlung aller vier Quadranten. Der Arkustangens wird in jedem Fall als Gleitpunktzahl in ATan zurückgeliefert. Die Winkel in ATan werden im Bogenmaß berechnet. Der Arkustangens von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_atan2, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
ATan := atan2(Y, X)
```

Parameter

- ▷ **Y** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel der y-Werte.
- ▷ **X** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel der x-Werte.
- ▷ **ATan** (output_control) . . . . . angle.rad(-array) ⇝ *real*
  Arkustangens des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_atan, tuple_asin, tuple_acos

Siehe auch

tuple_tan, tuple_tanh

Modul

Foundation

tuple_atanh (: : T : Atanh)

Berechnen des Areatangens hyperbolicus eines Tupels.

tuple_atanh berechnet den Areatangens hyperbolicus des Eingabetupels T. Der Areatangens hyperbolicus wird in jedem Fall als Gleitpunktzahl in Atanh zurückgeliefert. Der Areatangens hyperbolicus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_atanh, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Atanh := atanh(T)
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Atanh** (output_control) . . . . . number(-array) ⇝ *real*
  Areatangens hyperbolicus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_asinh, tuple_acosh

Siehe auch

tuple_tanh, tuple_tan, tuple_atan, tuple_atan2

Modul

Foundation

tuple_cbrt (: : T : Cbrt)

Berechnen der Quadratwurzel eines Tupels.

tuple_cbrt berechnet die Kubikwurzel des Eingabetupels T. Die Kubikwurzel wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Kubikwurzel von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_cbrt, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Cbrt := cbrt(T)
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Cbrt** (output_control) . . . . . number(-array) ⇝ *real*
  Kubikwurzel des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_pow

Siehe auch

tuple_sqrt

Modul

Foundation

tuple_ceil (: : T : Ceil)

Berechnen der Ceiling-Funktion eines Tupels.

tuple_ceil berechnet die Ceiling-Funktion des Eingabetupels T, d.h. die kleinste ganze Zahl größer gleich T. Die Ceiling-Funktion wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Ceiling-Funktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_ceil, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Ceil := ceil(T)
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Ceil** (output_control) . . . . . number(-array) ⇝ *real*
  Ceiling-Funktion des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

tuple_cos (: : T : Cos)

Berechnen des Kosinus eines Tupels.

tuple_cos berechnet den Kosinus des Eingabetupels T. Der Kosinus wird in jedem Fall als Gleitpunktzahl in Cos zurückgeliefert. Die Winkel in T werden im Bogenmaß gespeichert. Der Kosinus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_cos, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Cos := cos(T)
```

Parameter

- ▷ **T** (input_control) ..... angle.rad(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Cos** (output_control) ..... number(-array) ⇝ *real*
  Kosinus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sin, tuple_tan

Siehe auch

tuple_acos, tuple_cosh, tuple_acosh

Modul

Foundation

tuple_cosh (: : T : Cosh)

Berechnen des Kosinus hyperbolicus eines Tupels.

tuple_cosh berechnet den Kosinus hyperbolicus des Eingabetupels T. Der Kosinus hyperbolicus wird in jedem Fall als Gleitpunktzahl in Cosh zurückgeliefert. Der Kosinus hyperbolicus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_cosh, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Cosh := cosh(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Cosh** (output_control) . . . . . number(-array) ⇝ *real*
Kosinus hyperbolicus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sinh, tuple_tanh

Siehe auch

tuple_acosh, tuple_cos, tuple_acos

Modul

Foundation

tuple_cumul (: : Tuple : Cumul)

Berechnen der kumulativen Summen eines Tupels.

tuple_cumul berechnet die kumulativen Summen aus den jeweiligen Elementen des Eingabetupels Tuple und liefert sie in dem Ausgabetupel Cumul zurück, d.h. das i-te Element des Ausgabetupels Cumul ist die Summe der ersten i Elemente des Eingabetupels Tuple. Besteht Tuple zum Beispiel aus den Werten [1,2,3,4], so beinhaltet das Ausgabetupel Cumul die Werte [1,3,6,10]. Das Eingabetupel muss hierfür ausschließlich (ganze oder Gleitkomma-)Zahlen beinhalten. Die einzelnen Ergebnisse bestehen aus einer Gleitkommazahl, sobald mindestens ein Element der Berechnung eine Gleitkommazahl ist. Falls alle Elemente einer Berechnung hingegen ganze Zahlen sind, so besteht auch das entsprechende Summenelement in Cumul aus einer ganzen Zahl.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_cumul, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Cumul := cumul(Tuple)
```

Parameter

▷ **Tuple** (input_control) . . . . . number(-array) ⇝ *integer* / real
Eingabetupel.

▷ **Cumul** (output_control) . . . . . number(-array) ⇝ *real* / integer
Kumulative Summe der entsprechenden Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sum

Siehe auch

tuple_min2, tuple_max2, tuple_add

Modul

Foundation

tuple_deg (: : Rad : Deg)

Konvertieren eines Tupels von Bogenmaß nach Grad.

tuple_deg konvertiert das Eingabetupel Rad von Bogenmaß in Grad. Das Ergebnis wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Konversion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_deg, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Deg := deg(Rad)
```

Parameter

- ▷ **Rad** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Deg** (output_control) . . . . . number(-array) ⇝ *real*
  Eingabetupel in Grad.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

tuple_rad

Modul

Foundation

tuple_div (: : Q1, Q2 : Quot)

Dividieren zweier Tupel.

tuple_div berechnet den Quotienten der Eingabetupel Q1 und Q2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel dividiert. Ansonsten muss entweder Q1 oder Q2 die Länge 1 haben. In diesem Fall wird die Division für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Falls zwei ganze Zahlen dividiert werden, ist das Ergebnis wieder eine ganze Zahl. Falls einer der Operanden eine Gleitpunktzahl ist, ist das Ergebnis eine Gleitpunktzahl. Die Division von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_div, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Quot := Q1 / Q2
```

Parameter

- ▷ **Q1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **Q2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
  Restriktion: Q2 != 0

▷ **Quot** (output_control) . . . . . number(-array) ⇝ *real* / integer
Quotient der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_mult

Modul

Foundation

tuple_erf (: : T : Erf)

Berechnen der Fehlerfunktion eines Tupels.

tuple_erf berechnet die Fehlerfunktion des Eingabetupels T. Die Fehlerfunktion ist definiert als:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Der Wert der Fehlerfunktion wird in jedem Fall als Gleitpunktzahl in Erf zurückgeliefert. Die Fehlerfunktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_erf, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Erf := erf(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Erf** (output_control) . . . . . number(-array) ⇝ *real*
Wert der Fehlerfunktion des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_erfc

Siehe auch

tuple_exp

Modul

Foundation

tuple_erfc (: : T : Erfc)

Berechnen der komplementären Fehlerfunktion eines Tupels.

tuple_erfc berechnet die komplementäre Fehlerfunktion des Eingabetupels T. Die komplementäre Fehlerfunktion ist definiert als $1 - \text{erf}(x)$ (für die Definition von $\text{erf}(x)$ siehe tuple_erf). Der Wert der komplementären Fehlerfunktion wird in jedem Fall als Gleitpunktzahl in Erfc zurückgeliefert. Die komplementäre Fehlerfunktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_erfc, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Erfc := erfc(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Erfc** (output_control) . . . . . number(-array) ⇝ *real*
Wert der komplementären Fehlerfunktion des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_erf

Siehe auch

tuple_exp

Modul

Foundation

tuple_exp (: : T : Exp)

Berechnen der Exponentialfunktion eines Tupels.

tuple_exp berechnet die Exponentialfunktion des Eingabetupels T. Die Exponentialfunktion wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Exponentialfunktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_exp, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Exp := exp(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Exp** (output_control) . . . . . number(-array) ⇝ *real*
Exponentialfunktion des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_exp2, tuple_exp10, tuple_pow

Siehe auch

tuple_log, tuple_log2, tuple_log10

Modul

Foundation

tuple_exp10 (: : T : Exp)

Berechnen der Exponentialfunktion zur Basis 10 eines Tupels.

tuple_exp10 berechnet die Exponentialfunktion zur Basis 10 des Eingabetupels T. Die Exponentialfunktion wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Exponentialfunktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_exp10, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Exp := exp10(T)
```

Parameter

▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Exp** (output_control) ..... number(-array) ⇝ *real*
Exponentialfunktion zur Basis 10 des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_exp, tuple_exp2, tuple_pow

Siehe auch

tuple_log, tuple_log2, tuple_log10

Modul

Foundation

tuple_exp2 (: : T : Exp)

Berechnen der Exponentialfunktion zur Basis 2 eines Tupels.

tuple_exp2 berechnet die Exponentialfunktion zur Basis 2 des Eingabetupels T. Die Exponentialfunktion wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Exponentialfunktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_exp2, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Exp := exp2(T)
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Exp** (output_control) . . . . . number(-array) ⇝ *real*
  Exponentialfunktion zur Basis 2 des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_exp, tuple_exp10, tuple_pow

Siehe auch

tuple_log, tuple_log2, tuple_log10

Modul

Foundation

tuple_fabs (: : T : Abs)

Berechnen des Absolutbetrags eines Tupels (als Gleitkommazahlen).

tuple_fabs berechnet den Absolutbetrag des Eingabetupels T. Im Gegensatz zu tuple_abs wird der Absolutbetrag von tuple_fabs immer als eine Gleitpunktzahl zurückgeliefert. Der Absolutbetrag von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_fabs, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Abs := fabs(T)
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Abs** (output_control) . . . . . number(-array) ⇝ *real*
  Absolutbetrag des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_abs

Modul

Foundation

tuple_floor (: : T : Floor)

Berechnen der Floor-Funktion eines Tupels.

tuple_floor berechnet die Floor-Funktion des Eingabetupels T, d.h. die größte ganze Zahl kleiner gleich T. Die Floor-Funktion wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Floor-Funktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_floor, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Floor := floor(T)
```

Parameter

- ▷ **T** (input_control) . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Floor** (output_control) . . . number(-array) ⇝ *real*
  Floor-Funktion des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_ceil

Modul

Foundation

tuple_fmod (: : T1, T2 : Fmod)

Berechnen des Rests der Gleitpunkt-Division zweier Tupel.

tuple_fmod berechnet den Rest der Gleitpunkt-Division der Eingabetupel T1/T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel dividiert. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Division für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Das Ergebnis ist in jedem Fall eine Gleitpunktzahl. Die Division von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_fmod, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Fmod := fmod(T1, T2)
```

Parameter

- ▷ **T1** (input_control) . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
  Restriktion: T2 != 0.0

▷ **Fmod** (output_control) . . . . . number(-array) ⇝ *real*
Rest der Division der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_mod

Siehe auch

tuple_floor, tuple_ceil

Modul

Foundation

tuple_hypot (: : T1, T2 : Hypot)

Berechnen der Hypotenuse zweier Tupel.

tuple_hypot berechnet die Hypotenuse $\sqrt{T1^2 + T2^2}$ der Eingabetupel. Die Berechnung wird so durchgeführt, dass kein unnötiger arithmetischer Über- oder Unterlauf in den Zwischenschritten der Berechnung auftritt. Falls beide Tupel dieselbe Länge haben, wird die Hypotenuse der entsprechenden Elemente der beiden Tupel berechnet. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Hypotenuse für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels berechnet. Das Ergebnis ist in jedem Fall eine Gleitpunktzahl. Die Hypotenuse von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_hypot, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Hypot := hypot(T1, T2)
```

Parameter

▷ **T1** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel 1.

▷ **T2** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel 2.

▷ **Hypot** (output_control) . . . . . number(-array) ⇝ *real*
Hypotenuse der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sqrt

Siehe auch

tuple_mult, tuple_add

Modul

Foundation

tuple_ldexp (: : T1, T2 : Ldexp)

Berechnen der Ldexp-Funktion zweier Tupel.

`tuple_ldexp` berechnet die Ldexp-Funktion der Eingabetupel, d.h. `T1` $\cdot 2^{floor(\texttt{T2})}$. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel verwendet. Ansonsten muss entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Das Ergebnis ist in jedem Fall eine Gleitpunktzahl. Die Ldexp-Funktion von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_ldexp`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Ldexp := ldexp(T1, T2)
```

Parameter

- ▷ **T1** (input_control) . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
- ▷ **Ldexp** (output_control) . . . number(-array) ⇝ *real*
  Ldexp-Funktion der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`tuple_exp`

Modul

Foundation

tuple_lgamma (: : T : LogGamma)

Berechnen des Logarithmus' des Absolutbetrags des Werts der Gammafunktion eines Tupels.

`tuple_lgamma` berechnet den Logarithmus des Absolutbetrags des Werts der Gammafunktion des Eingabetupels `T` (für die Definition der Gammafunktion siehe `tuple_tgamma`). Sie ist definiert für jede reelle Zahl außer für nichtpositive ganze Zahlen. Der Logarithmus des Absolutbetrags des Werts der Gammafunktion wird in jedem Fall als Gleitpunktzahl in `LogGamma` zurückgeliefert. Der Logarithmus des Absolutbetrags des Werts der Gammafunktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_lgamma`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
LogGamma := lgamma(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.
▷ **LogGamma** (output_control) . . . . . number(-array) ⇝ *real*
Logarithmus des Absolutbetrags des Werts der Gammafunktion des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_tgamma

Modul

Foundation

tuple_log (: : T : Log)

Berechnen des natürlichen Logarithmus eines Tupels.

tuple_log berechnet den natürlichen Logarithmus des Eingabetupels T. Der natürliche Logarithmus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der natürliche Logarithmus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_log, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Log := log(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.
Restriktion: T > 0
▷ **Log** (output_control) . . . . . number(-array) ⇝ *real*
Natürlicher Logarithmus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_log2, tuple_log10

Siehe auch

tuple_exp, tuple_exp2, tuple_exp10, tuple_pow

Modul

Foundation

tuple_log10 (: : T : Log)

Berechnen des Logarithmus zur Basis 10 eines Tupels.

tuple_log10 berechnet den Logarithmus zur Basis 10 des Eingabetupels T. Der Logarithmus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Logarithmus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_log10, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Log := log10(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.
Restriktion: T > 0

▷ **Log** (output_control) . . . . . number(-array) ⇝ *real*
Logarithmus zur Basis 10 des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_log, tuple_log2

Siehe auch

tuple_exp, tuple_exp2, tuple_exp10, tuple_pow

Modul

Foundation

tuple_log2 (: : T : Log)

Berechnen des Logarithmus zur Basis 2 eines Tupels.

tuple_log2 berechnet den Logarithmus zur Basis 2 des Eingabetupels T. Der Logarithmus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Logarithmus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_log2, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Log := log2(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.
Restriktion: T > 0

▷ **Log** (output_control) . . . . . number(-array) ⇝ *real*
Logarithmus zur Basis 2 des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_log, tuple_log10

Siehe auch

tuple_exp, tuple_exp2, tuple_exp10, tuple_pow

Modul

Foundation

tuple_max2 (: : T1, T2 : Max2)

Bestimmen des elementweisen Maximums zweier Tupel.

tuple_max2 berechnet das Maximum der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel miteinander verglichen und die jeweiligen Maxima in das Ausgabetupel Max2 übernommen. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels verglichen. Eine Mischung aus Strings und Zahlen ist erlaubt, wenn die miteinander verglichenen Elemente jeweils denselben Typ haben.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_max2, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Max2 := max2(T1, T2)
```

Parameter

- ▷ **T1** (input_control) . . . . . tuple(-array) ⇝ *real* / integer / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . tuple(-array) ⇝ *real* / integer / string
  Eingabetupel 2.
- ▷ **Max2** (output_control) . . . . . tuple(-array) ⇝ *real* / integer / string
  Elementweises Maximum der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

tuple_min2, tuple_min, tuple_max, tuple_cumul

Modul

Foundation

tuple_min2 (: : T1, T2 : Min2)

Bestimmen des elementweisen Minimums zweier Tupel.

tuple_min2 berechnet das Minimum der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel miteinander verglichen und die jeweiligen Minima in das Ausgabetupel Min2 übernommen. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels verglichen. Eine Mischung aus Strings und Zahlen ist erlaubt, wenn die miteinander verglichenen Elemente jeweils denselben Typ haben.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_min2`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Min2 := min2(T1, T2)
```

Parameter

- ▷ **T1** (input_control) . . . . . tuple(-array) ⇝ *real* / integer / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . tuple(-array) ⇝ *real* / integer / string
  Eingabetupel 2.
- ▷ **Min2** (output_control) . . . . . tuple(-array) ⇝ *real* / integer / string
  Elementweises Minimum der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

`tuple_max2`, `tuple_min`, `tuple_max`

Modul

Foundation

tuple_mod (: : T1, T2 : Mod)

Berechnen des Rests der ganzzahligen Division zweier Tupel.

`tuple_mod` berechnet den Rest der ganzzahligen Division der Eingabetupel `T1`/`T2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel dividiert und der Rest in das Ausgabetupel `Mod` übernommen. Ansonsten muss entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Division für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Das Ergebnis ist in jedem Fall eine ganze Zahl. Die Division von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_mod`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Mod := T1 % T2
```

Parameter

- ▷ **T1** (input_control) . . . . . number(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . number(-array) ⇝ *integer*
  Eingabetupel 2.
  Restriktion: T2 != 0
- ▷ **Mod** (output_control) . . . . . number(-array) ⇝ *integer*
  Rest der Division der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

tuple_fmod, tuple_div

Modul

Foundation

tuple_mult (: : P1, P2 : Prod)

Multiplizieren zweier Tupel.

tuple_mult berechnet das Produkt der Eingabetupel P1 und P2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel multipliziert. Ansonsten muss entweder P1 oder P2 die Länge 1 haben. In diesem Fall wird die Multiplikation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Falls zwei ganze Zahlen multipliziert werden, ist das Ergebnis wieder eine ganze Zahl. Falls einer der Operanden eine Gleitpunktzahl ist, ist das Ergebnis eine Gleitpunktzahl. Die Multiplikation von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_mult, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Prod := P1 * P2
```

Parameter

- ▷ **P1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **P2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
- ▷ **Prod** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Produkt der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_div

Modul

Foundation

tuple_neg (: : T : Neg)

Negieren eines Tupels.

tuple_neg berechnet die Negierung des Eingabetupels T, d.h. Neg $= -$T. Die Negierung einer ganzen Zahl ist wieder eine ganze Zahl. Die Negierung einer Gleitpunktzahl ist eine Gleitpunktzahl. Die Negierung von Strings ist nicht erlaubt. Die Negierung eines leeren Eingabtupels ergibt wiederum ein leeres Tupel als Ausgabe.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_neg`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Neg := -T
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Neg** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Negation des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

tuple_pow (: : T1, T2 : Pow)

Berechnen der Potenzierung zweier Tupel.

`tuple_pow` berechnet die Potenzierung der Eingabetupel `T1` ^ `T2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel potenziert. Ansonsten muss entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Potenzierung für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Das Ergebnis ist in jedem Fall eine Gleitpunktzahl. Die Potenzierung von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_pow`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Pow := pow(T1, T2)
```

Parameter

- ▷ **T1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
- ▷ **Pow** (output_control) . . . . . number(-array) ⇝ *real*
  Potenzfunktion der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_exp`

Siehe auch

`tuple_log`, `tuple_log10`

Modul

Foundation

tuple_rad (: : Deg : Rad)

Konvertieren eines Tupels von Grad nach Bogenmaß.

tuple_rad konvertiert das Eingabetupel Deg von Grad in Bogenmaß. Das Ergebnis wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Konversion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_rad, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Rad := rad(Deg)
```

Parameter

- ▷ **Deg** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Rad** (output_control) . . . . . number(-array) ⇝ *real*
  Eingabetupel im Bogenmaß.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

tuple_deg

Modul

Foundation

tuple_sgn (: : T : Sgn)

Berechnen des Vorzeichens eines Tupels.

tuple_sgn berechnet das Vorzeichen der Elemente des Eingabetupels T und gibt sie als ganze Zahlen (-1,0,1) im Ausgabetupel Sgn zurück. Strings sind im Eingabetupel nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_sgn, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Sgn := sgn(T)
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Sgn** (output_control) . . . . . number(-array) ⇝ *integer*
  Vorzeichen des Eingabetupels als ganze Zahlen.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

tuple_sin (: : T : Sin)

Berechnen des Sinus eines Tupels.

tuple_sin berechnet den Sinus des Eingabetupels T. Der Sinus wird in jedem Fall als Gleitpunktzahl in Sin zurückgeliefert. Die Winkel in T werden im Bogenmaß gespeichert. Der Sinus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_sin, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Sin := sin(T)
```

Parameter

▷ **T** (input_control) ..... angle.rad(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Sin** (output_control) ..... number(-array) ⇝ *real*
Sinus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_cos, tuple_tan

Siehe auch

tuple_asin, tuple_sinh, tuple_asinh

Modul

Foundation

tuple_sinh (: : T : Sinh)

Berechnen des Sinus hyperbolicus eines Tupels.

tuple_sinh berechnet den Sinus hyperbolicus des Eingabetupels T. Der Sinus hyperbolicus wird in jedem Fall als Gleitpunktzahl in Sinh zurückgeliefert. Der Sinus hyperbolicus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_sinh, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Sinh := sinh(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Sinh** (output_control) . . . . . number(-array) ⇝ *real*
Sinus hyperbolicus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_cosh, tuple_tanh

Siehe auch

tuple_asinh, tuple_sin, tuple_asin

Modul

Foundation

tuple_sqrt (: : T : Sqrt)

Berechnen der Quadratwurzel eines Tupels.

tuple_sqrt berechnet die Quadratwurzel des Eingabetupels T. Die Quadratwurzel wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Quadratwurzel von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_sqrt, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Sqrt := sqrt(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.
Restriktion: T >= 0

▷ **Sqrt** (output_control) . . . . . number(-array) ⇝ *real*
Quadratwurzel des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_pow

Siehe auch

tuple_cbrt, tuple_hypot

Modul

Foundation

tuple_sub (: : D1, D2 : Diff)

Subtrahieren zweier Tupel.

tuple_sub berechnet die Differenz der Eingabetupel D1 und D2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel subtrahiert. Ansonsten muss entweder D1 oder D2 die Länge 1 haben. In diesem Fall wird die Subtraktion für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Falls zwei ganze Zahlen subtrahiert werden, ist das Ergebnis wieder eine ganze Zahl. Falls einer der Operanden eine Gleitpunktzahl ist, ist das Ergebnis eine Gleitpunktzahl. Die Subtraktion von Strings ist nicht erlaubt.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_sub, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Diff := D1 - D2
```

Parameter

- ▷ **D1** (input_control) . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **D2** (input_control) . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
- ▷ **Diff** (output_control) . . . number(-array) ⇝ *real* / integer
  Differenz der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_add

Modul

Foundation

tuple_tan (: : T : Tan)

Berechnen des Tangens eines Tupels.

tuple_tan berechnet den Tangens des Eingabetupels T. Der Tangens wird in jedem Fall als Gleitpunktzahl in Tan zurückgeliefert. Die Winkel in T werden im Bogenmaß gespeichert. Der Tangens von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_tan, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Tan := tan(T)
```

Parameter

▷ **T** (input_control) . . . . . angle.rad(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Tan** (output_control) . . . . . number(-array) ⇝ *real*
Tangens des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sin, tuple_cos

Siehe auch

tuple_atan, tuple_atan2, tuple_tanh, tuple_atanh

Modul

Foundation

tuple_tanh (: : T : Tanh)

Berechnen des Tangens hyperbolicus eines Tupels.

tuple_tanh berechnet den Tangens hyperbolicus des Eingabetupels T. Der Tangens hyperbolicus wird in jedem Fall als Gleitpunktzahl in Tanh zurückgeliefert. Der Tangens hyperbolicus von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_tanh, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Tanh := tanh(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Tanh** (output_control) . . . . . number(-array) ⇝ *real*
Tangens hyperbolicus des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sinh, tuple_cosh

Siehe auch

tuple_atanh, tuple_tan, tuple_atan, tuple_atan2

Modul

Foundation

tuple_tgamma (: : T : Gamma)

Berechnen der Gammafunktion eines Tupels.

tuple_tgamma berechnet die Gammafunktion des Eingabetupels T. Die Gammafunktion ist definiert als:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt.$$

Sie ist definiert für jede reelle Zahl außer für nichtpositive ganze Zahlen. Für nicht-negative ganze Zahlen m gilt

$$\Gamma(m+1) = m!$$

und für allgemeine reelle Zahlen x:

$$\Gamma(x+1) = x\Gamma(x).$$

Der Wert der Gammafunktion wird in jedem Fall als Gleitpunktzahl in Gamma zurückgeliefert. Die Gammafunktion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_tgamma, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Gamma := tgamma(T)
```

Parameter

▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Gamma** (output_control) . . . . . number(-array) ⇝ *real*
Wert der Gammafunktion des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_lgamma

Modul

Foundation

28.2 Bitoperationen

tuple_band (: : T1, T2 : BAnd)

Berechnen des bitweisen Und zweier Tupel.

tuple_band berechnet das bitweise Und der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_band`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
BAnd := T1 band T2
```

Parameter

- ▷ **T1** (input_control) . . . integer(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . integer(-array) ⇝ *integer*
  Eingabetupel 2.
- ▷ **BAnd** (output_control) . . . integer(-array) ⇝ *integer*
  Binäres Und der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_bor`, `tuple_bxor`, `tuple_bnot`

Siehe auch

`tuple_and`, `tuple_or`, `tuple_xor`, `tuple_not`

Modul

Foundation

tuple_bnot (: : T : BNot)

Berechnen des bitweisen Nicht eines Tupels.

`tuple_bnot` berechnet das bitweise Nicht des Eingabetupels `T`. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_bnot`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
BNot := bnot T
```

Parameter

- ▷ **T** (input_control) . . . integer(-array) ⇝ *integer*
  Eingabetupel.
- ▷ **BNot** (output_control) . . . integer(-array) ⇝ *integer*
  Binäres Nicht des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_band, tuple_bor, tuple_bxor

Siehe auch

tuple_and, tuple_or, tuple_xor, tuple_not

Modul

Foundation

tuple_bor (: : T1, T2 : BOr)

Berechnen des bitweisen Oder zweier Tupel.

tuple_bor berechnet das bitweise Oder der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_bor, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
BOr := T1 bor T2
```

Parameter

- ▷ **T1** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 2.
- ▷ **BOr** (output_control) ..... integer(-array) ⇝ *integer*
  Binäres Oder der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_band, tuple_bxor, tuple_bnot

Siehe auch

tuple_and, tuple_or, tuple_xor, tuple_not

Modul

Foundation

tuple_bxor (: : T1, T2 : BXor)

Berechnen des bitweisen Exklusiv-Oder zweier Tupel.

tuple_bxor berechnet das bitweise Exklusiv-Oder der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_bxor`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
BXor := T1 bxor T2
```

Parameter

- ▷ **T1** (input_control) . . . integer(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . integer(-array) ⇝ *integer*
  Eingabetupel 2.
- ▷ **BXor** (output_control) . . . integer(-array) ⇝ *integer*
  Binäres Exklusiv-Oder der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_band`, `tuple_bor`, `tuple_bnot`

Siehe auch

`tuple_and`, `tuple_or`, `tuple_xor`, `tuple_not`

Modul

Foundation

tuple_lsh (: : T, Shift : Lsh)

Bitweises Verschieben eines Tupels nach links.

`tuple_lsh` verschiebt das Tupel `T` bitweise um `Shift` Stellen nach links. Falls kein Überlauf eintritt, ist diese Operation äquivalent zu einer Multiplikation mit $2^{\texttt{Shift}}$. Falls `T` negativ ist, hängt das Ergebnis von der Rechnerarchitektur ab. Das Ergebnis ist undefiniert, falls `Shift` negativ oder größer als 32 ist. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muss entweder `T` oder `Shift` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_lsh`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Lsh := lsh(T, Shift)
```

Parameter

- ▷ **T** (input_control) . . . integer(-array) ⇝ *integer*
  Eingabetupel.
- ▷ **Shift** (input_control) . . . integer(-array) ⇝ *integer*
  Anzahl Stellen, um die verschoben werden soll.
- ▷ **Lsh** (output_control) . . . integer(-array) ⇝ *integer*
  Verschobenes Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_mult

Siehe auch

tuple_rsh

Modul

Foundation

tuple_rsh (: : T, Shift : Rsh)

Bitweises Verschieben eines Tupels nach rechts.

tuple_rsh verschiebt das Tupel T bitweise um Shift Stellen nach rechts. Diese Operation ist äquivalent zu einer Division mit 2^{Shift}. Falls T negativ ist, hängt das Ergebnis von der Rechnerarchitektur ab. Das Ergebnis ist undefiniert, falls Shift negativ oder größer als 32 ist. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muss entweder T oder Shift die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leere Eingabetupel

Falls einer oder beide Eingabetupel leer sind, wird ein leeres Tupel zurückgegeben.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_rsh, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Rsh := rsh(T, Shift)
```

Parameter

- ▷ **T** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel.
- ▷ **Shift** (input_control) ..... integer(-array) ⇝ *integer*
  Anzahl Stellen, um die verschoben werden soll.
- ▷ **Rsh** (output_control) ..... integer(-array) ⇝ *integer*
  Verschobenes Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_div

Siehe auch

tuple_lsh

Modul

Foundation

28.3 Datencontainer

copy_dict (: : DictHandle, GenParamName,
GenParamValue : CopiedDictHandle)

Kopieren eines Dictionaries.

copy_dict erstellt eine tiefe Kopie (d.h. von allen im Dictionary enthaltenen Handles und ikonischen Objekten wird eine tiefe Kopie erstellt) des Dictionary DictHandle und gibt diese als Dictionary in CopiedDictHandle zurück.

Wenn DictHandle ein Handle enthält, das nicht kopiert werden kann oder das schon freigegeben wurde, bricht copy_dict standardmäßig mit einer Fehlermeldung ab. Dieses Verhalten kann durch Setzen von GenParamName auf *'raise_error_if_content_not_serializable'* verändert werden, der zugehörige GenParamValue kann folgende Werte annehmen:

'true': Der Default: Fehler werden geworfen und der Kopierprozess abgebrochen.

'low_level': Nur Low-Level-Fehler werden geworfen. Anstatt des betreffenden Handles wird ein leeres Handle in CopiedDictHandle kopiert und der Kopierprozess weitergeführt. Der Umgang mit HALCON Low-Level-Fehlern wird durch *'do_low_error'* in set_system bestimmt.

'false': Es werden keine Fehler geworfen. Anstatt des betreffenden Handles wird ein leeres Handle in CopiedDictHandle kopiert und der Kopierprozess weitergeführt.

Parameter

▷ **DictHandle** (input_control) . . . . . dict ⇝ *handle*
Handle des Dictionaries.
Parameteranzahl: DictHandle == 1

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: []
Werteliste: GenParamName ∈ {'raise_error_if_content_not_serializable'}

▷ **GenParamValue** (input_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Wert des generischen Parameters.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {'true', 'false', 'low_level'}

▷ **CopiedDictHandle** (output_control) . . . . . dict ⇝ *handle*
Kopiertes Handle des Dictionaries.
Parameteranzahl: CopiedDictHandle == 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert copy_dict den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_dict, deserialize_handle, create_dict

Nachfolger

write_dict

Siehe auch

read_dict, write_dict, serialize_handle, deserialize_handle

Modul

Foundation

create_dict (: : : DictHandle)

Erzeugen eines neuen Dictionaries.

create_dict erzeugt ein neues, leeres Dictionary und gibt dessen Handle in DictHandle zurück.

Ein Dictionary ist ein Objekt, das ähnlich einem assoziativem Array (Wörterbuch) aufgebaut ist. Es erlaubt eine beliebige Anzahl von Daten zu speichern, welche mit Hilfe eines eindeutigen Schlüssels (String oder Integer) identifiziert werden. Jeder Schlüssel kann entweder einen Tupeldaten-Parameter oder ein ikonisches Objekt referenzieren. Diese Daten werden in dem Dictionary mit Hilfe der Operatoren `set_dict_object` und `set_dict_tuple` gespeichert und können über die Operatoren `get_dict_object` und `get_dict_tuple` wieder ausgelesen werden.

Die Daten, welche in dem Dictionary referenziert werden, sind stets Kopien ihrer Originale. Damit können die originalen Daten, nachdem sie in dem Dictionary durch die Operatoren `set_dict_tuple` oder `set_dict_object` gespeichert wurden, weiterverwendet werden. Dabei gelten folgende Besonderheiten:

- Objekte: Die Kopien sind Referenzen auf die Originale, wie in `copy_obj`. Insbesondere wirken sich Änderungen durch die Operatoren `set_grayval` oder `overpaint_region` auch auf das im Dictionary gespeicherte Objekt aus.
- Handles: Beim Speichern eines Handles in einem Dictionary wird nur der Handle-Wert, nicht aber das Handle-Objekt selbst kopiert (keine "deep copy").

Parameter

▷ **`DictHandle`** (output_control) ..... dict ⇝ *handle*
Handle des neuen Dictionaries.
Parameteranzahl: DictHandle == 1

Beispiel

```
Dicts := []
for idx := 0 to 4 by 1
  create_dict (DictHandle)
  Dicts[idx] := DictHandle
endfor
* ...
```

Ergebnis

`create_dict` liefert den Wert TRUE, sofern kein Allokierungsfehler auftritt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Nachfolger

`set_dict_tuple`, `set_dict_object`

Siehe auch

`set_dict_tuple`, `get_dict_tuple`, `set_dict_object`, `get_dict_object`, `get_dict_param`

Modul

Foundation

`get_dict_object` (: Object : DictHandle, Key :)

Zurückgeben der in einem Dictionary von einem Schlüssel referenzierten ikonischen Objekte.

`get_dict_object` fragt aus dem Dictionary `DictHandle` das mit dem Schlüssel `Key` assoziierte ikonische Objekt `Object` ab. Das ikonische Objekt muss zuvor mit Hilfe des Operators `set_dict_object` in dem Dictionary abgespeichert worden sein.

Falls der spezifizierte Schlüssel Key nicht in dem Dictionary hinterlegt ist oder die mit ihm verbundenen Daten kein ikonisches Objekt sondern ein Datentupel ist, schlägt get_dict_object fehl. Die gespeicherten Schlüssel und die Information über die von ihnen referenzierten Daten kann über den Operator get_dict_param abgefragt werden.

Parameter

▷ **Object** (output_object) ..... object(-array) ⇝ *object*
Vom Schlüssel referenziertes Objekt.

▷ **DictHandle** (input_control) ..... dict ⇝ *handle*
Handle des Dictionaries.
Parameteranzahl: DictHandle == 1

▷ **Key** (input_control) ..... string ⇝ *string* / integer
Schlüssel.
Parameteranzahl: Key == 1
Restriktion: length(Key) > 0

Beispiel

```
* ...
get_dict_param (Dict, 'key_exists', ['simple_string','foo','my_image'], \
                KeysPresence)
get_dict_param (Dict, 'key_data_type', ['simple_string','my_image'], \
                KeysType)
get_dict_object (Image, Dict, 'my_image')
```

Ergebnis

Im Erfolgsfall gibt get_dict_object den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler sind ungültige Parameter (Dictionary oder Schlüssel), Abwesenheit des Schlüssels in dem Dictionary oder wenn der Schüssel andere Datentypen als ein ikonisches Objekt referenziert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_dict_object

Nachfolger

get_dict_object

Alternativen

get_dict_tuple

Siehe auch

create_dict, set_dict_tuple, get_dict_tuple, set_dict_object, remove_dict_key, get_dict_param

Modul

Foundation

get_dict_param (: : DictHandle, GenParamName, Key : GenParamValue)

Abfragen von Parametern oder Informationen eines Dictionaries.

get_dict_param fragt die aktuellen Werte der Parameter eines Dictionaries oder anderweitige Informationen zum Status eines Dictionaries ab.

Jeder Aufruf von get_dict_param darf nur einen einzelnen Parameternamen in GenParamName enthalten. Es gibt zwei unterschiedliche Arten von Abfragen:

- Parameter, die auf das gesamte Dictionary angewendet werden. In diesem Fall dürfen keine Schlüssel übergeben werden, d.h. dem Parameter `Key` muss ein leeres Tupel übergeben werden.
- Parameter, die auf einzelne Schlüssel angewendet werden. In diesem Fall müssen die Schlüssel in dem Parameter `Key` übergeben werden, auf die der Parameter angewendet werden soll. Die Schlüssel werden dabei in der gleichen Reihenfolge abgearbeitet wie sie dem Parameter `Key` übergeben werden.

Parameternamen, die unabhängig von Schlüsseln angewendet werden:

'keys': Fragt alle Schlüssel ab, für die in dem Dictionary Daten hinterlegt sind, unabhängig von dem Typ der mit ihnen verbundenen Daten. Die Liste der Schlüssel wird über den Parameter `GenParamValue` als ein Tupel von Strings oder Integern zurückgegeben. Für diese Abfrage muss dem Parameter `Key` ein leeres Tupel übergeben werden.

Parametername, die Schlüssel-spezifisch angewendet werden:

'key_exists': Gibt *1* zurück, wenn zu dem übergebenen Schlüssel in dem Dictionary Daten hinterlegt sind, sonst *0*. Die Ergebnisse werden im Parameter `GenParamValue` zurückgegeben, jeweils ein Wert pro übergebenem Schlüssel.

'key_data_type': Gibt den Wert *'tuple'* für Schlüssel zurück, die in dem Dictionary Tupeldaten referenzieren. Die Tupeldaten selbst erhält man mit dem Operator `get_dict_tuple`. Der Wert *'object'* wird für Schlüssel zurückgegeben, welche Objektdaten referenzieren. Die Objektdaten selbst werden vom Operator `get_dict_object` zurückgegeben. Die Ergebnisse werden im Parameter `GenParamValue` zurückgegeben, jeweils ein Wert pro Schlüssel. Dieser Parameter kann verwendet werden um dynamisch zu entscheiden, ob `get_dict_tuple` oder `get_dict_object` verwendet werden muss um die Daten eines speziellen Schlüssels zu erhalten.

Parameter

▷ **`DictHandle`** (input_control) . . . . . dict ⇝ *handle*
Handle des Dictionaries.
Parameteranzahl: DictHandle == 1

▷ **`GenParamName`** (input_control) . . . . . string ⇝ *string*
Name des Parameters.
Parameteranzahl: GenParamName == GenParamValue
Defaultwert: 'keys'
Werteliste: `GenParamName` ∈ {'keys', 'key_exists', 'key_data_type'}

▷ **`Key`** (input_control) . . . . . string(-array) ⇝ *string* / integer
Schlüssel auf den der Parameter angewendet werden soll (leer für `GenParamName` = *'keys'*).

▷ **`GenParamValue`** (output_control) . . . . . tuple(-array) ⇝ *string* / integer / real
Wert der Parameterabfrage.

Beispiel

```
get_dict_param (Dict, 'keys', [], AllKeys)
get_dict_param (Dict, 'key_data_type', AllKeys, KeysType)
```

Ergebnis

Wenn alle Parameter und übergebenen Schlüssel gültig sind, gibt `get_dict_param` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`create_dict`

Siehe auch

create_dict, set_dict_tuple, get_dict_tuple, set_dict_object, get_dict_object, remove_dict_key

Modul

Foundation

get_dict_tuple (: : DictHandle, Key : Tuple)

Zurückgeben der in einem Dictionary über einen Schüssel referenzierten Tupeldaten.

get_dict_tuple fragt aus dem Dictionary DictHandle die mit einem Schlüssel Key assoziierten Tupeldaten Tuple ab. Die Tupeldaten müssen zuvor mit Hilfe des Operators set_dict_tuple in dem Dictionary abgespeichert worden sein.

Falls der spezifizierte Schlüssel Key nicht in dem Dictionary hinterlegt ist oder der mit ihm verbundene Eintrag kein Datentupel sondern ein ikonisches Objekt ist, schlägt get_dict_tuple fehl. Die gespeicherten Schlüssel und die Information über die von ihnen referenzierten Daten kann über den Operator get_dict_param abgefragt werden.

Parameter

- ▷ **DictHandle** (input_control) ..... dict ⇝ *handle*
  Handle des Dictionaries.
  Parameteranzahl: DictHandle == 1
- ▷ **Key** (input_control) ..... string ⇝ *string* / integer
  Schlüssel.
  Parameteranzahl: Key == 1
  Restriktion: length(Key) > 0
- ▷ **Tuple** (output_control) ..... tuple(-array) ⇝ *string* / integer / real
  Vom Schlüssel referenzierte Tupeldaten.

Beispiel

```
* ...
get_dict_param (Dict, 'key_exists', ['simple_string','foo','my_image'], \
                KeysPresence)
get_dict_param (Dict, 'key_data_type', ['simple_string','my_image'], \
                KeysType)
get_dict_tuple (Dict, 'simple_string', TupleString)
```

Ergebnis

Im Erfolgsfall gibt get_dict_tuple den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler sind ungültige Parameter (Dictionary oder Schlüssel), Abwesenheit des Schlüssels in dem Dictionary oder wenn der Schüssel andere Datentypen als ein Datentupel referenziert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

set_dict_tuple

Nachfolger

get_dict_tuple

Alternativen

get_dict_object

Siehe auch

create_dict, set_dict_tuple, set_dict_object, get_dict_object, remove_dict_key, get_dict_param

Modul

Foundation

read_dict (: : FileName, GenParamName, GenParamValue : DictHandle)

Lesen eines Dictionaries aus einer Datei.

read_dict liest ein Dictionary aus der durch FileName gegebenen Datei und gibt das Handle des Dictionaries in DictHandle zurück.

Der Operator unterstützt die folgenden Dateiformate:

'hdict': Binäres HALCON-Format für Dictionaries. Dateien dieses Typs können mit write_dict geschrieben werden. Die Standard-Dateiendung für dieses Format ist 'hdict'.

'json': JSON (JavaScript Object Notation) Dateiformat. Die Standard-Dateiendung für dieses Format ist 'json'.

Falls keine Datei mit dem angegebenen Namen existiert, sucht der Operator durch Anhängen der Standard-Dateiendungen nach der Datei. Der Dateityp wird automatisch aus der Dateiendung und dem Dateiinhalt erkannt (welche übereinstimmen müssen).

Eine Reihe weiterer optionaler Parameter kann gesetzt werden. Die Namen und Werte dieser Parameter werden in GenParamName und GenParamValue übergeben. Einige der Parameter gelten nicht für alle Dateitypen. Die folgenden Werte sind für GenParamName möglich:

'json_value_true', *'json_value_false'*, *'json_value_null'*: Setzt die für JSON-Primitive verwendeten Werte. Gültige JSON-Primitive sind *'true'*, *'false'* und *'null'*. Wenn ein solches Primitiv in einer JSON-Datei gefunden wird, wird der entsprechende Eintrag im Dictionary auf den mit diesen Parametern gesetzten Wert gesetzt. Der Wert wird in GenParamValue übergeben und muss ein Tupel der Länge 1 sein.

Standardmäßig werden die Primitive auf folgende Werte abgebildet: *1* für *'true'*, *0* für *'false'* und *'HNULL'* für *'null'*. Diese Parameter haben nur einen Einfluss, wenn eine JSON-Datei gelesen wird.

Parameter

▷ **FileName** (input_control) . . . . . filename.read ⇝ *string*
Dateiname der zu lesenden Datei.
Dateiendung: .hdict, .json

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: []
Werteliste: GenParamName ∈ {'json_value_true', 'json_value_false', 'json_value_null'}

▷ **GenParamValue** (input_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Wert des generischen Parameters.
Defaultwert: []
Wertevorschläge: GenParamValue ∈ {0, 1, 'HNULL'}

▷ **DictHandle** (output_control) . . . . . dict ⇝ *handle*
Handle des Dictionaries.
Parameteranzahl: DictHandle == 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_dict den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Vorgänger

`write_dict`

Siehe auch

`write_dict`, `serialize_handle`, `deserialize_handle`

Modul

Foundation

`remove_dict_key ( : : DictHandle, Key : )`

Löschen eines Schlüssels aus einem Dictionary.

`remove_dict_key` löscht die in `Key` übergebenen Schlüssel aus dem in `DictHandle` übergebenen Dictionary und gibt die vom Schlüssel referenzierten ikonischen Objekte oder Tupeldaten frei.

Falls ein Fehler während der Verarbeitung eines oder mehrerer Schlüssel auftritt, beispielsweise weil ein Schlüssel nicht im Dictionary existiert, wird versucht, so weit wie möglich alle weiteren Schlüssel zu entfernen bevor der Operator mit dem entsprechenden Fehler zurückkehrt.

Parameter

▷ **`DictHandle`** (input_control) . . . . . dict ⇝ *handle*
Handle des Dictionaries.
Parameteranzahl: DictHandle == 1

▷ **`Key`** (input_control) . . . . . string(-array) ⇝ *string* / integer
Zu löschender Schlüssel.
Restriktion: length(Key) > 0

Beispiel

```
* Remove all keys
get_dict_param (Dict, 'keys', [], Keys)
remove_dict_key (Dict, Keys)
```

Ergebnis

Im Erfolgsfall gibt `remove_dict_key` den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DictHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_dict`

Nachfolger

`get_dict_param`

Siehe auch

create_dict, set_dict_tuple, get_dict_tuple, set_dict_object, get_dict_object, get_dict_param

Modul

Foundation

set_dict_object (Object : : DictHandle, Key :)

Hinzufügen eines Schlüssel-Objekt-Paares zu einem Dictionary.

set_dict_object speichert in einem Dictionary ähnlich zu einem assoziativem Array ein ikonisches Objekt in Verbindung mit einem Schlüssel ab. Das Dictionary wird im Parameter DictHandle übergeben.

Object wird durch die Operation kopiert und kann daher anschließend ohne Seiteneffekte weiterverwendet werden. Dabei werden die Objektdaten in die HALCON-Datenbank kopiert, d.h., das neue Objekt enthält Verweise auf Object, vgl. copy_obj.

Sowohl leere Objekte als auch ikonische Tupelobjekte sind zulässig und können von einem Schlüssel referenziert werden. Falls bereits Daten (Tupel oder Objekt) durch den angegebenen Schlüssel referenziert werden, werden die alten Daten freigegeben und durch Object ersetzt.

Key ist ein String oder eine ganze Zahl (integer). Bei Strings muss die Groß-/Kleinschreibung beachtet.

Die ikonischen Objekte für den übergebenen Schlüssel können mit Hilfe des Operators get_dict_object aus dem Dictionary wieder ausgelesen werden.

Parameter

- ▷ **Object** (input_object) ..... object(-array) ⇝ *object*
  Vom Schlüssel zu referenzierendes ikonisches Objekt.
- ▷ **DictHandle** (input_control) ..... dict ⇝ *handle*
  Handle des Dictionaries.
  Parameteranzahl: DictHandle == 1
- ▷ **Key** (input_control) ..... string ⇝ *string* / integer
  Schlüssel.
  Parameteranzahl: Key == 1
  Restriktion: length(Key) > 0

Beispiel

```
create_dict (Dict)
read_image (Image, 'filename')
set_dict_object (Image, Dict, 'my_image')
```

Ergebnis

Im Erfolgsfall gibt set_dict_tuple den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler sind ungültige Parameter (Dictionary oder Schlüssel) oder ein Allokierungsfehler.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DictHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

create_dict

Nachfolger

set_dict_object, set_dict_tuple

Alternativen

set_dict_tuple

Siehe auch

create_dict, set_dict_tuple, get_dict_tuple, get_dict_object, remove_dict_key, get_dict_param

Modul

Foundation

set_dict_tuple (: : DictHandle, Key, Tuple :)

Hinzufügen eines Schlüssel-Tupeldaten-Paares zu einem Dictionary.

set_dict_tuple speichert in dem Dictionary DictHandle ähnlich zu einem assoziativem Array ein Datentupel in Verbindung mit einem Schlüssel ab.

Tuple wird in das Dictionary kopiert und kann anschließend weiterverwendet oder gelöscht werden. Ein leeres Tupel ist ebenfalls ein gültiger Wert, der mit einem Schlüssel referenziert werden kann. Falls bereits Daten (Tupel oder Objekt) durch den gegebenen Schlüssel referenziert werden, werden die alten Daten freigegeben und durch Tuple ersetzt.

Key ist ein String oder eine ganze Zahl (integer). Bei Strings muss die Groß-/Kleinschreibung beachtet.

Die Tupeldaten für den übergebenen Schlüssel können mit Hilfe des Operators get_dict_tuple aus dem Dictionary wieder ausgelesen werden.

Achtung

Falls Tuple Handles enthält, werden nur die Handle-Werte kopiert. Von Handles, die in dem Tupel enthalten sind, wird keine tiefe Kopie erzeugt.

Parameter

- ▷ **DictHandle** (input_control) ..... dict ⇝ *handle*
  Handle des Dictionaries.
  Parameteranzahl: DictHandle == 1
- ▷ **Key** (input_control) ..... string ⇝ *string* / integer
  Schlüssel.
  Parameteranzahl: Key == 1
  Restriktion: length(Key) > 0
- ▷ **Tuple** (input_control) ..... tuple-array ⇝ *string* / integer / real
  Vom Schlüssel zu referenzierende Tupeldaten.

Beispiel

```
create_dict (Dict)
set_dict_tuple (Dict, 'simple_integer', 27)
set_dict_tuple (Dict, 'simple_string', 'Hello world')
set_dict_tuple (Dict, 'mixed_tuple', ['The answer', 42])
set_dict_tuple (Dict, 0, 'This is zero')
```

Ergebnis

Im Erfolgsfall gibt set_dict_tuple den Wert TRUE zurück. Andernfalls wird eine Fehlerbehandlung durchgeführt. Mögliche Fehler sind ungültige Parameter (Dictionary oder Schlüssel) oder ein Allokierungsfehler.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- DictHandle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Vorgänger

`create_dict`

Nachfolger

`set_dict_tuple`, `set_dict_object`

Alternativen

`set_dict_object`

Siehe auch

`create_dict`, `get_dict_tuple`, `set_dict_object`, `get_dict_object`, `get_dict_param`, `remove_dict_key`

Modul

Foundation

write_dict (: : DictHandle, FileName, GenParamName, GenParamValue :)

Abspeichern eines Dictionaries in einer Datei.

`write_dict` speichert das Dictionary `DictHandle` in der Datei `FileName` ab. Die Datei wird als 'hdict' geschrieben, dem HALCON-Format für Dictionaries. Die Dateiendung für das Dictionary ist in HALCON 'hdict'.

Wenn `DictHandle` ein Handle enthält, das nicht serialisiert werden kann oder das schon freigegeben wurde, bricht `write_dict` standardmäßig mit einer Fehlermeldung ab. Dieses Verhalten kann durch Setzen von `GenParamName` auf *'raise_error_if_content_not_serializable'* verändert werden, der zugehörige `GenParamValue` kann folgende Werte annehmen:

'true': Das Standardverhalten, Fehler werden geworfen.

'false': Es wird kein Fehler geworfen, ein leeres Handle wird anstatt des nicht serialisierbaren Handles geschrieben.

'low_level': Nur Low-Level-Fehler werden geworfen, die das betroffene Handle enthalten. Ein leeres Handle wird anstatt des nicht serialisierbaren Handles geschrieben. Der Umgang mit HALCON Low-Level-Fehlern wird durch *'do_low_error'* in `set_system` bestimmt.

Parameter

▷ **DictHandle** (input_control) . . . . . dict ⇝ *handle*
Handle des Dictionaries.
Parameteranzahl: DictHandle == 1

▷ **FileName** (input_control) . . . . . filename.write ⇝ *string*
Dateiname der zu lesenden Datei.
Dateiendung: `.hdict`

▷ **GenParamName** (input_control) . . . . . attribute.name(-array) ⇝ *string*
Name des generischen Parameters.
Defaultwert: []
Werteliste: `GenParamName` ∈ {'raise_error_if_content_not_serializable'}

▷ **GenParamValue** (input_control) . . . . . attribute.name(-array) ⇝ *string* / integer / real
Wert des generischen Parameters.
Defaultwert: []
Wertevorschläge: `GenParamValue` ∈ {'true', 'false', 'low_level'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_dict` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

read_dict, deserialize_handle, create_dict

Nachfolger

read_dict

Siehe auch

read_dict, serialize_handle, deserialize_handle

Modul

Foundation

28.4 Elementauswahl

tuple_find (: : Tuple, ToFind : Indices)

Suchen eines Tupels in einem anderem Tupel und Rückgabe der Indizes.

tuple_find sucht im Eingabetupel Tuple in aufsteigender Reihenfolge das zweite Eingabetupel ToFind und liefert als Resultat die Indizes der gefundenen Subtupel (bezogen auf das Eingabetupel Tuple) in dem Ergebnistupel Indices zurück. Besteht Tuple zum Beispiel aus den Werten [3,4,5,6,1,2,3,4,0] und enthält ToFind die beiden Werte [3,4], so beinhaltet das Ausgabetupel Indices die Ergebniswerte [0,6]. Falls im ersten Eingabetupel Tuple das zweite Eingabetupel ToFind nicht enthalten ist, so wird -1 im Ergebnistupel Indices zurückgegeben. Die Eingabetupel dürfen Strings und Zahlen auch gemischt enthalten.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_find, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Indices := find(Tuple, ToFind)
```

Parameter

- ▷ **Tuple** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **ToFind** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel mit zu suchenden Werten.
- ▷ **Indices** (output_control) ..... integer(-array) ⇝ *integer*
  Indizes der gefundenen Subtupel in dem zu untersuchenden Tupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_find_first, tuple_find_last, tuple_sort, tuple_inverse, tuple_equal_elem

Modul

Foundation

tuple_find_first (: : Tuple, ToFind : Index)

Suchen nach dem Index des ersten gefundenen Subtupel in einem Tupel.

tuple_find_first durchsucht Tuple in aufsteigender Reihenfolge nach dem zweiten Tupel ToFind und gibt den entsprechenden Index bezüglich Tuple zurück. Beispielsweise liefert tuple_find_first für Tuple mit den Werten [3,4,5,6,1,2,3,4,0] und ToFind mit den Werten [3,4] für Index den Wert 0. Enthält Tuple ToFind nicht, so liefert der Operator für Index den Wert -1. Der Operator akzeptiert gemischte Eingabe-Tupel aus Zahlen und Strings.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_find_first, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Index := find_first(Tuple, ToFind)
```

Parameter

- ▷ **Tuple** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **ToFind** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel mit zu suchenden Werten.
- ▷ **Index** (output_control) ..... integer ⇝ *integer*
  Index des ersten gefundenen Subtupel in dem zu untersuchenden Tupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_find, tuple_find_last, tuple_sort, tuple_equal_elem

Modul

Foundation

tuple_find_last (: : Tuple, ToFind : Index)

Suchen nach dem Index des letzten gefundenen Subtupel in einem Tupel.

tuple_find_last durchsucht Tuple rückwärts, d.h. in absteigender Reihenfolge, nach der ersten Übereinstimmung mit den Werten des zweiten Tupels ToFind und gibt den entsprechenden Index bezüglich Tuple zurück. Beispielsweise liefert tuple_find_last für Tuple mit den Werten [3,4,5,6,1,2,3,4,0] und ToFind mit den Werten [3,4] für Index den Wert 6. Enthält Tuple ToFind nicht, so liefert der Operator für Index den Wert -1. Der Operator akzeptiert gemischte Eingabe-Tupel aus Zahlen und Strings.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_find_last, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Index := find_last(Tuple, ToFind)
```

Parameter

- ▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **ToFind** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel mit zu suchenden Werten.
- ▷ **Index** (output_control) . . . . . integer ⇝ *integer*
  Index des letzten gefundenen Subtupel in dem zu untersuchenden Tupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_find, tuple_find_first, tuple_sort, tuple_equal_elem

Modul

Foundation

tuple_first_n (: : Tuple, Index : Selected)

Auswählen der vorderen Elemente eines Tupels bis zum Index „n“.

tuple_first_n wählt die vorderen Elemente des Tupels Tuple bis zum Index Index aus und liefert sie in dem Ausgabetupel Selected zurück. Der Index eines Tupelelements wird beginnend mit 0 gezählt, das heißt, das erste Tupelelement wird über den Index 0 ausgewählt. Insgesamt werden also Index+1 Elemente zurückgeliefert.

Index muss eine ganze Zahl sein (oder eine Gleitkommazahl, deren Nachkommastellen 0 sind).

Sonderfall: Leeres Eingabetupel

Falls Tuple leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_first_n, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Selected := firstn(Tuple, Index)
```

Parameter

- ▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Index** (input_control) . . . . . integer ⇝ *integer*
  Index des letzten auszuwählenden Elements.
- ▷ **Selected** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Ausgewählte Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_last_n, tuple_select, tuple_str_bit_select, tuple_concat, tuple_select_mask

Siehe auch

tuple_remove

Modul

Foundation

tuple_last_n (: : Tuple, Index : Selected)

Auswählen aller Elemente ab Index „n“ bis um Ende eines Tupels.

tuple_last_n wählt die hinteren Elemente des Tupels Tuple aus und liefert sie in dem Ausgabetupel Selected zurück. Selected enthält dann alle Elemente des Tupels Tuple ab dem „n-ten“ Element (inklusive des „n-ten“ Elements). Der Index „n“ wird durch den Parameter Index festgelegt. Folgerichtig muss Index eine ganze Zahl beinhalten (falls Index eine Gleitkommazahl enthält, so muss diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Der Index eines Tupelelements wird beginnend mit 0 gezählt, das heißt, das erste Tupelelement wird über den Index 0 ausgewählt.

Sonderfall: Leeres Eingabetupel

Falls Tuple leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_last_n, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Selected := lastn(Tuple, Index)
```

Parameter

- ▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Index** (input_control) . . . . . integer ⇝ *integer*
  Index des ersten auszuwählenden Elements.
- ▷ **Selected** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Ausgewählte Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_first_n, tuple_select, tuple_str_bit_select, tuple_concat, tuple_select_mask

Siehe auch

tuple_remove

Modul

Foundation

tuple_select (: : Tuple, Index : Selected)

Auswählen einzelner Elemente aus einem Tupel.

tuple_select wählt einzelne Elemente aus einem Tupel Tuple aus und liefert sie über den Parameter Selected zurück. Der Parameter Index gibt hierbei einen oder mehrere Indizes an, über die festgelegt wird, welche Elemente ausgewählt werden. Folgerichtig darf Index ausschließlich ganze Zahlen beinhalten (falls Index eine Gleitkommazahl enthält, so muss diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Tupelindizes werden beginnend mit 0 gezählt, das heißt, das erste Tupelelement wird über den Index 0 ausgewählt.

Sonderfall: Leeres Eingabetupel

Falls Tuple leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_select, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Selected := Tuple[Index]
Selected := subset(Tuple, Index)
```

Parameter

- ▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Index** (input_control) . . . . . integer(-array) ⇝ *integer*
  Indizes der auszuwählenden Elemente.
- ▷ **Selected** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Ausgewähltes Tupelelement.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_first_n, tuple_last_n, tuple_str_bit_select, tuple_concat, tuple_select_range, tuple_select_rank

Siehe auch

tuple_remove

Modul

Foundation

tuple_select_mask (: : Tuple, Mask : Selected)

Auswählen einzelner Elemente aus einem Tupel anhand einer Maske.

tuple_select_mask wählt einzelne Elemente aus einem Tupel Tuple aus und gibt sie mit dem Parameter Selected zurück. Der Parameter Mask legt für jedes Element fest, ob es ausgewählt wird. Ist der Wert größer als 0, so wird das entsprechende Element in Selected übernommen. Folgerichtig darf Mask ausschließlich Zahlen beinhalten und muss die gleiche Länge wie Tuple haben.

Sonderfall: Leere Eingabetupel

Falls beide Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück. Falls nur eins der Eingabetupel leer ist und das andere nicht, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_select_mask, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Selected := select_mask(Tuple, Mask)
```

Parameter

- ▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Mask** (input_control) . . . . . integer(-array) ⇝ *integer*
  Spezifiziert jeweils mit >0 die auszuwählenden Elemente.
- ▷ **Selected** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Ausgewählte Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_first_n, tuple_last_n, tuple_str_bit_select, tuple_concat, tuple_select, tuple_select_range, tuple_select_rank

Siehe auch

tuple_remove

Modul

Foundation

tuple_select_range (: : Tuple, Leftindex, Rightindex : Selected)

Auswählen mehrerer Elemente eines Tupels.

tuple_select_range wählt mehrere aufeinanderfolgende Elemente eines Tupels Tuple aus und liefert sie über den Parameter Selected zurück. Der Parameter Leftindex gibt hierbei den Index des ersten auszuwählenden Tupelelements und der Parameter Rightindex den Index des letzten auszuwählenden Tupelelements an. Folgerichtig müssen beide Parameter Leftindex und Rightindex eine ganze Zahl beinhalten (falls sie eine Gleitkommazahl enthalten, so muss diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Der Index eines Tupelelements wird beginnend mit 0 gezählt, das heißt, das erste Tupelelement wird über den Index 0 ausgewählt. Das Ergebnistupel Selected enthält dann alle Elemente von Tuple, die zwischen den Positionen Leftindex und Rightindex stehen (inklusive der Elemente mit dem Index Leftindex und Rightindex). Sind beide Indices gleich groß, so wird ein einzelnes Element ausgewählt. Der Parameter Rightindex kann darüber hinaus auch um 1 kleiner sein, als Leftindex. In diesem Fall wird, wenn Leftindex oder Rightindex ein gültiger Index sind, ein leeres Tupel zurück geliefert. Der Index in Rightindex muss somit immer größer oder gleich Leftindex - 1 sein.

Sonderfall: Leere Eingabetupel

Falls Leftindex und Rightindex leere Tupel sind, wird ein leerer Tupel zurückgegeben. Wenn nur einer dieser beiden Parameter oder Tuple ein leeres Tupel ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_select_range, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Selected := Tuple[Leftindex:Rightindex]
```

Parameter

- ▷ **Tuple** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Leftindex** (input_control) ..... integer(-array) ⇝ *integer*
  Index des ersten auszuwählenden Elements.
- ▷ **Rightindex** (input_control) ..... integer(-array) ⇝ *integer*
  Index des letzten auszuwählenden Elements.
- ▷ **Selected** (output_control) ..... tuple(-array) ⇝ *integer* / real / string
  Ausgewählte Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_select, tuple_first_n, tuple_last_n, tuple_select_mask, tuple_str_bit_select, tuple_concat, tuple_select_rank

Siehe auch

tuple_remove

Modul

Foundation

tuple_select_rank (: : Tuple, RankIndex : Selected)

Auswählen eines Elements mit Rang n aus einem Tupel.

tuple_select_rank sortiert die Elemente aus einem Tupel Tuple in aufsteigender Reihenfolge und liefert das Element mit Rang n über den Parameter Selected zurück. Der Parameter RankIndex gibt hierbei den Index des ausgewählten Elements an. Folgerichtig darf RankIndex ausschließlich eine ganze Zahl beinhalten (falls RankIndex eine Gleitkommazahl enthält, so muss diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Tupelindizes werden beginnend mit 0 gezählt, das heißt, das niedrigste Tupelelement wird über den Index 0 ausgewählt.

Sonderfall: Leeres Eingabetupel

Falls Tuple leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_select_rank, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Selected := select_rank(Tuple, RankIndex)
```

Parameter

- ▷ **Tuple** (input_control) ..... number(-array) ⇝ *integer* / real
  Eingabetupel.
- ▷ **RankIndex** (input_control) ..... number ⇝ *integer* / real
  Rang des auszuwählenden Elements.
- ▷ **Selected** (output_control) ..... number ⇝ *integer* / real
  Ausgewähltes Tupelelement.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sort_index, tuple_sort

Siehe auch

tuple_median, tuple_select

Modul

Foundation

tuple_str_bit_select (: : Tuple, Index : Selected)

Auswählen eines einzelnen Zeichens oder Bits aus einem Tupel.

tuple_str_bit_select wählt aus einem Tupel Tuple von Strings und/oder Zahlen ein einzelnes Zeichen beziehungsweise Bit aus und gibt es in dem Ausgabetupel Selected zurück. Tuple kann hierbei aus Strings und/oder ganzen Zahlen bestehen. Welches Zeichen beziehungsweise Bit ausgewählt wird, hängt von dem Eingabeparameter Index ab, der ausschließlich aus einer Zahl „n“ bestehen darf. Ist dies eine Gleitkommazahl, so muss sie eine ganze Zahl repräsentieren (d.h., alle Nachkommastellen müssen 0 sein). Das Ausgabetupel Selected enthält nun für jedes Element aus Tuple ein neues Element, das aus dem „n-ten“ Zeichen (bei Strings) beziehungsweise dem „n-ten“ Bit (bei ganzen Zahlen) des korrespondierenden Elements von Tuple besteht.

Falls Tuple leer ist, wird eine Fehlermeldung angezeigt.

Unicode Codepoints versus Bytes

Der Index bezieht sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn der Index sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit set_system('tsp_tuple_string_operator_mode','byte') in einen Byte-Modus umgeschaltet

werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_str_bit_select`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Selected := Tuple{Index}
```

Parameter

- ▷ **`Tuple`** (input_control) . . . . . tuple(-array) ⇝ *string* / integer
  Eingabetupel.
- ▷ **`Index`** (input_control) . . . . . integer ⇝ *integer*
  Position des Zeichens oder Bits.
- ▷ **`Selected`** (output_control) . . . . . tuple(-array) ⇝ *string* / integer
  Tupel mit den ausgewählten Zeichen und Bits.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_select`, `tuple_first_n`, `tuple_last_n`, `tuple_concat`, `tuple_strchr`, `tuple_strrchr`, `tuple_str_first_n`, `tuple_str_last_n`, `tuple_substr`, `tuple_and`, `tuple_or`, `tuple_xor`, `tuple_not`

Siehe auch

`tuple_remove`

Modul

Foundation

`tuple_uniq` (: : Tuple : Uniq)

Eliminieren direkt aufeinanderfolgender Wiederholungen identischer Elemente in einem Tupel.

`tuple_uniq` eliminiert aus dem Eingabetupel `Tuple` direkt aufeinanderfolgende Wiederholungen und liefert die verbleibenden Elemente im Ausgabetupel `Uniq` zurück. Besteht `Tuple` zum Beispiel aus den Werten [0,0,1,1,1,2,0,1], so beinhaltet das Ausgabetupel `Uniq` die Werte [0,1,2,0,1]. Im Eingabetupel dürfen Strings und Zahlen auch gemischt vorkommen.

Um ein Tupel `Uniq` zu erhalten, das jeden verschiedenen Eintrag von `Tuple` genau einmal enthält, sollte vorher der Operator `tuple_sort` angewendet werden. Dabei ist zu beachten, dass das Ausgabetupel `Uniq` dann jedoch sortiert ist. Das Resultat des obigen Beispiels ist dann [0,1,2].

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_uniq`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Uniq := uniq(Tuple)
```

Wie oben bereits erwähnt kann `tuple_sort` benutzt werden, um das Tupel `Uniq` zu erhalten, das jeden verschiedenen Eintrag von `Tuple` genau einmal enthält.

```
Uniq := uniq(sort(Tuple))
```

Parameter

▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Uniq** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string
Tupel ohne aufeinanderfolgende identische Elemente.

Beispiel

```
Tuple := [0,0,1,1,1,2,0,1]
*
tuple_uniq (Tuple, Uniq)
*
tuple_sort (Uniq, Sorted)
tuple_uniq (Sorted, Uniq1)
```

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`tuple_inverse`, `tuple_sort`

Alternativen

`tuple_intersection`

Modul

Foundation

28.5 Elementreihenfolge

tuple_inverse (: : Tuple : Inverted)

Invertieren eines Tupels.

`tuple_inverse` invertiert das Eingabetupel `Tuple`. Das Ergebnistupel `Inverted` enthält also dieselben Elemente wie `Tuple` jedoch in umgekehrter Reihenfolge.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_inverse`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Inverted := inverse(Tuple)
```

Parameter

▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Inverted** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string
Inversion des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_sort`, `tuple_sort_index`

Modul

Foundation

tuple_sort (: : Tuple : Sorted)

Sortieren der Elemente eines Tupels in aufsteigender Reihenfolge.

`tuple_sort` sortiert die Elemente des Eingabetupels `Tuple` in aufsteigender Reihenfolge und liefert das Resultat in dem Ergebnistupel `Sorted` zurück. Als Voraussetzung hierfür müssen alle Elemente von `Tuple` vergleichbar sein, das heißt, entweder `Tuple` besteht gänzlich aus Strings oder es enthält ausschließlich Zahlen, wobei in letzterem Fall auch ganze Zahlen und Gleitkommazahlen vermischt auftreten dürfen.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_sort`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Sorted := sort(Tuple)
```

Parameter

- ▷ **Tuple** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Sorted** (output_control) ..... tuple(-array) ⇝ *integer* / real / string
  Sortiertes Tupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

`tuple_sort_index`, `tuple_inverse`

Modul

Foundation

tuple_sort_index (: : Tuple : Indices)

Sortieren eines Tupels und Rückgabe der Indizes des sortierten Tupels.

`tuple_sort_index` sortiert die Elemente des Eingabetupels `Tuple` in aufsteigender Reihenfolge und liefert als Resultat die Indizes der Elemente des sortierten Tupels (bezogen auf das Eingabetupel) in dem Ergebnistupel `Indices` zurück. Als Voraussetzung hierfür müssen alle Elemente von `Tuple` vergleichbar sein, das heißt, entweder `Tuple` besteht gänzlich aus Strings oder es enthält ausschließlich Zahlen, wobei in letzterem Fall auch ganze Zahlen und Gleitkommazahlen auch vermischt auftreten dürfen.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_sort_index`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Indices := sort_index(Tuple)
```

Parameter

▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Indices** (output_control) . . . . . integer(-array) ⇝ *integer*
Sortiertes Tupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sort, tuple_inverse

Modul

Foundation

28.6 Generierung

clear_handle (: : Handle :)

Gibt den Inhalt eines Handles frei.

clear_handle gibt den Inhalt des Handles Handle frei. Das Handle ist danach in einem freigegebenen Zustand und kann nicht mehr mit anderen Operatoren verwendet werden.

In der Regel werden Handles automatisch freigegeben, wenn alle Referenzen in allen Tupeln überschrieben wurden. clear_handle kann eingesetzt werden, um ein Handle zu einem definierten Zeitpunkt freizugeben, auch wenn es noch Referenzen auf dieses Handle gibt.

clear_handle gibt Handles beliebigen Typs frei und kann daher anstelle von typspezifischen Freigabeoperatoren wie clear_matrix oder clear_shape_model verwendet werden.

Parameter

▷ **Handle** (input_control) . . . . . tuple(-array) ⇝ *handle*
Das freizugebende Handle.

Ergebnis

Sind die Parameterwerte korrekt, d.h. enthält Handle ein Handle, dann liefert clear_handle den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator modifiziert den Zustand des folgenden Eingabeparameters:

- Handle

Während der Ausführung dieses Operators muss der Zugriff auf den Wert dieses Parameters synchronisiert werden, wenn er über mehrere Threads hinweg verwendet wird.

Siehe auch

tuple_is_valid_handle

Modul

Foundation

tuple_concat (: : T1, T2 : Concat)

Verknüpfen zweier Tupel zu einem neuen Tupel.

tuple_concat verknüpft die zwei Eingabetupel T1 und T2 zu einem neuen Tupel Concat, dessen vordere Elemente aus den Elementen von T1 und dessen hintere Elemente aus denen von T2 bestehen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_concat, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Concat := [T1, T2]
```

Parameter

- ▷ **T1** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Concat** (output_control) ..... tuple(-array) ⇝ *integer* / real / string
  Ergebnis der Verknüpfung der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_gen_const, tuple_str_bit_select, tuple_select, tuple_str_first_n, tuple_str_last_n, tuple_substr

Siehe auch

tuple_remove

Modul

Foundation

tuple_gen_const (: : Length, Const : Newtuple)

Erzeugen eines Tupels definierter Länge mit einer initialen Belegung.

tuple_gen_const erzeugt ein neues Tupel und gibt es in Newtuple zurück. Die Anzahl der Tupelelemente wird mit dem Parameter Length festgelegt. Length darf folglich nur aus einer einzigen Zahl bestehen. Enthält Length eine Gleitkommazahl, so darf diese nur eine ganze Zahl enthalten (alle Nachkommastellen müssen 0 sein). Der Datentyp der einzelnen Elemente des neu generierten Tupels und die initiale Belegung der Elemente wird durch den zweiten Eingabeparameter Const festgelegt. Const darf nur ein einziges Element enthalten. Alle Elemente von Newtuple entsprechen dann in ihrem Typ und ihrer Belegung exakt dem Datenelement von Const.

Sonderfall: Leere Eingabetupel

Falls eines oder mehrere Eingabetupel leer sind, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_gen_const, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Newtuple := gen_tuple_const(Length, Const)
```

Parameter

- ▷ **Length** (input_control) . . . . . number ⇝ *integer* / real
  Länge des zu erzeugenden Tupels.
- ▷ **Const** (input_control) . . . . . tuple ⇝ *integer* / real / string / handle
  Konstante für die Initialisierung der Tupelelemente.
- ▷ **Newtuple** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Neues Tupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_str_bit_select, tuple_select, tuple_str_first_n, tuple_str_last_n, tuple_concat, tuple_insert, tuple_replace, tuple_gen_sequence

Siehe auch

tuple_rand

Modul

Foundation

tuple_gen_sequence (: : Start, End, Step : Sequence)

Erzeugen eines Tupels mit einer Sequenz von äquidistanten Werten.

tuple_gen_sequence erzeugt einen neuen Tupel Sequence mit einer Sequenz von äquidistanten Werten:

[Start, Start + Step, Start + 2*Step, ... End]

Step darf nicht Null sein und das Vorzeichen von (End - Start) muss dem Vorzeichen von Step entsprechend. Der letzte Wert in Sequence kann kleiner sein als End, wenn (End - Start) nicht ganzzahlig durch Step teilbar ist.

Sonderfall: Leere Eingabetupel

Falls eines oder mehrere Eingabetupel leer sind, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_gen_sequence, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

Sequence := [Start:Step:End]

Alternative Syntax: Sequence := [Start:End], wobei Step auf *1* gesetzt wird.

Parameter

- ▷ **Start** (input_control) . . . . . number ⇝ *integer* / real
  Startwert der Sequenz.
- ▷ **End** (input_control) . . . . . number ⇝ *integer* / real
  Endwert der Sequenz.
- ▷ **Step** (input_control) . . . . . number ⇝ *integer* / real
  Inkrement.
- ▷ **Sequence** (output_control) . . . . . number(-array) ⇝ *integer* / real
  Ergebnissequenz.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_str_bit_select, tuple_select, tuple_str_first_n, tuple_str_last_n, tuple_concat, tuple_insert, tuple_replace, tuple_gen_const

Siehe auch

tuple_rand

Modul

Foundation

tuple_rand (: : Length : Rand)

Erzeugen eines Tupels von Zufallszahlen zwischen 0 und 1.

tuple_rand gibt ein Tupel von Zufallszahlen zurück, die im Intervall [0,1) liegen. Dabei bestimmt der Parameter Length die Anzahl der Elemente des Ausgabetupels, d.h. wie viele Zufallszahlen generiert werden sollen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_rand, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Rand := rand(Length)
```

Parameter

- ▷ **Length** (input_control) ..... number ⇝ *integer*
  Länge des zu erzeugenden Tupels.
- ▷ **Rand** (output_control) ..... number(-array) ⇝ *real*
  Tupel von Zufallszahlen.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

tuple_gen_const

Modul

Foundation

28.7 Konversion

handle_to_integer (: : Handle : CastedHandle)

Konvertiert ein Handle in eine Ganzzahl.

Es wird nicht empfohlen, diesen Operator in HDevelop zu verwenden. Er existiert nur aus Gründen der Rückwärtskompatibilität.

handle_to_integer konvertiert das Handle Handle in eine Integerdarstellung und gibt diese in CastedHandle zurück.

Falls der Legacy Handle Modus deaktiviert ist (default), bleibt die automatische Freigabe für dieses Handle aktiv. Sobald alle Instanzen des Handles in allen Tupeln überschrieben wurden, wird das Handle und sein Inhalt automatisch freigegeben und CastedHandle wird ungültig. Falls CastedHandle in diesem Fall an im Legacy Handle Modus laufende Codeteile übergeben werden soll, muss Handle für die Dauer dieses Aufrufs in mindestens einem Tupel erhalten bleiben, um zu vermeiden, dass es freigegeben wird.

Falls der Legacy Handle Modus aktiviert ist (siehe set_system) wird die automatische Freigabe für dieses Handle deaktiviert. Das Handle muss dann manuell mittels clear_handle oder den clear-Operator des entsprechenden Handletyps freigegeben werden.

Achtung

Es wird nicht empfohlen, diesen Operator in HDevelop zu verwenden. Er existiert nur aus Gründen der Rückwärtskompatibilität.

Parameter

- ▷ **Handle** (input_control) ..... handle(-array) ⇝ *handle*
  Das zu konvertierende Handle.
- ▷ **CastedHandle** (output_control) ..... pointer(-array) ⇝ *integer*
  Das in einen Integer umgewandelte Handle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert handle_to_integer den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

integer_to_handle

Modul

Foundation

integer_to_handle (: : IntegerHandle : Handle)

Konvertiert eine Ganzzahl in ein Handle.

Es wird nicht empfohlen, diesen Operator in HDevelop zu verwenden. Er existiert nur aus Gründen der Rückwärtskompatibilität.

integer_to_handle konvertiert den Integer IntegerHandle, der ein Handle repräsentiert, in ein Handle und gibt dieses in Handle zurück. Dies erlaubt die Konvertierung von Handles, die mit aktiviertem Legacy Handle Modus erzeugt wurden, in korrekte Handles.

Falls der Legacy Handle Modus deaktiviert ist (default), wird die automatische Freigabe für dieses Handle aktiviert. In diesem Fall wird das Handle automatisch freigegeben, wenn alle Referenzen darauf in allen Handles überschrieben wurden. Falls IntegerHandle in diesem Fall weiterhin verwendet werden soll, muss eine Referenz auf das zurückgegebene Handle behalten werden, um die automatische Freigabe zu unterbinden.

Falls der Legacy Handle Modus aktiviert ist, wird das zurückgegebene Handle wieder vom Typ Integer sein. Weiterhin muss das Handle in diesem Fall mit clear_handle oder dem clear-Operator des entsprechenden Handletyps freigegeben werden.

Achtung

Es wird nicht empfohlen, diesen Operator in HDevelop zu verwenden. Er existiert nur aus Gründen der Rückwärtskompatibilität.

Parameter

- ▷ **IntegerHandle** (input_control) ..... pointer(-array) ⇝ *integer*
  Das zu konvertierende Handle in Integer-Darstellung.
- ▷ **Handle** (output_control) ..... handle(-array) ⇝ *handle*
  Das Handle vom korrekten Handle-Typ.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert integer_to_handle den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Dieser Operator liefert ein Handle zurück. Es ist zu beachten, dass der Zustand einer Instanz dieses Handletyps durch bestimmte Operatoren geändert werden kann, obwohl das Handle als Eingabeparameter in diesen Operatoren verwendet wird.

Siehe auch

`handle_to_integer`

Modul

Foundation

tuple_chr (: : T : Chr)

Konvertieren eines Tupels von ganzen Zahlen in ein Tupel von Strings.

`tuple_chr` konvertiert das Eingabetupel `T`, bestehend aus ganzen Zahlen, in ein Tupel aus Strings der Länge 1. Wenn die Stringkodierung der HALCON Bibliothek UTF-8 ist (siehe `set_system('filename_encoding','utf8')`) und die String-Operatoren auf der Basis von Codepoints arbeiten (siehe `set_system('tuple_string_operator_mode','codepoint')`), was für beide Optionen die Standardeinstellung ist, akzeptiert der Operator beliebige Unicode-Zeichencodes und gibt Strings mit der entsprechenden UTF-8 Repräsentation der Unicode-Zeichen zurück. Wenn die Stringkodierung der HALCON Bibliothek *'locale'* ist oder die String-Operatoren im Modus *'byte'* arbeiten, akzeptiert der Operator nur ganze Zahlen zwischen 0 und 256. In diesem Fall gibt der Operator `tuple_chr` Strings mit jeweils einer Länge von einem Byte zurück, wobei die Eingabezahl ein ANSI-Code ist. Detailliertere Informationen zu den verschiedenen Modi und weitere Aspekte zur Stringkodierung sind unter Tupel / String-Operationen zu finden.

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_chr`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Chr := chr(T)
```

Parameter

- ▷ **T** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel mit Unicode-Zeichencodes oder ANSI-Codes.
  Restriktion: 0 <= T
- ▷ **Chr** (output_control) ..... string(-array) ⇝ *string*
  Ausgabetupel mit Strings, die aus den Zeichencodes im Eingabetupel generiert wurden.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_chrt`

Siehe auch

`tuple_ord`, `tuple_ords`

Modul

Foundation

tuple_chrt (: : T : Chrt)

Konvertieren eines Tupels von ganzen Zahlen in ein Tupel von Strings.

tuple_chrt konvertiert das Eingabetupel T, bestehend aus ganzen Zahlen, in ein Tupel aus Strings und Einträgen mit der Zahl 0. Alle anderen Zahlen im Eingabetupel werden in Zeichen konvertiert, die bis zur nächsten 0 im Eingabetupel jeweils zu einem String zusammengefasst werden. Wenn die Stringkodierung der HALCON Bibliothek UTF-8 ist (siehe set_system ('filename_encoding','utf8')) und die String-Operatoren auf der Basis von Codepoints arbeiten (siehe set_system('tuple_string_operator_mode','codepoint')), was für beide Optionen die Standardeinstellung ist, interpretiert der Operator die Eingabezahlen als Unicode-Zeichencodes und wandelt diese in die entsprechende UTF-8 Darstellung der jeweiligen Zeichen um. Wenn die Stringkodierung der HALCON Bibliothek *'locale'* ist oder die String-Operatoren im Modus *'byte'* arbeiten, akzeptiert der Operator nur ganze Zahlen zwischen 0 und 256. Diese werden als ANSI-Code interpretiert und Byte für Byte zu einem String zusammengesetzt. In diesem Modus wird nicht überprüft, ob die Bytefolgen in den Ergebnisstrings gültige Zeichen entsprechend der eingestellten Stringkodierung ergeben. Detailliertere Informationen zu den verschiedenen Modi und weitere Aspekte zur Stringkodierung sind unter Tupel / String-Operationen zu finden.

Der Operator versucht so viele Eingabezahlen wie möglich in einem String zusammenzufassen. Ist das nächste Element im Eingabetupel eine 0, wird der aktuelle String abgeschlossen und die Zahl 0 zum Ausgabetupel hinzugefügt. Folgen im Eingabetupel weitere Zahlen ungleich 0, wird ein neuer String angefangen. Der Operator tuple_chrt kann benutzt werden, um mit read_serial eingelesene Daten in Strings umzuwandeln. Dieses Vorgehen erlaubt auch das Lesen von Bytes mit dem Wert 0.

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_chrt, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Chrt := chrt(T)
```

Parameter

▷ **T** (input_control) ..... integer(-array) ⇝ *integer*
Eingabetupel mit ganzen Zahlen.
Restriktion: 0 <= T

▷ **Chrt** (output_control) ..... string(-array) ⇝ *string* / integer
Ausgabetupel mit Strings, welche durch die Zahl 0 voneinander getrennt sind.

Beispiel

```
read_serial (SerialHandle, 100, Data)
tuple_chrt (Data, Strings)
```

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_chr

Siehe auch

tuple_ord, tuple_ords, read_serial

Modul

Foundation

tuple_int (: : T : Int)

Konvertieren eines Tupels in ein Tupel von ganzzahligen Werten.

tuple_int konvertiert das Eingabetupel T in ein Tupel von von ganzzahligen Werten. Dabei wird T abgeschnitten. Die Konversion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_int, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Int := int(T)
```

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Int** (output_control) . . . . . number(-array) ⇝ *integer*
  Ergebnis der Konversion in ganze Zahlen.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_round

Modul

Foundation

tuple_is_number (: : T : IsNumber)

Testen, ob Elemente eines Tupels (inkl. Strings) Zahlen darstellen.

tuple_is_number testet das Eingabetupel T daraufhin, welches Element eine Zahl darstellt. Dabei wird für schon vorhandene Zahlen 1 zurückgeliefert. Für Strings wird getestet, ob der String eine Zahl darstellt oder nicht. Für Strings, die Zahlen darstellen, wird eine 1 zurückgeliefert, ansonsten eine 0

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_is_number, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsNumber := is_number(T)
```

Parameter

- ▷ **T** (input_control) . . . . . tuple(-array) ⇝ *real* / integer / string
  Eingabetupel.
- ▷ **IsNumber** (output_control) . . . . . integer(-array) ⇝ *integer*
  Tupel mit Booleschen Werten.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_is_int, tuple_is_real, tuple_type

Siehe auch

tuple_number

Modul

Foundation

tuple_number (: : T : Number)

Konvertieren eines Tupels (von Strings) in ein Tupel von Zahlen.

tuple_number konvertiert das Eingabetupel T in ein Tupel von Zahlen. Dabei werden schon vorhandene Zahlen im Eingabetupel kopiert. Strings werden in den passenden Zahlentyp (Gleitpunktzahlen oder ganze Zahlen) konvertiert oder als Strings kopiert, falls sie keine Zahl darstellen. Strings, die mit 0x bzw. 0 beginnen, werden als Hexadezimalzahlen bzw. Oktalzahlen interpretiert. Der String '20' z.B. wird in die Zahl 20 konvertiert, '020' in 16, '0x20' in 32.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_number, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Number := number(T)
```

Parameter

- ▷ **T** (input_control) ..... tuple(-array) ⇝ *string* / real / integer
  Eingabetupel.
- ▷ **Number** (output_control) ..... tuple(-array) ⇝ *real* / integer / string
  Eingabetupel als Zahlen.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

tuple_is_number, tuple_string

Modul

Foundation

tuple_ord (: : T : Ord)

Konvertieren eines Tupels von Strings der Länge 1 in ein Tupel von ganzen Zahlen.

tuple_ord konvertiert das Eingabetupel T, das nur Strings der Länge 1 enthalten darf, in ein Tupel von Ganzzahlen. Wenn die Stringkodierung der HALCON Bibliothek UTF-8 ist (siehe set_system ('filename_encoding','utf8')) und die String-Operatoren auf der Basis von Codepoints arbeiten (siehe set_system('tuple_string_operator_mode','codepoint')), was für beide Optionen die Standardeinstellung ist, akzeptiert der Operator beliebige Unicode-Zeichen (Codepoint) als UTF-8-String und gibt den entsprechenden Unicode-Zeichencode zurück. Wenn die Stringkodierung der HALCON Bibliothek *'locale'* ist oder die String-Operatoren im Modus *'byte'* arbeiten, akzeptiert der Operator nur ein einzelnes Byte pro Eingabestring. In diesem Fall gibt der Operator tuple_ord den ANSI-Code des Eingabe-Bytes als ganze Zahl zwischen 0 und 256 zurück. Detailliertere Informationen zu den verschiedenen Modi und weitere Aspekte zur Stringkodierung sind unter Tupel / String-Operationen zu finden.

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_ord`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Ord := ord(T)
```

Parameter

- ▷ **T** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel mit Strings der Länge 1.
- ▷ **Ord** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ausgabetupel mit den Unicode-Zeichencodes oder den ANSI-Codes der Zeichen des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_ords`

Siehe auch

`tuple_chr`, `tuple_chrt`

Modul

Foundation

tuple_ords (: : T : Ords)

Konvertieren eines Tupels von Strings in ein Tupel von ganzen Zahlen.

`tuple_ords` konvertiert das Eingabetupel `T`, das nur Strings und ganze Zahlen enthalten darf, in ein Tupel von Ganzzahlen. Wenn die Stringkodierung der HALCON Bibliothek UTF-8 ist (siehe `set_system('filename_encoding','utf8')`) und die String-Operatoren auf der Basis von Codepoints arbeiten (siehe `set_system('tuple_string_operator_mode','codepoint')`), was für beide Optionen die Standardeinstellung ist, gibt der Operator für die Eingabestrings die entsprechenden Unicode-Zeichencodes zurück. Wenn die Stringkodierung der HALCON Bibliothek *'locale'* ist oder die String-Operatoren im Modus *'byte'* arbeiten, gibt der Operator den ANSI-Code für jeden Byte des Eingabestrings zurück. In diesem Modus kann das Ergebnis von der aktuell verwenden Codepage für Strings mit nicht-ASCII Zeichen abhängen. Detailliertere Informationen zu den verschiedenen Modi und weitere Aspekte zur Stringkodierung sind unter Tupel / String-Operationen zu finden.

Entsprechend ihrer Reihenfolge innerhalb des Strings und des Tupels, werden die Zeichencodes der einzelnen Strings in das Ausgabetupel geschrieben. Ganzzahlen werden entsprechend ihrer Position im Eingabetupel in das Ausgabetupel kopiert. Der Operator `tuple_ords` kann genutzt werden, um Ausgaben mit `write_serial` vorzubereiten. Insbesondere kann zwischen zwei Strings der Wert 0 als Trenner geschrieben werden, indem die Ganzzahl 0 in das Eingabetupel `T` eingefügt wird.

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_ords`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Ords := ords(T)
```

Parameter

- ▷ **T** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Eingabetupel mit Strings.
- ▷ **Ords** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ausgabetupel mit den Unicode-Zeichencodes oder ANSI-Codes des Eingabestrings.

Beispiel

```
tuple_ords (['String 1', 0, 'String 2', 0], Data)
write_serial (SerialHandle, Data)
```

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_ord

Siehe auch

tuple_chr, tuple_chrt, write_serial

Modul

Foundation

tuple_real (: : T : Real)

Konvertieren eines Tupels in ein Tupel von Gleitkommazahlen.

tuple_real konvertiert das Eingabetupel T in ein Tupel von Gleitkommazahlen. Die Konversion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_real, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Real := real(T)
```

Parameter

▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
Eingabetupel.

▷ **Real** (output_control) ..... number(-array) ⇝ *real*
Eingabetupel als Gleitkommazahlen.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

tuple_round (: : T : Round)

Konvertieren eines Tupels in ein Tupel von ganzen Zahlen.

tuple_round konvertiert das Eingabetupel T in ein Tupel von ganzen Zahlen. Dabei wird T zur nächsten ganzen Zahl gerundet. Die Konversion von Strings ist nicht erlaubt.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_round`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Round := round(T)
```

Parameter

- ▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Round** (output_control) ..... number(-array) ⇝ *integer*
  Ergebnis der Rundung.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_int`

Modul

Foundation

tuple_string (: : T, Format : String)

Konvertieren eines Tupels in ein Tupel von Strings.

`tuple_string` konvertiert Zahlen in Strings und modifiziert Strings. Der Operator hat zwei Parameter: `T` enthält die Zahlen oder Strings, die konvertiert werden sollen. `Format` spezifiziert die Konversion.

Im folgenden werden zuerst einige Beispiele für die Benutzung von `tuple_string` gegeben und anschließend die Struktur des `Format`-Strings erläutert.

Beispiele

Beispiele für die Konvertierung von Zahlen in Strings:

`T` (Input)	`Format` (Input)	`String` (Output)
23	'10.2f'	' 23.00'
23	'-10.2f'	'23.00 '
4	'.7f'	'4.0000000'
1234.56789	'+10.3f'	' +1234.568'
255	'x'	'ff'
255	'X'	'FF'
0xff	'.5d'	'00255'

Beispiele für die Modifikation von Strings:

`T` (Input)	`Format` (Input)	`String` (Output)
'total'	'10s'	' total'
'total'	'-10s'	'total '
'total'	'-10.3s'	'tot '

Format-String

Der `Format`-String besteht aus den folgenden vier Teilen:

`<flags><field width><precision><conversion characters>`

flags Null oder mehr Flags, in beliebiger Reihenfolge, die die Bedeutung der Konversionsspezifikation modifizieren. Die Flags können aus den folgenden Zeichen bestehen:

- **-** Das Ergebnis der Konversion wird links im Ergebnisfeld ausgerichtet.
- **+** Das Ergebnis einer vorzeichenbehafteten Konversion fängt immer mit dem Vorzeichen an, also + oder −.
- <**space**> Falls das erste Zeichen einer vorzeichenbehafteten Konversion kein Vorzeichen ist, wird ein Leerzeichen vor das Ergebnis eingefügt. Falls das <space> Flag und das + Flag gleichzeitig angegeben werden, wird das <space> Flag ignoriert.
- **#** Der Wert soll in eine „alternativen Form“ konvertiert werden. Für die `d` und `s` Konversionen hat dieses Flag keine Auswirkungen. Bei der `o` Konversion (siehe unten) erhöht es die Präzision, so dass das erste Zeichen immer als 0 ausgegeben wird. Bei den `x` oder `X` Konversionen (siehe unten) wird an Ergebnisse ungleich Null immer `0x` oder `0X` am Anfang eingefügt. Bei den `e`, `E`, `f`, `g`, und `G` Konversionen enthält das Ergebnis immer den Dezimalpunkt, selbst wenn auf den Dezimalpunkt keine Ziffern folgen. Bei den `g`, und `G` Konversionen werden außerdem, im Gegensatz zum normalen Verhalten, abschließende Nullen nicht entfernt.
- **0** Der Wert soll mit Nullen aufgefüllt werden. Für `d`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `F`, `g`, und `G`-Konvertierungen wird der konvertierten Wert auf der linken Seite mit Nullen statt mit Leerzeichen aufgefüllt. Wenn die `0` und − Flags erscheinen, wird das `0` Flag ignoriert. Wenn bei einer numerischen Umrechnung eine Genauigkeit gegeben ist (`d`, `o`, `u`, `x`, und `X`), wird das `0` Flag ignoriert. Bei anderen Konvertierungen ist das Verhalten undefiniert.

field width Eine optionale Zeichenfolge von Dezimalziffern, die eine minimale Breite des Ergebnisfeldes spezifiziert. Falls der konvertierte Wert weniger Zeichen als die Feldbreite hat, wird er links (oder rechts, falls das Linksausrichtungsflag - spezifiziert worden ist) mit Leerzeichen bis zur Feldbreite aufgefüllt.

precision Die Präzision spezifiziert die minimale Anzahl von Ziffern, die für die `d`, `o`, `x`, oder `X` Konversionen ausgegeben werden soll (das Ergebnisfeld wird mit führenden Nullen aufgefüllt). Für die `e` und `f` Konversionen spezifiziert sie die Anzahl der Ziffern, die hinter dem Dezimalpunkt erscheinen sollen, für die `g` Konversion die maximale Anzahl von signifikanten Stellen, und für die `s` Konversion die maximale Anzahl von Zeichen, die von einem String ausgegeben werden sollen. Die Präzision hat die Form **.** gefolgt von eine Folge von Dezimalzahlen. Eine leere Folge wird als Null interpretiert.

conversion characters Ein Konversionszeichen spezifiziert die Art der Konversion, die auszuführen ist:

- **d,o,x,X** Das ganzzahlige Argument wird als vorzeichenbehaftete Dezimalzahl (`d`), vorzeichenlose Oktalzahl (`o`) oder vorzeichenlose Hexadezimalzahl (`x` und `X`) ausgegeben. Die `x` Konversion verwendet die Zeichen und Buchstaben `0123456789abcdef`, während die `X` Konversion `0123456789ABCDEF` verwendet. Die Präzision spezifiziert hier die minimale Anzahl von Ziffern, die ausgegeben werden sollen. Wenn die zu konvertierende Zahl mit weniger Ziffern repräsentiert werden kann als das angegebene Minimum, werden führende Nullen eingefügt. Wenn nichts anderes angegeben wird, ist die Präzision 1. Wenn die Zahl 0 mit einer Präzision von 0 ausgegeben werden soll, wird ein leerer String erzeugt.
- **f** Die Gleitkommazahl im Argument wird in dezimaler Notation in folgender Art ausgegeben: `[-]ddd.ddd`. Hierbei ist die Anzahl Ziffern hinter dem Dezimalpunkt gleich der Präzision. Wenn die Präzision nicht angegeben wird, werden sechs Ziffern ausgegeben; wenn die Präzision explizit auf 0 gesetzt wird, werden kein Dezimalpunkt und keine Nachkommastellen ausgegeben.
- **e,E** Die Gleitkommazahl im Argument wird in dezimaler Notation in folgender Art ausgegeben: `[-]d.ddde±dd`, wobei genau eine Zahl vor dem Dezimalpunkt steht und die Anzahl von Ziffern hinter dem Dezimalpunkt gleich der Präzision ist. Wenn die Präzision nicht angegeben wird, werden sechs Ziffern ausgegeben; wenn die Präzision explizit auf 0 gesetzt wird, werden kein Dezimalpunkt und keine Nachkommastellen ausgegeben. In der `E` Konversion wird der Exponent in der Ausgabe durch `E` anstelle von `e` gekennzeichnet. Der Exponent enthält immer mindestens zwei Ziffern. Falls die auszugebende Zahl mehr als zwei Ziffern im Exponenten benötigt, werden weitere Ziffern ausgegeben.
- **g,G** Die Gleitkommazahl im Argument wird in der Art der `f` oder `e` Konversionen (oder in der Art der `E` Konversion, falls die `G` Konversion spezifiziert worden ist) ausgegeben, wobei die Präzision die Anzahl der signifikanten Stellen angibt. Die Art der Konversion hängt von der zu konvertierenden Zahl ab; die `e` Konversion wird nur dann verwendet, wenn der Exponent des Ergebnisses der Konversion kleiner als -4 oder größer gleich der Präzision ist. Abschließende Nullen werden aus dem Ergebnis entfernt. Ein Dezimalpunkt wird nur ausgegeben, falls er von einer Ziffer gefolgt wird.
- **s** Von dem String-Argument werden solange Zeichen ausgegeben, bis das Ende des Strings oder die in der Präzision angegebene Anzahl von Zeichen erreicht ist. Wenn die Präzision nicht angegeben wird, wird sie als unendlich groß interpretiert und alle Zeichen bis zum Ende des Strings werden ausgegeben.

b Ähnlich wie die `s` Konversion, nur dass der String sog. Backslash-Escape-Sequenzen (Zeichenfolgen, die mit
beginnen) enthalten kann, die in die Zeichen, die sie repräsentieren umgewandelt werden.

In keinem Fall führt eine nicht existierende oder ungenügende Feldbreite zur Beschneidung des Feldes; wenn das Ergebnis der Konversion breiter als die Feldbreite ist, wird das Feld verbreitert, so dass das Ergebnis hineinpasst.

Sonderfall: Leere Eingabetupel

Falls eines oder mehrere Eingabetupel leer sind, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_string`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
String := T $ Format
```

Parameter

- ▷ **T** (input_control) ..... tuple(-array) ⇝ *real* / integer / string
  Eingabetupel.
- ▷ **Format** (input_control) ..... string ⇝ *string*
  Format-String.
- ▷ **String** (output_control) ..... string(-array) ⇝ *string*
  In Strings konvertiertes Eingabetupel.

Beispiel

```
*
* '     23.00'
tuple_string (23, '10.2f', String)
String := 23$'10.2f'
*
* '23.00     '
tuple_string (23, '-10.2f', String)
String := 23$'-10.2f'
*
* '4.0000000'
tuple_string (4, '.7f', String)
String := 4$'.7f'
*
* ' +1234.568'
tuple_string (1234.56789, '+10.3f', String)
String := 1234.56789$'+10.3f'
*
* 'ff'
tuple_string (255, 'x', String)
String := 255$'x'
*
* 'FF'
tuple_string (255, 'X', String)
String := 255$'X'
*
* '00255'
tuple_string (0xff, '.5d', String)
String := 0xff$'.5d'
*
* '     total'
tuple_string ('total', '10s', String)
String := 'total'$'10s'
*
* 'total     '
```

```
tuple_string ('total', '-10s', String)
String := 'total'$'-10s'
*
* 'tot       '
tuple_string ('total', '-10.3s', String)
String := 'total'$'-10.3s'
```

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_sub

Modul

Foundation

28.8 Logische Operationen

tuple_and (: : T1, T2 : And)

Berechnen des logischen Und zweier Tupel.

tuple_and berechnet das logische Und der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leere Eingabetupel

Falls eines oder mehrere Eingabetupel leer sind, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_and, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
And := T1 and T2
```

Parameter

- ▷ **T1** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 2.
- ▷ **And** (output_control) ..... integer(-array) ⇝ *integer*
  Logisches Und der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_or, tuple_xor, tuple_not

Siehe auch

tuple_band, tuple_bor, tuple_bxor, tuple_bnot

Modul

Foundation

tuple_not (: : T : Not)

Berechnen des logischen Nicht eines Tupels.

tuple_not berechnet das logische Nicht des Eingabetupels T. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_not, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Not := not T1
```

Parameter

- ▷ **T** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel.
- ▷ **Not** (output_control) ..... integer(-array) ⇝ *integer*
  Binäres Nicht des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_and, tuple_or, tuple_xor

Siehe auch

tuple_band, tuple_bor, tuple_bxor, tuple_bnot

Modul

Foundation

tuple_or (: : T1, T2 : Or)

Berechnen des logischen Oder zweier Tupel.

tuple_or berechnet das logische Oder der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leere Eingabetupel

Falls eines oder mehrere Eingabetupel leer sind, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_or, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Or := T1 or T2
```

Parameter

- ▷ **T1** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 2.
- ▷ **Or** (output_control) ..... integer(-array) ⇝ *integer*
  Logisches Oder der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_and, tuple_xor, tuple_not

Siehe auch

tuple_band, tuple_bor, tuple_bxor, tuple_bnot

Modul

Foundation

tuple_xor (: : T1, T2 : Xor)

Berechnen des logischen Exklusiv-Oder zweier Tupel.

tuple_xor berechnet das logische Exklusiv-Oder der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Sonderfall: Leere Eingabetupel

Falls eines oder mehrere Eingabetupel leer sind, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_xor, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Xor := T1 xor T2
```

Parameter

▷ **T1** (input_control) . . . integer(-array) ⇝ *integer*
Eingabetupel 1.

▷ **T2** (input_control) . . . integer(-array) ⇝ *integer*
Eingabetupel 2.

▷ **Xor** (output_control) . . . integer(-array) ⇝ *integer*
Binäres Exklusiv-Oder der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_and, tuple_or, tuple_not

Siehe auch

tuple_band, tuple_bor, tuple_bxor, tuple_bnot

Modul

Foundation

28.9 Manipulation

tuple_insert (: : Tuple, Index, InsertTuple : Extended)

Einfügen von Elementen in ein Tupel ab Index.

tuple_insert fügt einzelne Elemente aus dem Tupel InsertTuple ab Index in das Tupel Tuple ein und gibt das veränderte Tupel mit dem Parameter Extended zurück. Der Parameter Index enthält hierbei den Startindex der einzufügenden Elemente und InsertTuple die Elementwerte, die eingefügt werden. Alle nachfolgenden Tupelelemente verschieben sich nach hinten. Folgerichtig darf Index ausschließlich eine ganze Zahl beinhalten (falls Index eine Gleitkommazahl enthält, so muss diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Tupelindizes werden beginnend mit 0 gezählt, das heißt das erste Tupelelement wird über den Index 0 ausgewählt. Falls der Indexwert der Tupellänge von Tuple entspricht, so wird InsertTuple hinten angefügt. Die Länge des Ausgabetupels Extended entspricht somit immer der Summe der Länge der beiden Eingabetupel. Besteht Tuple zum Beispiel aus den Werten [0,0,0,1,1,1] und enthalten Index und InsertTuple die Werte [3] und [2,2,2], so beinhaltet das Ausgabetupel Extended die Ergebniswerte [0,0,0,2,2,2,1,1,1]. Die Eingabetupel Tuple und InsertTuple dürfen gleichzeitig Strings und Zahlen enthalten.

Sonderfall: Leere Eingabetupel

Falls eines oder mehrere Eingabetupel leer sind, wird eine Fehlermeldung angezeigt.

Parameter

- ▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Index** (input_control) . . . . . integer ⇝ *integer*
  Startindex.
- ▷ **InsertTuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Tupel mit den einzufügenden Elementen.
- ▷ **Extended** (output_control) . . . . . tuple-array ⇝ *integer* / real / string
  Tupel mit den eingefügten Elementen.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_concat, tuple_replace, tuple_gen_const

Siehe auch

tuple_remove

Modul

Foundation

tuple_remove (: : Tuple, Index : Reduced)

Eliminieren einzelner Elemente aus einem Tupel.

tuple_remove eliminiert einzelne Elemente aus einem Tupel Tuple und liefert die verbleibenden Elemente über den Parameter Reduced zurück. Der Parameter Index enthält hierbei die Indizes der Elemente, die eliminiert werden sollen. Folgerichtig darf Index ausschließlich ganze Zahlen beinhalten (falls Index eine Gleitkommazahl enthält, so muss diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Tupelindizes werden beginnend mit 0 gezählt, das heißt das erste Tupelelement wird über den Index 0 ausgewählt. Doppelte Indizes oder Indizes, die außerhalb des gültigen Bereichs liegen, werden ignoriert.

Sonderfall: Leeres Eingabetupel

Falls `Tuple` leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_remove`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Reduced := remove(Tuple, Index)
```

Parameter

- ▷ **`Tuple`** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **`Index`** (input_control) ..... integer(-array) ⇝ *integer*
  Indizes der zu löschenden Elemente.
- ▷ **`Reduced`** (output_control) ..... tuple(-array) ⇝ *integer* / real / string
  Verbleibende Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_first_n`, `tuple_last_n`, `tuple_str_bit_select`, `tuple_concat`, `tuple_insert`, `tuple_replace`

Siehe auch

`tuple_select`, `tuple_select_mask`

Modul

Foundation

`tuple_replace` (: : Tuple, Index, ReplaceTuple : Replaced)

Ersetzen einzelner Elemente aus einem Tupel.

`tuple_replace` ersetzt einzelne Elemente aus einem Tupel `Tuple` und gibt das veränderte Tupel mit dem Parameter `Replaced` zurück. Der Parameter `Index` enthält hierbei die Indizes der Elemente und `ReplaceTuple` die entsprechenden Elementwerte, die ersetzt werden. Folglich darf `Index` ausschließlich ganze Zahlen beinhalten (falls `Index` eine Gleitkommazahl enthält, so muss diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Tupelindizes werden beginnend mit 0 gezählt, das heißt das erste Tupelelement wird über den Index 0 ausgewählt. Falls `ReplaceTuple` nur einen Wert enthält, dann wird dieser an allen in `Index` angegebenen Indizes eingesetzt. Falls ein Indexwert größer als die bisherige Tupellänge ist, so wird das Ergebnistupel `Replaced` entsprechend verlängert und mit Nullen aufgefüllt. Falls zum Beispiel `Tuple` [1] enthält, `Index` die Werte [2,4] enthält und `ReplaceTuple` die Werte [3,5] enthält, so ergibt sich `Replaced` gleich [1,0,3,0,5].

Die Eingabetupel `Tuple` und `ReplaceTuple` dürfen gleichzeitig Strings und Zahlen enthalten.

Sonderfall: Leere Eingabetupel

Ist eines der beiden Tupel `Index` und `ReplaceTuple` leer und das andere nicht, wird eine Fehlermeldung angezeigt. Sind beide leer, entspricht das Ausgabetupel `Replaced` dem Eingabetupel `Tuple`. Sind beide nicht leer, aber das Eingabetupel `Tuple` ist leer, wird das leere Tupel entsprechend der oben beschriebenen Vorgehensweise erweitert.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_replace`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Replaced := replace(Tuple, Index, ReplaceTuple)
```

Parameter

- ▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Index** (input_control) . . . . . integer(-array) ⇝ *integer*
  Indizes der zu ersetzenden Elemente.
- ▷ **ReplaceTuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Tupel mit den einzusetzenden Elementen.
- ▷ **Replaced** (output_control) . . . . . tuple-array ⇝ *integer* / real / string
  Tupel mit den ersetzten Elementen.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_select`, `tuple_first_n`, `tuple_last_n`, `tuple_select_mask`, `tuple_str_bit_select`, `tuple_concat`, `tuple_select_rank`

Siehe auch

`tuple_remove`, `tuple_insert`

Modul

Foundation

28.10 Mengen-Operationen

tuple_difference (: : Set1, Set2 : Difference)

Berechnet die Differenzmenge von zwei Tupeln.

`tuple_difference` liefert die Differenz von `Set1` und `Set2` in `Difference` zurück. Die Differenz von [0,1,3,3,5] und [2,3,5,10] ist zum Beispiel [0,1]. Elemente unterschiedlichen Typs werden als unterschiedliche Elemente betrachtet, das heißt, 1.0 und 1 sind nicht gleich. Wenn `Set1` und `Set2` unterschiedliche Elementtypen enthalten, kann die Differenz gleich `Set1` sein. Die Differenz von [2.0,5.0,0,10] und [2,5,3] ist zum Beispiel [2.0,5.0,0,10].

Sonderfall: Leere Eingabetupel

Falls beide Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück. Gleichermaßen gibt der Operator ein leeres Tupel zurück, wenn `Set1` leer ist und `Set2` nicht. Falls jedoch `Set2` leer ist und `Set1` nicht, entspricht die Ausgabe `Set1`.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_difference`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Difference := difference(Set1, Set2)
```

Parameter

- ▷ **Set1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
  Eingabe-Tupel.
- ▷ **Set2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
  Eingabe-Tupel.
- ▷ **Difference** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
  Die Differenz aus den zwei Eingabe-Tupeln.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_difference` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

tuple_symmdiff

Siehe auch

tuple_intersection, tuple_symmdiff, tuple_union

Modul

Foundation

tuple_intersection (: : Set1, Set2 : Intersection)

Berechnet die Schnittmenge von zwei Tupeln.

tuple_intersection liefert die Schnittmenge von Set1 und Set2 in Intersection zurück. Der Schnitt von [0,1,3,3,5] und [2,3,5,10] ist zum Beispiel [3,5]. Elemente unterschiedlichen Typs werden als unterschiedliche Elemente betrachtet, das heißt, 1.0 und 1 sind nicht gleich. Wenn Set1 und Set2 unterschiedliche Elementtypen enthalten, kann ihre Schnittmenge leer sein. So ist zum Beispiel der Schnitt von [2.0,5.0,0,10] und [2,5,3] leer.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_intersection, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Intersection := intersection(Set1, Set2)
```

Parameter

- ▷ **Set1** (input_control) ..... tuple(-array) ⇝ *integer* / real / string / handle
  Eingabe-Tupel.
- ▷ **Set2** (input_control) ..... tuple(-array) ⇝ *integer* / real / string / handle
  Eingabe-Tupel.
- ▷ **Intersection** (output_control) ..... tuple(-array) ⇝ *integer* / real / string / handle
  Die Schnittmenge aus den zwei Eingabe-Tupeln.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert tuple_intersection den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

tuple_union

Siehe auch

tuple_difference, tuple_symmdiff, tuple_union

Modul

Foundation

tuple_symmdiff (: : Set1, Set2 : SymmDiff)

Berechnet die symmetrische Differenz von zwei Tupeln.

tuple_symmdiff liefert die symmetrische Differenz von Set1 und Set2 in SymmDiff zurück. Ist zum Beispiel Set1 gleich [0,1,3,3,5] und Set2 gleich [2,3,5,10], dann ist SymmDiff [0,1,2,10]. Elemente unterschiedlichen Typs werden als unterschiedliche Elemente betrachtet, das heißt, 1.0 und 1 sind nicht gleich. Wenn Set1 und Set2 unterschiedliche Elementtypen enthalten, kann die symmetrische Differenz die Vereinigungsmenge sein. Die symmetrische Differenz von [2,5,3] und [2.0,5.0] ist so zum Beispiel [2,5,3,2.0,5.0].

Sonderfall: Leere Eingabetupel

Falls beide Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück. Falls ein Eingabetupel leer ist und das andere nicht, dann entspricht die Ausgabe dem Tupel, dass nicht leer ist.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_symmdiff, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
SymmDiff := symmdiff(Set1, Set2)
```

Parameter

- ▷ **Set1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
  Eingabe-Tupel.
- ▷ **Set2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
  Eingabe-Tupel.
- ▷ **SymmDiff** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
  Die symmetrische Differenz aus den zwei Eingabe-Tupeln.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert tuple_symmdiff den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

tuple_difference

Siehe auch

tuple_difference, tuple_intersection, tuple_union

Modul

Foundation

tuple_union (: : Set1, Set2 : Union)

Berechnet die Vereinigungsmenge von zwei Tupeln.

tuple_union liefert die Vereinigungsmenge von Set1 und Set2 in Union zurück. Die Vereinigung von [0,1,3,3,5] und [3,4] ist zum Beispiel [0,1,3,4,5]. Elemente unterschiedlichen Typs werden als unterschiedliche Elemente betrachtet, das heißt, 1.0 und 1 sind nicht gleich. Die Vereinigung von [2,5,3] und [4.0,5.0] ist zum Beispiel [2,5,3,4.0,5.0].

Sonderfall: Leere Eingabetupel

Falls beide Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück. Falls ein Eingabetupel leer ist und das andere nicht, dann entspricht die Ausgabe dem Tupel, dass nicht leer ist.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_union, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Union := union(Set1, Set2)
```

Parameter

▷ **Set1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
Eingabe-Tupel.

▷ **Set2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
Eingabe-Tupel.

▷ **Union** (output_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
Die Vereinigungsmenge aus den zwei Eingabe-Tupeln.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_union` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf interner Datenebene.

Alternativen

`tuple_intersection`

Siehe auch

`tuple_difference`, `tuple_intersection`, `tuple_symmdiff`

Modul

Foundation

28.11 Merkmale

get_handle_object (: Object : Handle, Key :)

Zurückgeben eines über einen Schlüssel referenzierten ikonischen Objekts eines Handles

`get_handle_object` liefert das ikonische Objekt, das von dem Schlüssel `Key` in `Handle` referenziert wird, in `Object` zurück. Die Liste der von einem Handle unterstützten Schlüssel und ihrer Typen kann zuvor mit `get_handle_param` abgerufen werden.

Zu beachten ist, dass `get_handle_object` nur für die Implementierung generischer Inspektions- und Debugmechanismen gedacht ist. Die für ein Handle zurückgegebenen Schlüssel können sich auch ohne weitere Ankündigung ändern. Weiterhin werden manche Handletypen und Handleeigenschaften von diesem Mechanismus eventuell nicht unterstützt. Dieser Mechanismus ist nicht auf Performanz optimiert. Um die Eigenschaften von Handles zuverlässig abzufragen, etwa zum Steuern des weiteren Programmablaufs, wird empfohlen, die jeweils auf einen Handletyp spezialisierten Operatoren wie `get_object_model_3d_params` oder `get_shape_model_params` zu verwenden.

Parameter

▷ **Object** (output_object) . . . . . object(-array) ⇝ *object*
Vom Schlüssel referenziertes ikonisches Objekt.

▷ **Handle** (input_control) . . . . . handle ⇝ *handle*
Handle, dessen Eigenschaft abgerufen werden soll.

▷ **Key** (input_control) . . . . . string ⇝ *string* / integer
Abzurufender Schlüssel.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_handle_object` den Wert TRUE. Falls das Handle ungültig ist, den übergebenen Schlüssel nicht unterstützt oder der Schlüssel ein Tupel referenziert, wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

get_handle_tuple

Siehe auch

get_handle_param

Modul

Foundation

get_handle_param (: : Handle, GenParamName, Key : GenParamValue)

Zurückgeben von Informationen über ein Handle.

get_handle_param gibt Informationen über Handle zurück. Der Operator bietet einen generischen Mechanismus, um den Inhalt von Handles aller Typen zu debuggen. Er liefert insbesondere eine Liste von Schlüsseln für das Handle zurück, zu denen Werte mittels get_handle_tuple und get_handle_object abgefragt werden können.

Zu beachten ist, dass get_handle_param nur für die Implementierung generischer Inspektions- und Debugmechanismen gedacht ist. Die für ein Handle zurückgegebenen Schlüssel können sich auch ohne weitere Ankündigung ändern. Weiterhin werden manche Handletypen und Handleeigenschaften von diesem Mechanismus eventuell nicht unterstützt. Dieser Mechanismus ist nicht auf Performanz optimiert. Um die Eigenschaften von Handles zuverlässig abzufragen, etwa zum Steuern des weiteren Programmablaufs, wird empfohlen, die jeweils auf einen Handletyp spezialisierten Operatoren wie get_object_model_3d_params oder get_shape_model_params zu verwenden.

Je nach dem Wert von GenParamName können mit diesem Operator die folgenden Eigenschaften abgefragt werden:

'keys': Gibt in GenParamValue eine Liste von Schlüsseln zurück, die vom übergebenen Handle unterstützt werden. Jeder Schlüssel enthält entweder ein ikonisches Objekt oder ein Tupel. Für diese Abfrage muss der Parameter Key auf ein leeres Tupel gesetzt werden. Aufgrund des generischen Mechanismus dieses Operators sind einige der zurückgegebenen Schlüssel für das übergebene Handle möglicherweise nicht gültig. Für solche Schlüssel liefern get_handle_tuple bzw. get_handle_object einen Fehler zurück.

'key_data_type': Gibt den Datentyp des in Key übergebenen Schlüssels zurück. Der Typ wird in GenParamValue zurückgegeben und ist entweder *'tuple'* oder *'object'*. Je nach Typ kann der dem Schlüssel zugewiesene Wert mit get_handle_tuple oder get_handle_object abgerufen werden.

Parameter

▷ **Handle** (input_control) . . . . . handle ⇝ *handle*
Handle, dessen Eigenschaft abgerufen werden soll.

▷ **GenParamName** (input_control) . . . . . string ⇝ *string*
Abzurufende Eigenschaft.
Defaultwert: 'keys'
Werteliste: GenParamName ∈ {'keys', 'key_data_type'}

▷ **Key** (input_control) . . . . . string(-array) ⇝ *string*
Optionaler Schlüssel, dessen Eigenschaften abgerufen werden.
Defaultwert: []

▷ **GenParamValue** (output_control) . . . . . tuple(-array) ⇝ *string* / integer / real
Zurückgegebener Wert.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_handle_param den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

`clear_handle`, `get_handle_tuple`, `get_handle_object`

Siehe auch

`get_handle_tuple`, `get_handle_object`

Modul

Foundation

get_handle_tuple (: : Handle, Key : Tuple)

Zurückgeben eines über einen Schlüssel referenzierten Tupels eines Handles.

`get_handle_tuple` liefert das Tupel, das von dem Schlüssel `Key` in `Handle` referenziert wird, in `Tuple` zurück. Die Liste der von einem Handle unterstützten Schlüssel kann zuvor mit `get_handle_param` abgerufen werden.

Zu beachten ist, dass `get_handle_tuple` nur für die Implementierung generischer Inspektions- und Debugmechanismen gedacht ist. Die für ein Handle zurückgegebenen Schlüssel können sich auch ohne weitere Ankündigung ändern. Weiterhin werden manche Handletypen und Handleeigenschaften von diesem Mechanismus eventuell nicht unterstützt. Dieser Mechanismus ist nicht auf Performanz optimiert. Um die Eigenschaften von Handles zuverlässig abzufragen, etwa zum Steuern des weiteren Programmablaufs, wird empfohlen, die jeweils auf einen Handletyp spezialisierten Operatoren wie `get_object_model_3d_params` oder `get_shape_model_params` zu verwenden.

Parameter

▷ **Handle** (input_control) . . . . . handle ⇝ *handle*
Handle, dessen Eigenschaft abgerufen werden soll.

▷ **Key** (input_control) . . . . . string ⇝ *string* / integer
Abzurufender Schlüssel.
Parameteranzahl: Key == 1
Restriktion: length(Key) > 0

▷ **Tuple** (output_control) . . . . . tuple(-array) ⇝ *string* / integer / real
Vom Schlüssel referenzierte Tupeldaten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_handle_tuple` den Wert TRUE. Falls das Handle ungültig ist, den übergebenen Schlüssel nicht unterstützt oder der Schlüssel ein ikonisches Objekt referenziert, wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_handle_param`

Alternativen

`get_handle_object`

Siehe auch

`get_handle_param`, `get_handle_object`

Modul

Foundation

tuple_deviation (: : Tuple : Deviation)

Berechnen der Standardabweichung aller Elemente eines Zahlentupels.

tuple_deviation berechnet die Standardabweichung aller Elemente des Eingabetupels Tuple und liefert sie in dem Ausgabeparameter Deviation als Gleitkommazahl zurück. Das Eingabetupel muss ausschließlich aus (ganzen oder Gleitkomma-)Zahlen bestehen.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_deviation, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Deviation := deviation(Tuple)
```

Parameter

- ▷ **Tuple** (input_control) ..... number(-array) ⇝ *integer / real*
  Eingabetupel.
- ▷ **Deviation** (output_control) ..... number(-array) ⇝ *real*
  Standardabweichung der Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_mean, tuple_sum, tuple_min, tuple_max, tuple_length, tuple_median

Modul

Foundation

tuple_histo_range (: : Tuple, Min, Max, NumBins : Histo, BinSize)

Berechnen der Werteverteilung eines Tupels innerhalb eines bestimmten Wertebereichs.

tuple_histo_range erstellt die Werteverteilung Histo des Tupels Tuple im Wertebereich [Min, Max]. Die Werte für Min und Max werden abgerundet falls alle Einträge des Tupels Tuple vom Datentyp integer (Ganzzahl) sind. Der Wertebereich wird in NumBins gleichgroße Klassen aufgeteilt. Die Größe einer solchen Klasse wird in BinSize ausgegeben. Liegt ein Wert zwischen zwei Klassen wird er immer der kleineren Klasse zugeordnet. Bei einem Tupel mit Einträgen vom Datentyp real (Gleitkommazahl) ist die Klassengröße $\mathtt{BinSize} = \frac{\mathtt{Max}-\mathtt{Min}}{\mathtt{NumBins}}$. Sind alle Einträge vom Datentyp integer (Ganzzahl) wird die Klassengröße über $\mathtt{BinSize} = \frac{\mathtt{Max}-\mathtt{Min}+1}{\mathtt{NumBins}}$ berechnet.

Sonderfall: Leere Eingabetupel

Falls eines oder mehrere Eingabetupel leer sind, wird eine Fehlermeldung angezeigt.

Achtung

Bei Tupeln, die nur Daten vom Typ integer (Ganzzahl) enthalten kann es in Abhängigkeit von BinSize zu den folgenden Effekten kommen: Für $\mathtt{BinSize} > 1$ werden mehrere aufeinanderfolgende Zahlen der selben Klasse zugeordnet. Ist BinSize dabei keine natürliche Zahl werden die Zahlen ungleichmäßig auf die Klassen verteilt (Für $\mathtt{BinSize} = 1.5$ werden beispielsweise die ersten beiden Zahlen der ersten Klasse, die dritte Zahl der zweiten Klasse und die vierte und fünfte Zahl der dritten Klasse zugeordnet). Dies macht sich durch einzelne Spitzen im Histogramm Histo bemerkbar. Gilt $\mathtt{BinSize} < 1$ werden je nach BinSize einigen Klassen keine Zahlen zugeordnet (Für $\mathtt{BinSize} = 0.5$ wird beispielsweise die erste Zahl der ersten Klasse und die zweite Zahl der dritten Klasse zugeordnet). Das Histogramm Histo weist dann einige Lücken auf, was der Struktur eines Kamms ähnelt.

Bei einem Tupel mit Einträgen vom Datentyp `real` (Gleitkommazahl) werden für `Min` = `Max` alle Einträge des entsprechenden Wertes nur der ersten Klasse zugeordnet.

Parameter

- ▷ **`Tuple`** (input_control) ..... number-array ⇝ *real* / integer
  Eingabetupel.
- ▷ **`Min`** (input_control) ..... number ⇝ *real* / integer
  Minimaler Wert.
- ▷ **`Max`** (input_control) ..... number ⇝ *real* / integer
  Maximaler Wert.
  Restriktion: Max >= Min
- ▷ **`NumBins`** (input_control) ..... integer ⇝ *integer*
  Anzahl Klassen.
  Restriktion: NumBins >= 1
- ▷ **`Histo`** (output_control) ..... histogram(-array) ⇝ *integer*
  Zu berechnendes Histogramm.
- ▷ **`BinSize`** (output_control) ..... real ⇝ *real*
  Klassengröße.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_histo_range` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`tuple_min`, `tuple_max`

Nachfolger

`create_funct_1d_array`

Siehe auch

`gray_histo`, `gray_histo_abs`, `gray_histo_range`

Modul

Foundation

`tuple_length` (: : Tuple : Length)

Bestimmen der Länge eines Tupels.

`tuple_length` liefert die Anzahl der Elemente des Eingabetupels `Tuple` zurück.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator 0 zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_length`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Length := |Tuple|
```

Parameter

- ▷ **`Tuple`** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **`Length`** (output_control) ..... integer ⇝ *integer*
  Anzahl der Elemente des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_min, tuple_max, tuple_mean, tuple_deviation, tuple_sum, tuple_median

Siehe auch

tuple_type

Modul

Foundation

tuple_max (: : Tuple : Max)

Bestimmen des maximalen Elements eines Tupels.

tuple_max bestimmt das maximale Element von allen Elementen des Eingabetupels Tuple und liefert es in dem Ausgabeparameter Max zurück. Das Eingabetupel muss hierfür entweder durchwegs Strings enthalten oder ausschließlich (ganze oder Gleitkomma-)Zahlen beinhalten. Eine Mischung aus Strings und Zahlen ist nicht erlaubt. Das Ergebnis besteht in einer Gleitkommazahl, sobald mindestens ein Element des Eingabetupels eine Gleitkommazahl ist. Falls alle Elemente von Tuple hingegen ganze Zahlen sind, so besteht auch Max aus einer ganzen Zahl.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_max, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Max := max(Tuple)
```

Parameter

▷ **Tuple** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Max** (output_control) . . . . . tuple(-array) ⇝ *real* / integer / string
Maximales Element aller Eingabetupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_min, tuple_mean, tuple_deviation, tuple_sum, tuple_length, tuple_median

Siehe auch

tuple_max2, tuple_min2

Modul

Foundation

tuple_mean (: : Tuple : Mean)

Berechnen des Durchschnittswerts aller Elemente eines Zahlentupels.

tuple_mean berechnet den Durchschnittswert aller Elemente des Eingabetupels Tuple und liefert ihn in dem Ausgabeparameter Mean als Gleitkommazahl zurück. Das Eingabetupel muss ausschließlich aus (ganzen oder Gleitkomma-)Zahlen bestehen.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_mean, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Mean := mean(Tuple)
```

Parameter

- ▷ **Tuple** (input_control) ..... number(-array) ⇝ *integer* / real
  Eingabetupel.
- ▷ **Mean** (output_control) ..... number ⇝ *real*
  Durchschnittswert der Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_deviation, tuple_sum, tuple_min, tuple_max, tuple_length, tuple_median

Modul

Foundation

tuple_median (: : Tuple : Median)

Berechnen des Medians aller Elemente eines Zahlentupels.

tuple_median berechnet den Median aller Elemente des Eingabetupels Tuple und liefert ihn in dem Ausgabeparameter Median zurück. Das Eingabetupel muss ausschließlich aus (ganzen oder Gleitkomma-)Zahlen bestehen. Der Median ist definiert als das Element mit Rang n/2 (siehe tuple_select_rank).

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_median, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Median := median(Tuple)
```

Parameter

- ▷ **Tuple** (input_control) ..... number(-array) ⇝ *integer* / real
  Eingabetupel.
- ▷ **Median** (output_control) ..... number ⇝ *integer* / real
  Median der Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_select_rank

Siehe auch

tuple_mean, tuple_min, tuple_max

Modul

Foundation

tuple_min (: : Tuple : Min)

Bestimmen des minimalen Elements eines Tupels.

tuple_min bestimmt das minimale Element von allen Elementen des Eingabetupels Tuple und liefert es in dem Ausgabeparameter Min zurück. Das Eingabetupel muss hierfür entweder durchwegs Strings enthalten oder ausschließlich (ganze oder Gleitkomma-)Zahlen beinhalten. Eine Mischung aus Strings und Zahlen ist nicht erlaubt. Das Ergebnis besteht in einer Gleitkommazahl, sobald mindestens ein Element des Eingabetupels eine Gleitkommazahl ist. Falls alle Elemente von Tuple hingegen ganze Zahlen sind, so besteht auch Min aus einer ganzen Zahl.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, wird eine Fehlermeldung angezeigt.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_min, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Min := min(Tuple)
```

Parameter

- ▷ **Tuple** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Min** (output_control) ..... tuple(-array) ⇝ *real* / integer / string
  Minimales Element aller Eingabetupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_max, tuple_mean, tuple_deviation, tuple_sum, tuple_length, tuple_median

Siehe auch

tuple_max2, tuple_min2

Modul

Foundation

tuple_sum (: : Tuple : Sum)

Berechnen der Summe aller Elemente eines Tupels.

tuple_sum berechnet die Summe aus allen Elementen des Eingabetupels Tuple und liefert sie in dem Ausgabeparameter Sum zurück. Das Eingabetupel muss hierfür entweder durchwegs Strings enthalten oder ausschließlich (ganze oder Gleitkomma-)Zahlen beinhalten. Eine Mischung aus Strings und Zahlen ist nicht erlaubt. Das Ergebnis besteht in einer Gleitkommazahl, sobald mindestens ein Element des Eingabetupels eine Gleitkommazahl ist. Falls alle Elemente von Tuple hingegen ganze Zahlen sind, so besteht auch Sum aus einer ganzen Zahl. Bei Strings wird zur Summenbildung statt der Addition die Konkatenation verwendet, so dass die einzelnen Elemente der Reihe nach verknüpft werden.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_sum`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Sum := sum(Tuple)
```

Parameter

▷ **Tuple** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Sum** (output_control) ..... tuple(-array) ⇝ *real* / integer / string
Summe der Tupelelemente.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_mean`, `tuple_deviation`, `tuple_min`, `tuple_max`, `tuple_length`, `tuple_median`

Siehe auch

`tuple_cumul`

Modul

Foundation

28.12 String-Operationen

Dieses Kapitel beschreibt Operatoren für String-Operationen.

Allgemeine Information

Die HALCON-Bibliothek kodiert Strings standardmäßig in UTF-8.

UTF-8 ist eine Kodierung für Unicode-Zeichen, die alle Unicode-Codepoints mit einem bis vier Bytes kodieren kann. „Unicode“ bezeichnet den Zeichensatz, mit dem jedem Zeichen eines Strings ein Codepoint (z.B. ’U+0041’ für ’A’) zugewiesen werden kann. Mit UTF-8 werden dann die Unicode-Codepoints in Binärdaten übersetzt. Der Speicherbedarf für alle ASCII-Zeichen beträgt in der Regel 1 Byte. Bestimmte Zeichen wie beispielsweise deutsche Umlaute, oder griechische und kyrillische Zeichen erfordern 2 Bytes. Asiatische Zeichen belegen bis zu 4 Bytes je Zeichen.

Die HALCON-String-Operatoren arbeiten standardmäßig auf Basis von Unicode-Codepoints. D.h., beim Zugriff auf ein Zeichen eines Strings wird immer der entsprechende Unicode-Codepoint des Zeichens zurückgegeben, unabhängig davon, wie viele Bytes benötigt werden, um den Codepoint in UTF-8 darzustellen. Somit können Multi-Byte Zeichen, wie asiatische Zeichen oder deutsche Umlaute auf allen Systemen einheitlich übersetzt werden. Bitte beachten Sie, dass der Unicode-Standard es auch erlaubt, druckbare Zeichen aus mehreren Codepoints zusammenzusetzen (mit Hilfe sogenannter „Combining Diacritical Marks“). Dies wird derzeit von HALCON nicht vollständig unterstützt: in HALCON werden die Codepoints separat verarbeitet, und beim Vergleich von Strings werden äquivalente Zeichen nicht gleich gesetzt, wenn sie mit unterschiedlichen Codepoints codiert sind.

Sollte es beim Ausführen von alten Programmen Kompatibilitätsprobleme geben, kann die Stringkodierung der HALCON-Bibliothek von *’utf8’* auf *’locale’* gesetzt werden (Legacy-Modus). Dann werden Strings abhängig von der Ländereinstellung gespeichert und die String-Operatoren arbeiten – wie in früheren HALCON-Versionen – nicht zeichen- sondern byteweise. Sollte auch im UTF-8-Modus eine byteweise Verarbeitung von Zeichen nötig sein, kann mit dem Operator `set_system` die Option *’tuple_string_operator_mode’* von *’codepoint’* auf *’byte’* gesetzt werden. Dann arbeiten die String-Operatoren nicht mehr auf der Basis von Codepoints. Die Bytefolge eines Strings kann beispielsweise zum Debugging interessant sein.

tuple_environment (: : Names : Values)

Einlesen einer oder mehrerer Umgebungsvariablen.

tuple_environment liest den Inhalt aller Umgebungsvariablen, deren Namen in dem Eingabetupel Names definiert sind und liefert ihn in den Elementen des Ausgabetupels Values zurück. Das Eingabetupel darf nur Strings enthalten. Enthält das Eingabetupel einen Namen, für den keine gültige Umgebungsvariable existiert, so wird für diesen Namen ein leerer String als Ergebnis zurückgeliefert.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_environment, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Values := environment(Names)
```

Parameter

▷ **Names** (input_control) ..... string(-array) ⇝ *string*
Eingabetupel mit den Namen der Umgebungsvariable(n).

▷ **Values** (output_control) ..... string(-array) ⇝ *string*
Inhalt der Umgebungsvariable(n).

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_strstr, tuple_strrstr, tuple_strchr, tuple_strrchr, tuple_strlen, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_substr

Modul

Foundation

tuple_regexp_match (: : Data, Expression : Matches)

Extrahieren von Teilen aus Zeichenketten mit Hilfe von regulären Ausdrücken.

tuple_regexp_match wendet den regulären Ausdruck Expression auf eine oder mehrere Zeichenketten in Data an und liefert jeweils die erste dem Suchmuster entsprechende Teilzeichenkette in Matches zurück. In der Regel wird eine Ausgabezeichenkette pro Eingabezeichenkette zurückgegeben, wobei die Zeichenketten leer sind, wenn keine Übereinstimmung gefunden wurde. Falls der reguläre Ausdruck jedoch Gruppen mit Teilergebnissen (siehe unten) enthält, hängt das Verhalten von der Anzahl der Eingabezeichenketten ab: Gibt es nur eine Eingabe, so ist das Ergebnis das Tupel aller Teilzeichenketten, die dem Suchmuster einer Gruppe entsprechen. Gibt es mehrere Eingaben, so wird für jede Eingabe das Ergebnis der ersten Gruppe zurückgegeben.

Eine Zusammenfassung der Syntax regulärer Ausdrücke folgt. Grundsätzlich steht jedes Zeichen im Suchmuster als Literal für ein entsprechendes Zeichen in den Eingabedaten, wobei jedoch die nachfolgend aufgeführten Symbole eine spezielle Bedeutung haben (die beschriebene Syntax ist kompatibel zu Perl):

^ Steht für den Anfang der Zeichenkette
$ Steht für das Ende der Zeichenkette, ggf. inklusive einem Newline als letztes Zeichen
. Steht für jedes Zeichen außer Newline
[...] Steht für jedes in den eckigen Klammern aufgeführte Literal. Ist das erste Zeichen ein '^', so ist der Ausdruck negiert. Mit dem Zeichen '-' kann man Wertebereiche angeben wie in '[A-Z0-9]'. Andere Zeichen verlieren ihre spezielle Bedeutung innerhalb eckiger Klammern, außer '\'.
Innerhalb der eckigen Klammern können die folgenden POSIX Zeichenklassen verwendet werden (dabei ist zu beachten, dass die eckigen Klammern um den Klassennamen zusätzlichen angegeben werden müssen):
[:alnum:] Ziffern und Buchstaben
[:alpha:] Buchstaben
[:blank:] Leerzeichen und Tab
[:cntrl:] Steuerzeichen
[:digit:] Ziffern
[:graph:] Alle druckbaren Zeichen ohne Leerzeichen
[:lower:] Kleinbuchstaben
[:print:] Alle druckbaren Zeichen einschließlich Leerzeichen
[:punct:] Interpunktionszeichen
[:space:] Alle Whitespace-Zeichen ([:blank:], Zeilenumbrüche, ...)
[:upper:] Großbuchstaben
[:xdigit:] Für Hexadezimalzahlen verwendete Buchstaben und Ziffern (0-9a-fA-F).

* Erlaubt 0 oder mehr Wiederholungen des vorhergehenden Literals / Gruppe
+ Erlaubt 1 oder mehr Wiederholungen
? Erlaubt 0 oder 1 Wiederholung
{n,m} Erlaubt n bis m Wiederholungen
{n} Erlaubt genau n Wiederholungen

Die obigen Wiederholungsquantifikatoren sind im Normalfall "gierig", d.h. es wird die maximal mögliche Übereinstimmung gesucht. Mit einem zusätzlichen ? wird die minimale Übereinstimmung gesucht, z.B. +?

| Trennt alternative Suchausdrücke
( ) Gruppiert einen Unterausdruck und speichert ein Teilergebnis.
(?:) Reine Gruppierung (z.B. für Wiederholungen) ohne Teilergebnis.
(?=) Positive Vorausschau (lookahead, notwendige Bedingung rechts vom Match)
(?!) Negative Vorausschau (lookahead, unerlaubte Bedingung rechts vom Match)
(?<=) Positive Rückschau (lookbehind, notwendige Bedingung links vom Match)
(?<!) Negative Rückschau (lookbehind, unerlaubte Bedingung links vom Match)

\ Escape-Zeichen, erlaubt die Verwendung eines Symbols mit spezieller Bedeutung als Literal. Achtung: Manche Sprachen wie HDevelop und C/C++ verwenden den Backslash bereits als allgemeines Escape-Zeichen für Zeichenketten. In diesem Fall steht '\\.' für ein Punktliteral und '\\\\' für ein Backslashliteral. Darüber hinaus gibt es einige Sequenzen mit spezieller Bedeutung (wobei die großgeschriebene Variante jeweils die Negation darstellt):
\d, \D Steht für Ziffern
\w, \W Steht für Ziffern, Buchstaben und Unterstrich
\s, \S Steht für Leerraum (Whitespace)
\b, \B Steht für eine Wortgrenze

Falls der angegebene Ausdruck syntaktisch inkorrekt ist, wird als Fehler ein falscher Wert im Steuerparameter 2 bemängelt. Zusätzliche Informationen erhält man in einem Benachrichtigungsfenster, wenn `set_system ('do_low_error','true')` gesetzt ist, oder aber in der Ausgabekonsole von HDevelop.

Weiterhin lassen sich einige Optionen setzen, indem man für `Expression` ein Tupel übergibt. In diesem Fall wird das erste Element als regulärer Ausdruck behandelt, und jedes weitere Element als Option.

- *'ignore_case'*: Ignorieren von Groß- und Kleinschreibung
- *'multiline'*: '^' und *'$'* stehen für Start und Ende einzelner Zeilen
- *'dot_matches_all'*: *'.'* steht auch für Newlines

- *'newline_lf'*, *'newline_crlf'*, *'newline_cr'*: Kodierung von Newlines in den Eingabedaten. Als Default wird auf allen Systemen LF verwendet (obwohl in Windows-Dateien normalerweise CRLF als Zeilenumbruch verwendet wird, werden beim Einlesen dieser Dateien die Zeilenumbrüche i.d.R. in *'\n'* umgewandelt, was dem LF entspricht).

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

Falls der Eingabeparameter `Data` ein leeres Tupel ist, gibt der Operator ein leeres Tupel zurück. Wenn `Expression` ein leeres Tupel ist, wird ein Fehler zurückgegeben.

Unicode Codepoints versus Bytes

Die Auswertung der regulären Ausdrücke berücksichtigt Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Auswertung der regulären Ausdrücke sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit `set_system ('tsp_tuple_string_operator_mode','byte')` in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_regexp_match`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Matches := regexp_match(Data, Expression)
```

Parameter

▷ **`Data`** (input_control) ..... string(-array) ⇝ *string*
Eingabezeichenketten, auf die das Suchmuster angewendet wird

▷ **`Expression`** (input_control) ..... string(-array) ⇝ *string*
Regulärer Ausdruck.
Defaultwert: '.*'
Wertevorschläge: `Expression` ∈ {'.*', 'ignore_case', 'multiline', 'dot_matches_all', 'newline_lf', 'newline_crlf', 'newline_cr'}

▷ **`Matches`** (output_control) ..... string(-array) ⇝ *string*
Dem Suchmuster entsprechende Teilzeichenketten

Beispiel

```
tuple_regexp_match ('abba', 'a*b*', Result)
* Returns 'abb'

tuple_regexp_match ('abba', 'b*a*', Result)
* Returns 'a'

tuple_regexp_match ('abba', 'b+a*', Result)
* Returns 'bba'

tuple_regexp_match ('abba', '.a', Result)
* Returns 'ba'

tuple_regexp_match ('abba', '[ab]*', Result)
* Returns 'abba'

tuple_regexp_match (['img123','img124'], 'img(.*)', Result)
* Returns ['123','124']

tuple_regexp_match ('mydir/img001.bmp', 'img(.*)\\.(.*)', Result)
* Returns ['001','bmp']
```

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_strstr

Siehe auch

tuple_regexp_replace, tuple_regexp_test, tuple_regexp_select

Literatur

Perl Compatible Regular Expressions (PCRE), http://www.pcre.org/

Modul

Foundation

tuple_regexp_replace (: : Data, Expression, Replace : Result)

Ersetzen eines Teils einer Zeichenkette mit Hilfe von regulären Ausdrücken.

tuple_regexp_replace wendet den regulären Ausdruck Expression auf eine oder mehrere Zeichenketten in Data an und ersetzt jeweils die **erste** dem Suchmuster entsprechende Teilzeichenkette durch Replace. Die bearbeiteten Zeichenketten werden in Result zurückgegeben.

Für die Beschreibung von Syntax und Optionen des regulären Ausdrucks wird auf die Dokumentation des Operators tuple_regexp_match verwiesen. Zusätzlich unterstützt der Operator tuple_regexp_replace noch die Option *'replace_all'*, mit der bei mehrfachen Übereinstimmungen auch mehrfache Ersetzungen durchgeführt werden.

Im Ausdruck Replace kann man mit *'$0'* auf die dem Suchmuster entsprechende Teilzeichenkette verweisen, mit *'$i'* auf das Teilergebnis der i-ten Gruppe (für $i <= 9$), sowie mit *'$$'* auf das Literal *'$'*.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

Wenn Data ein leeres Tupel ist, gibt der Operator ein leeres Tupel zurück. Wenn Replace ein leeres Tupel ist und Data nicht leer ist, wird eine Fehlermeldung angezeigt.

Unicode Codepoints versus Bytes

Die Auswertung der regulären Ausdrücke berücksichtigt Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Auswertung der regulären Ausdrücke sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit set_system ('tsp_tuple_string_operator_mode','byte') in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_regexp_replace, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Result := regexp_replace(Data, Expression, Replace)
```

Parameter

- ▷ **Data** (input_control) ..... string(-array) ⇝ *string*
  Zu bearbeitende Eingabezeichenketten
- ▷ **Expression** (input_control) ..... string(-array) ⇝ *string*
  Regulärer Ausdruck.
  Defaultwert: '.*'
  Wertevorschläge: Expression ∈ {'.*', 'replace_all', 'ignore_case', 'multiline', 'dot_matches_all', 'newline_lf', 'newline_crlf', 'newline_cr'}
- ▷ **Replace** (input_control) ..... string ⇝ *string*
  Ersetzungsausdruck.
- ▷ **Result** (output_control) ..... string(-array) ⇝ *string*
  Bearbeitete Zeichenketten.

Beispiel

```
tuple_regexp_replace(['img10.bmp','img11.bmp','img12.bmp'], \
                     'img(.*).bmp', 'out$1.txt', Result)
* Returns ['out10.txt','out11.txt','out12.txt']
```

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

tuple_regexp_match, tuple_regexp_test, tuple_regexp_select

Modul

Foundation

tuple_regexp_select (: : Data, Expression : Selection)

Auswählen von Tupelelementen, die einem regulären Ausdruck entsprechen.

tuple_regexp_select wendet den regulären Ausdruck Expression auf eine oder mehrere Zeichenketten in Data an, und gibt die Elemente, die eine dem Suchmuster entsprechende Teilzeichenkette enthalten, in Selection zurück. Dies ist ein bequemer Weg, um z.B. ein über den Operator list_files erhaltenes Tupel von Dateinamen zu filtern.

Für die Beschreibung von Syntax und Optionen des regulären Ausdrucks wird auf die Dokumentation des Operators tuple_regexp_match verwiesen. Zusätzlich unterstützt der Operator tuple_regexp_test noch die Option *'invert_match'*, mit der die Elemente ausgewählt werden, die dem Suchmuster **nicht** entsprechen.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

Unicode Codepoints versus Bytes

Die Auswertung der regulären Ausdrücke berücksichtigt Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Auswertung der regulären Ausdrücke sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit set_system ('tsp_tuple_string_operator_mode','byte') in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_regexp_select, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Selection := regexp_select(Data, Expression)
```

Parameter

▷ **Data** (input_control) ..... string(-array) ⇝ *string*
Eingabezeichenketten, auf die das Suchmuster angewendet wird

▷ **Expression** (input_control) ..... string(-array) ⇝ *string*
Regulärer Ausdruck.
Defaultwert: '.*'
Wertevorschläge: Expression ∈ {'.*', 'invert_match', 'ignore_case', 'multiline', 'dot_matches_all', 'newline_lf', 'newline_crlf', 'newline_cr'}

▷ **Selection** (output_control) ..... string(-array) ⇝ *string*
Zeichenketten mit Übereinstimmung

Beispiel

```
tuple_regexp_select (['.','..','mydir','a.png','b.txt','c.bmp','d.dat'], \
                     '.(bmp|png)', Result)
* Returns ['a.png','c.bmp']

tuple_regexp_select (Files, ['training','invert_match'], Matches)
* Returns all file names that do *not* contain the string 'training'
```

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_select

Siehe auch

tuple_regexp_match, tuple_regexp_replace, tuple_regexp_test

Modul

Foundation

tuple_regexp_test (: : Data, Expression : NumMatches)

Testen, ob eine Zeichenkette einem regulären Ausdruck entspricht.

tuple_regexp_test wendet den regulären Ausdruck Expression auf eine oder mehrere Zeichenketten in Data an, und gibt die Anzahl der Elemente, die eine dem Suchmuster entsprechende Teilzeichenkette enthalten, in NumMatches zurück. Insbesondere ist für eine einzelne Eingabe das Ergebnis 1, falls die Zeichenkette eine Übereinstimmung enthält, ansonsten 0.

Für die Beschreibung von Syntax und Optionen des regulären Ausdrucks wird auf die Dokumentation des Operators tuple_regexp_match verwiesen. Zusätzlich unterstützt der Operator tuple_regexp_test noch die Option *'invert_match'*, mit der die Elemente gezählt werden, die dem Suchmuster **nicht** entsprechen.

Für die bequeme Verwendung in einer Bedingung ist diese Funktionalität in HDevelop auch als Operator =~ verfügbar.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

Falls das Eingabetupel leer ist, gibt der Operator 0 zurück.

Unicode Codepoints versus Bytes

Die Auswertung der regulären Ausdrücke berücksichtigt Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Auswertung der regulären Ausdrücke sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit set_system ('tsp_tuple_string_operator_mode','byte') in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_regexp_test, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

NumMatches := regexp_test(Data, Expression) oder NumMatches := Data =~ Expression

Parameter

▷ **Data** (input_control) ..... string(-array) ⇝ *string*
Eingabezeichenketten, auf die das Suchmuster angewendet wird

▷ **Expression** (input_control) ..... string(-array) ⇝ *string*
Regulärer Ausdruck.
Defaultwert: '.*'
Wertevorschläge: Expression ∈ {'.*', 'invert_match', 'ignore_case', 'multiline', 'dot_matches_all', 'newline_lf', 'newline_crlf', 'newline_cr'}

▷ **NumMatches** (output_control) ..... integer ⇝ *integer*
Anzahl der Zeichenketten mit Übereinstimmung

Beispiel

```
tuple_regexp_test ('p10662599755', '[A-Z]*', Result)
* Returns 0

tuple_regexp_test ('p10662599755', ['[A-Z]*','ignore_case'], Result)
* Returns 1
```

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_strstr

Siehe auch

tuple_regexp_match, tuple_regexp_replace, tuple_regexp_select

Modul

Foundation

tuple_split (: : String, Separator : Substrings)

Aufteilen von Strings in mehrere Strings mittels Trennsymbolen.

tuple_split sucht in den einzelnen Strings des Eingabetupels String nach den Trennsymbolen, die in dem Eingabetupel Separator definiert sind. tuple_split teilt dann die untersuchten Strings in die Teilstrings, die durch die Trennsymbole separiert werden, und gibt diese in Substrings zurück. Das Verhalten hängt von der Länge der Eingabe-Tupel ab:

- Separator enthält genau einen String: Jeder String in String wird durch die Trennsymbole in Separator aufgeteilt. Z.B. sei String = ['alpha:1', 'beta:2', 'gamma:3'] und Separator = ['a:']. Dann gibt der Operator das Ausgabetupel Substrings = ['lph', '1', 'bet', '2', 'g', 'mm', '3'] zurück.
- String enthält genau einen String: Der einzelne String wird je Element des Tupels Separator aufgeteilt. Z.B. sei String = 'alpha:1 beta:2 gamma:3' und Separator = [':', '123']. Dann besteht das Ausgabetupel Substrings aus den Teilstrings 'alpha', '1', 'beta', '2', 'gamma' und '3' aufgrund der Trennung durch das erste Element in Separator (':') sowie aus den Teilstrings 'alpha:', 'beta:' und 'gamma:' aufgrund der Trennung durch das zweite Element in Separator ('123').
- Beide Tupel beinhalten die selbe Anzahl Strings: Die Suche geschieht elementweise. Das heißt, der erste String von String wird gemäß den Trennsymbolen des ersten Strings von Separator aufgeteilt, der zweite String von String wird gemäß den Trennsymbolen des zweiten Strings von Separator aufgeteilt und so weiter.
- Eines oder beide der Eingabetupel sind leer: Ein leeres Tupel wird zurück gegeben.

Anmerkungen zum Parameter Separator: Enthalten die Strings von Separator mehr als ein Zeichen, so werden alle in einem String enthaltenen Zeichen als Trennsymbole interpretiert (siehe das oben gegebene Beispiel). Folgen in einem String aus String mehrere Trennsymbole direkt einander, so werden diese wie ein einzelnes Trennsymbol behandelt. Trennsymbole am Anfang und am Ende von String führen nicht zu einem leeren Teilstring.

Es gilt zu beachten, dass beide Eingabetupel (String und Separator) ausschließlich aus Strings bestehen müssen. Anderenfalls bricht tuple_split mit einem entsprechenden Fehler ab. Weisen die beiden Eingabetupel eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_split mit einem entsprechenden Fehler ab.

Unicode Codepoints versus Bytes

Die Trennzeichen werden als Unicode Codepoints behandelt. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Trennzeichen als einzelne Bytes behandelt werden sollen, kann dieser Operator mit `set_system('tsp_tuple_string_operator_mode','byte')` in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_split`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Substrings := split(String, Separator)
```

Parameter

- ▷ **`String`** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit aufzuteilenden Strings.
- ▷ **`Separator`** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit Trennsymbolen.
- ▷ **`Substrings`** (output_control) ..... string(-array) ⇝ *string*
  Teil-Strings nach Aufteilung mittels Trennsymbolen.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_strstr`, `tuple_strrstr`, `tuple_strchr`, `tuple_strrchr`, `tuple_strlen`, `tuple_str_first_n`, `tuple_str_last_n`, `tuple_environment`

Modul

Foundation

`tuple_str_first_n` (: : String, Position : Substring)

Ausschneiden aller Zeichen bis zur Position „n“ aus einem Stringtupel.

`tuple_str_first_n` schneidet aus jedem String des Eingabetupels `String` die ersten Zeichen bis einschließlich zur Position „n“ aus und liefert sie als einen neuen String in dem Ausgabetupel `Substring` zurück (Bemerkung: Die Position innerhalb eines Strings wird beginnend mit 0 gezählt). Die Zahl „n“ wird hierbei durch das zweite Eingabetupel `Position` definiert. Enthält `Position` nur ein Element, so legt dieses Element die Zahl „n“ für alle Strings von `String` fest. Beinhaltet `Position` genauso viele Elemente wie `String`, so legt das erste Element von `Position` die Zahl „n“ für den ersten String von `String` fest, das zweite Element von `Position` legt die Zahl „n“ für den zweiten String von `String` fest und so weiter. Enthält `Position` mehr als ein Element und besteht `String` nur aus einem String, so werden aus diesem String mehrere Anfangssegmente ausgeschnitten, deren Längen durch die Elemente von `Position` festgelegt sind. Beinhalten die beiden Eingabetupel mehr als ein Element und besitzen sie eine unterschiedliche Zahl an Elementen, so bricht `tuple_str_first_n` mit einer Fehlermeldung ab.

Falls beide Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück. Gleichermaßen gibt der Operator ein leeres Tupel zurück, wenn `String` leer ist und `Position` nicht. Falls jedoch `Position` leer ist und `String` nicht, wird eine Fehlermeldung angezeigt.

Unicode Codepoints versus Bytes

Die Position bezieht sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Position sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit `set_system('tsp_tuple_string_operator_mode','byte')` in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_str_first_n`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Substring := str_firstn(String, Position)
```

Parameter

- ▷ **`String`** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **`Position`** (input_control) . . . . . number(-array) ⇝ *integer* / real
  Eingabetupel mit den Positionen „n“.
- ▷ **`Substring`** (output_control) . . . . . string(-array) ⇝ *string*
  Die ersten Zeichen bis zu den Positionen „n“ aller Eingabestrings.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_str_last_n`, `tuple_substr`, `tuple_strstr`, `tuple_strrstr`, `tuple_strlen`, `tuple_strchr`, `tuple_strrchr`, `tuple_split`, `tuple_environment`

Modul

Foundation

`tuple_str_last_n ( : : String, Position : Substring )`

Ausschneiden aller Zeichen ab der Position „n“ aus einem Stringtupel.

`tuple_str_last_n` schneidet aus jedem String des Eingabetupels `String` alle Zeichen bis zum String-Ende beginnend mit dem Zeichen an der Position „n“ aus und liefert sie als einen neuen String in dem Ausgabetupel `Substring` zurück. Die Zahl „n“ wird hierbei durch das zweite Eingabetupel `Position` definiert. Enthält `Position` nur ein Element, so legt dieses Element die Zahl „n“ für alle Strings von `String` fest. Beinhaltet `Position` genauso viele Elemente wie `String`, so legt das erste Element von `Position` die Zahl „n“ für den ersten String von `String` fest, das zweite Element von `Position` legt die Zahl „n“ für den zweiten String von `String` fest und so weiter. Enthält `Position` mehr als ein Element und besteht `String` nur aus einem String, so werden aus diesem String mehrere Endsegmente ausgeschnitten, deren Länge durch die Elemente von `Position` festgelegt ist. Beinhalten die beiden Eingabetupel mehr als ein Element und besitzen sie eine unterschiedliche Zahl an Elementen, so bricht `tuple_str_last_n` mit einer Fehlermeldung ab.

Falls beide Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück. Gleichermaßen gibt der Operator ein leeres Tupel zurück, wenn `String` leer ist und `Position` nicht. Falls jedoch `Position` leer ist und `String` nicht, wird eine Fehlermeldung angezeigt.

Unicode Codepoints versus Bytes

Die Position bezieht sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Position sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit `set_system('tsp_tuple_string_operator_mode','byte')` in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_str_last_n`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Substring := str_lastn(String, Position)
```

Parameter

- ▷ **String** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **Position** (input_control) . . . . . number(-array) ⇝ *integer* / real
  Eingabetupel mit den Positionen „n“.
- ▷ **Substring** (output_control) . . . . . string(-array) ⇝ *string*
  Alle Zeichen von den Positionen „n“ bis zum String-Ende für alle Eingabestrings.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_str_first_n`, `tuple_substr`, `tuple_strstr`, `tuple_strrstr`, `tuple_strlen`, `tuple_strchr`, `tuple_strrchr`, `tuple_split`, `tuple_environment`

Modul

Foundation

tuple_strchr (: : String, ToFind : Position)

Vorwärtssuchen nach einem Zeichen in einem Tupel von Strings.

`tuple_strchr` sucht in dem Eingabetupel `String` nach den in dem Eingabetupel `ToFind` definierten Zeichen. Beide Eingabetupel müssen durchgehend aus Strings bestehen. Anderenfalls bricht `tuple_strchr` mit einem entsprechenden Fehler ab. Enthalten die Strings von `ToFind` mehr als ein Zeichen, so wird ausschließlich das erste Zeichen in einem String berücksichtigt und alle folgenden Zeichen ignoriert. Besteht `String` nur aus einem einzigen String, so werden in diesem alle in `ToFind` definierten Zeichen gesucht. Das Ausgabetupel besteht also in diesem Fall aus genauso vielen Elementen wie das Eingabetupel `ToFind`. Ist die Suche nach einem Zeichen erfolgreich, so wird die Position (beginnend mit 0 für das erste Zeichen eines Strings), an der das Zeichen das erste Mal innerhalb des untersuchten Strings gefunden wurde, in dem Ausgabetupel `Position` zurückgegeben. Wurde ein Zeichen nicht gefunden, so wird für dessen Position der Wert -1 zurückgegeben. Enthalten beide Eingabetupel `String` und `ToFind` gleich viele Elemente, so werden sie elementweise bearbeitet, das heißt, das erste Zeichen im ersten Element von `ToFind` wird in dem ersten String von `String`, das erste Zeichen des zweiten ELements von `ToFind` wird im zweiten String von `String` gesucht und so weiter. Die einzelnen Elemente des Ausgabetupels enthalten dann das Resultat der elementweisen Suche. Besteht `ToFind` nur aus einem Element, so wird das dort definierte Zeichen in allen Strings des Eingabetupels `String` gesucht. Das Ausgabetupel `Position` besitzt dann also genauso viele Elemente wie das Eingabetupel `String`. Weisen die beiden Eingabetupel `String` und `ToFind` eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht `tuple_strchr` mit einem entsprechenden Fehler ab.

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

Unicode Codepoints versus Bytes

Die Position bezieht sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Position sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit `set_system('tsp_tuple_string_operator_mode','byte')` in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_strchr`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Position := strchr(String, ToFind)
```

Parameter

▷ **String** (input_control) . . . . . string(-array) ⇝ *string*
Eingabetupel mit zu untersuchenden Strings.

▷ **ToFind** (input_control) . . . . . string(-array) ⇝ *string*
Eingabetupel mit zu suchenden Zeichen.

▷ **Position** (output_control) . . . . . integer(-array) ⇝ *integer*
Positionen der gesuchten Zeichen in den untersuchten Strings.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_strrchr`, `tuple_strstr`, `tuple_strrstr`, `tuple_strlen`, `tuple_substr`, `tuple_str_first_n`, `tuple_str_last_n`, `tuple_split`, `tuple_environment`

Modul

Foundation

tuple_strlen (: : T1 : Length)

Berechnen der Länge der einzelnen Strings eines Stringtupels.

`tuple_strlen` untersucht jeden String des Eingabetupels `T1` auf dessen Länge und liefert die Länge der einzelnen Strings in dem Ausgabetupel `Length` zurück. Das Eingabetupel `T1` darf nur Strings enthalten. Ansonsten bricht `tuple_strlen` mit einer entsprechenden Fehlermeldung ab.

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

Unicode Codepoints versus Bytes

Die Länge des Strings bezieht sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Länge sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit `set_system('tsp_tuple_string_operator_mode','byte')` in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_strlen`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Length := strlen(T1)
```

Parameter

▷ **T1** (input_control) . . . . . string(-array) ⇝ *string*
Eingabetupel.

▷ **Length** (output_control) . . . . . integer(-array) ⇝ *integer*
Länge der Strings des Eingabetupels.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_strstr, tuple_strrstr, tuple_strchr, tuple_strrchr, tuple_substr, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Siehe auch

tuple_is_string

Modul

Foundation

tuple_strrchr (: : String, ToFind : Position)

Rückwärtssuchen nach einem Zeichen in einem Tupel von Strings.

tuple_strrchr sucht in dem Eingabetupel String nach den in dem Eingabetupel ToFind definierten Zeichen. Beide Eingabetupel müssen durchwegs aus Strings bestehen. Anderenfalls bricht tuple_strrchr mit einem entsprechenden Fehler ab. Enthalten die Strings von ToFind mehr als ein Zeichen, so wird ausschließlich das erste Zeichen in einem String berücksichtigt und alle folgenden Zeichen ignoriert. Die Suche nach einem Zeichen geschieht rückwärts also vom Ende zum Anfang des zu untersuchenden Strings. Besteht String nur aus einem einzigen String, so werden in diesem alle in ToFind definierten Zeichen gesucht. Das Ausgabetupel besteht also in diesem Fall aus genauso vielen Elementen wie das Eingabetupel ToFind. Ist die Suche nach einem Zeichen erfolgreich, so wird die Position (beginnend mit 0 für das erste Zeichen eines Strings), an der das Zeichen das erste Mal innerhalb des untersuchten Strings gefunden wurde, in dem Ausgabetupel Position zurückgegeben. Wurde ein Zeichen nicht gefunden, so wird für dessen Position der Wert -1 zurückgegeben. Enthalten beide Eingabetupel String und ToFind gleich viele Elemente, so werden sie elementweise bearbeitet, das heißt, das erste Zeichen im ersten Element von ToFind wird in dem ersten String von String, das erste Zeichen des zweiten ELements von ToFind wird im zweiten String von String gesucht und so weiter. Die einzelnen Elemente des Ausgabetupels enthalten dann das Resultat der elementweisen Suche. Besteht ToFind nur aus einem Element, so wird das dort definierte Zeichen in allen Strings des Eingabetupels String gesucht. Das Ausgabetupel Position besitzt dann also genauso viele Elemente wie das Eingabetupel String. Weisen die beiden Eingabetupel String und ToFind eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_strrchr mit einem entsprechenden Fehler ab.

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

Unicode Codepoints versus Bytes

Die Position bezieht sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Position sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit set_system('tsp_tuple_string_operator_mode','byte') in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_strrchr, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Position := strrchr(String, ToFind)
```

Parameter

- ▷ **String** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **ToFind** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit zu suchenden Zeichen.
- ▷ **Position** (output_control) ..... integer(-array) ⇝ *integer*
  Positionen der gesuchten Zeichen in den untersuchten Strings.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_strchr, tuple_strstr, tuple_strrstr, tuple_strlen, tuple_substr, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Modul

Foundation

tuple_strrstr (: : String, ToFind : Position)

Rückwärtssuchen nach einem Teilstring in einem Tupel von Strings.

tuple_strrstr sucht in dem Eingabetupel String nach den in dem Eingabetupel ToFind definierten Strings. Beide Eingabetupel müssen durchgehend aus Strings bestehen. Anderenfalls bricht tuple_strrstr mit einem entsprechenden Fehler ab. Besteht String nur aus einem einzigen String, so werden in diesem alle in ToFind definierten Strings gesucht. Das Ausgabetupel besteht also in diesem Fall aus genauso vielen Elementen wie das Eingabetupel ToFind. Die Suche geschieht rückwärts also vom Ende zum Anfang des zu untersuchenden Strings. Ist die Suche nach einem String erfolgreich, so wird die Position, an der der gesuchte String innerhalb des untersuchten Strings das erste Mal gefunden wurde, in dem Ausgabetupel Position zurückgegeben (die Position innerhalb eines Strings wird beginnend mit 0 gezählt). Wurde ein String nicht gefunden, so wird für dessen Position der Wert -1 zurückgegeben. Enthalten beide Eingabetupel String und ToFind gleich viele Elemente, so werden sie elementweise bearbeitet, das heißt, der erste String von ToFind wird in dem ersten String von String, der zweite von ToFind im zweiten von String gesucht und so weiter. Die einzelnen Elemente des Ausgabetupels enthalten dann das Resultat der elementweisen Suche. Besteht ToFind nur aus einem String, so wird dieser String in allen Strings des Eingabetupels String gesucht. Das Ausgabetupel Position besitzt dann also genauso viele Elemente wie das Eingabetupel String. Weisen die beiden Eingabetupel String und ToFind eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_strrstr mit einem entsprechenden Fehler ab.

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

Unicode Codepoints versus Bytes

Die Position bezieht sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Position sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit set_system('tsp_tuple_string_operator_mode','byte') in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_strrstr, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Position := strrstr(String, ToFind)
```

Parameter

- ▷ **String** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **ToFind** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit zu suchenden Teilstrings.
- ▷ **Position** (output_control) ..... integer(-array) ⇝ *integer*
  Positionen der gesuchten Strings in den untersuchten Strings.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_strstr, tuple_strlen, tuple_strchr, tuple_strrchr, tuple_substr, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Modul

Foundation

tuple_strstr (: : String, ToFind : Position)

Vorwärtssuchen nach einem Teilstring in einem Tupel von Strings.

tuple_strstr sucht in dem Eingabetupel String nach den in dem Eingabetupel ToFind definierten Strings. Beide Eingabetupel müssen durchgehend aus Strings bestehen. Anderenfalls bricht tuple_strstr mit einem entsprechenden Fehler ab. Besteht String nur aus einem einzigen String, so werden in diesem alle in ToFind definierten Strings gesucht. Das Ausgabetupel besteht also in diesem Fall aus genauso vielen Elementen wie das Eingabetupel ToFind. Ist die Suche nach einem String erfolgreich, so wird die Position, an der der gesuchte String innerhalb des untersuchten Strings das erste Mal gefunden wurde, in dem Ausgabetupel Position zurückgegeben (die Position wird beginnend mit 0 gezählt). Wurde ein String nicht gefunden, so wird für dessen Position der Wert -1 zurückgegeben. Enthalten beide Eingabetupel String und ToFind gleich viele Elemente, so werden sie elementweise bearbeitet, das heißt, der erste String von ToFind wird in dem ersten String von String, der zweite von ToFind im zweiten von String gesucht und so weiter. Die einzelnen Elemente des Ausgabetupels enthalten dann das Resultat der elementweisen Suche. Besteht ToFind nur aus einem String, so wird dieser String in allen Strings des Eingabetupels String gesucht. Das Ausgabetupel Position besitzt dann also genauso viele Elemente wie das Eingabetupel String. Weisen die beiden Eingabetupel String und ToFind eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_strstr mit einem entsprechenden Fehler ab.

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

Unicode Codepoints versus Bytes

Die Position bezieht sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Position sich auf die Bytes des Strings beziehen soll, kann dieser Operator mit set_system('tsp_tuple_string_operator_mode','byte') in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_strstr, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Position := strstr(String, ToFind)
```

Parameter

- ▷ **String** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **ToFind** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit zu suchenden Teilstrings.
- ▷ **Position** (output_control) ..... integer(-array) ⇝ *integer*
  Positionen der gesuchten Strings in den untersuchten Strings.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_strrstr, tuple_strlen, tuple_strchr, tuple_strrchr, tuple_substr, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Modul

Foundation

tuple_substr (: : String, Position1, Position2 : Substring)

Ausschneiden von Zeichen ab Position „n1“ bis einschließlich „n2“ aus einem Stringtupel.

tuple_substr schneidet aus jedem String des Eingabetupels String alle Zeichen beginnend mit dem Zeichen an der Position „n1“ bis einschließlich zur Position „n2“ aus und liefert sie als einen neuen String in dem Ausgabetupel Substring zurück. Die Zahlen „n1“ und „n2“ werden hierbei durch das zweite und dritte Eingabetupel Position1 und Position2 definiert, das jeweils die gleiche Länge haben muss. Enthalten Position1 und Position2 nur jeweils ein Element, so legt dieses Element die Zahlen „n1“ und „n2“ für alle Strings von String fest. Beinhalten Position1 und Position2 genauso viele Elemente wie String, so legen die jeweils ersten Elemente von Position1 und Position2 die Zahlen „n1“ und „n2“ für den ersten String von String fest. Die zweiten Elemente von Position1 und Position2 legen dann die Zahlen „n1“ und „n2“ für den zweiten String von String fest und so weiter. Enthalten Position1 und Position2 mehr als ein Element und besteht String nur aus einem String, so werden aus diesem String mehrere Segmente ausgeschnitten, deren Länge durch die Elemente von Position1 und Position2 festgelegt sind. Beinhalten alle Eingabetupel mehr als ein Element und unterscheiden sie sich gleichzeitig in der Anzahl der Elemente, so bricht tuple_substr mit einer Fehlermeldung ab.

Falls String ein leeres Tupel ist, gibt der Operator ein leeres Tupel zurück. Falls String nicht leer ist und Position1 und/oder Position2 leere Tupel sind, wird eine Fehlermeldung angezeigt.

Unicode Codepoints versus Bytes

Die Positionen beziehen sich auf Unicode Codepoints. Ein Codepoint kann aus mehreren Bytes im UTF-8 String zusammengesetzt sein. Wenn die Positionen sich auf die Bytes des Strings beziehen sollen, kann dieser Operator mit set_system('tsp_tuple_string_operator_mode','byte') in einen Byte-Modus umgeschaltet werden. Wenn *'filename_encoding'* auf *'locale'* gesetzt ist (legacy), verwendet dieser Operator immer den Byte-Modus.

Für allgemeine Informationen zu String-Operationen siehe auch Tupel / String-Operationen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_substr, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Substring := String{Position1:Position2}
```

Parameter

- ▷ **String** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit zu untersuchenden Strings.
- ▷ **Position1** (input_control) ..... number(-array) ⇝ *integer* / real
  Eingabetupel mit den Start-Positionen „n1“.
- ▷ **Position2** (input_control) ..... number(-array) ⇝ *integer* / real
  Eingabetupel mit den End-Positionen „n2“.
- ▷ **Substring** (output_control) ..... string(-array) ⇝ *string*
  Alle Zeichen von den Positionen „n1“ bis „n2“ für alle Eingabestrings.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_str_first_n, tuple_str_last_n, tuple_strstr, tuple_strrstr, tuple_strlen, tuple_strchr, tuple_strrchr, tuple_split, tuple_environment

Modul

Foundation

28.13 Typ

tuple_is_handle (: : T : IsHandle)

Testen, ob die interne Repräsentation eines Tupels vom Typ `handle` *ist.*

`tuple_is_handle` testet, ob die interne Repräsentation des Eingabetupels `T` vom Typ `handle` ist. In diesem Fall wird *1* (true) zurückgegeben, ansonsten wird *0* (false) zurückgegeben. Falls ein Tupel nur aus Elementen des Typs `handle` besteht, kann `IsHandle` trotzdem gleich *0* sein, falls die interne Repräsentation des Tupels H_TYPE_MIXED ist; siehe `tuple_type` für weitere Details.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator *1* zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_is_handle`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsHandle := is_handle(T)
```

Achtung

Auch falls alle Elemente des Tupels vom Typ `handle` sind, liefert `tuple_is_handle` *0* (false) zurück, falls die interne Repräsentation des Tupels vom Typ H_TYPE_MIXED ist. Um zu testen, ob die Elemente des Tupels vom Typ `handle` sind, sollte der Operator `tuple_is_handle_elem` verwendet werden.

Parameter

- ▷ **T** (input_control) . . . . . tuple(-array) ⇝ *handle* / real / integer / string
  Eingabetupel.
- ▷ **IsHandle** (output_control) . . . . . number ⇝ *integer*
  Boolescher Wert, der angibt, ob das Eingabetupel vom Typ `handle` ist.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_handle` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_type`, `tuple_is_number`, `tuple_is_handle_elem`, `tuple_is_string`, `tuple_sem_type`

Siehe auch

`tuple_is_int`, `tuple_is_real`, `tuple_is_string`

Modul

Foundation

tuple_is_handle_elem (: : T : IsHandle)

Testen, ob die Elemente eines Tupels vom Typ `handle` *sind.*

`tuple_is_handle_elem` testet die Elemente des Eingabetupels `T`. Für alle Elemente vom Typ `handle` wird *1* (true) zurückgegeben, ansonsten wird *0* (false) zurückgegeben.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_is_handle_elem`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsHandle := is_handle_elem(T)
```

Parameter

▷ **T** (input_control) . . . . . tuple(-array) ⇝ *handle* / real / integer / string
Eingabetupel.

▷ **IsHandle** (output_control) . . . . . number(-array) ⇝ *integer*
Boolesche Werte, die angeben, ob die Elemente des Eingabetupels vom Typ `handle` sind.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_handle_elem` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_type`, `tuple_type_elem`, `tuple_is_handle`

Siehe auch

`tuple_is_mixed`, `tuple_is_int`, `tuple_is_real`, `tuple_is_string`, `tuple_is_mixed`, `tuple_is_int_elem`, `tuple_is_real_elem`, `tuple_is_number`, `tuple_is_handle`

Modul

Foundation

tuple_is_int (: : T : IsInt)

Testen, ob die interne Repräsentation eines Tupels vom Typ `integer` (Ganzzahl) ist.

`tuple_is_int` testet, ob die interne Repräsentation des Eingabetupels `T` vom Typ `integer` (Ganzzahl) ist. In diesem Fall wird 1 (true) zurückgegeben, ansonsten wird 0 (false) zurückgegeben. Falls ein Tupel nur aus Elementen des Typs `integer` besteht, kann `IsInt` trotzdem gleich 0 sein, falls die interne Repräsentation des Tupels H_TYPE_MIXED ist; siehe `tuple_type` für weitere Details.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator 1 zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_is_int`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsInt := is_int(T)
```

Achtung

Auch falls alle Elemente des Tupels vom Typ `integer` sind, liefert `tuple_is_int` 0 (false) zurück, falls die interne Repräsentation des Tupels vom Typ H_TYPE_MIXED ist. Um zu testen, ob die Elemente des Tupels vom Typ `integer` sind, sollte der Operator `tuple_is_int_elem` verwendet werden.

Parameter

▷ **T** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string / handle
Eingabetupel.

▷ **IsInt** (output_control) . . . . . number ⇝ *integer*
Ist das Eingabetupel vom Typ `integer` (Ganzzahl)?

Beispiel

```
tuple_is_int ([3.1416,'pi',3], IsIntA)
* IsIntA = false
tuple_is_int ([1,2,3], IsIntB)
* IsIntB = true
tuple_is_int ([], IsIntC)
* IsIntC = true
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_int` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_type`, `tuple_is_number`, `tuple_is_int_elem`

Siehe auch

`tuple_is_real`, `tuple_is_string`, `tuple_is_handle`

Modul

Foundation

tuple_is_int_elem (: : T : IsInt)

Testen, ob die Elemente eines Tupels vom Typ `integer` (Ganzzahl) sind.

`tuple_is_int_elem` testet die Elemente des Eingabetupels `T`. Für alle Elemente vom Typ `integer` (Ganzzahl) wird 1 (true) zurückgegeben, ansonsten wird 0 (false) zurückgegeben.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_is_int_elem`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsInt := is_int_elem(T)
```

Achtung

`tuple_is_int_elem` gibt `true` zurück, falls der interne Datentyp der Elemente des Tupels `integer` ist. Im Gegensatz zu `tuple_is_number` wird nicht zurückgegeben, ob die Elemente eines Tupel als `integer` dargestellt werden könnten.

Parameter

▷ **T** (input_control) ..... tuple(-array) ⇝ *integer* / real / string / handle
Eingabetupel.

▷ **IsInt** (output_control) ..... number(-array) ⇝ *integer*
Sind die Elemente des Eingabetupels vom Typ `integer` (Ganzzahl)?

Beispiel

```
tuple_is_int_elem ([3.1416,'pi',3,3.0], IsInt)
* IsInt = [false, false, true, false]
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_int_elem` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_type, tuple_type_elem, tuple_is_int

Siehe auch

tuple_is_mixed, tuple_is_int, tuple_is_real, tuple_is_string, tuple_is_mixed, tuple_is_real_elem, tuple_is_string_elem, tuple_is_number, tuple_is_handle_elem, tuple_is_handle

Modul

Foundation

tuple_is_mixed (: : T : IsMixed)

Testen, ob ein Tupel vom Typ mixed (gemischt) ist.

tuple_is_mixed testet das Eingabetupel T. Ist das Tupel vom Typ mixed (gemischt) wird 1 (true) zurückgegeben, ansonsten wird 0 (false) zurückgegeben.

Wenn der Typ des Tupels T mixed ist, aber erwartet wird, dass alle Elemente des Tupels von einem bestimmten Datentyp sind, kann das Tupel explizit in ein besser passendes Format konvertiert werden. Dazu können die Operatoren tuple_int, tuple_real oder tuple_string mit Format='s' verwendet werden. Dies erhöht die Verarbeitungsgeschwindigkeit des konvertierten Tupels und lohnt sich insbesondere, falls das Tupel danach mehrmals verwendet wird.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator 1 zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_is_mixed, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsMixed := is_mixed(T)
```

Parameter

▷ **T** (input_control) ..... tuple(-array) ⇝ *string* / real / integer / handle
Eingabetupel.

▷ **IsMixed** (output_control) ..... number ⇝ *integer*
Ist das Eingabetupel vom Typ mixed (gemischt)?

Beispiel

```
tuple_is_mixed ([3.1416,'pi',3], IsMixedA)
* IsMixedA = true
tuple_is_mixed (['a','b','111'], IsMixedB)
* IsMixedB = false
tuple_is_mixed ([], IsMixedC)
* IsMixedC = true
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert tuple_is_mixed den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_type, tuple_is_number

Siehe auch

tuple_is_int, tuple_is_real

Modul

Foundation

tuple_is_real (: : T : IsReal)

Testen, ob die interne Repräsentation eines Tupels vom Typ real (Gleitkommazahl) ist.

tuple_is_real testet, ob die interne Repräsentation des Eingabetupels T vom Typ real (Gleitkommazahl) ist. In diesem Fall wird 1 (true) zurückgegeben, ansonsten wird 0 (false) zurückgegeben. Falls ein Tupel nur aus Elementen des Typs real besteht, kann IsReal trotzdem gleich 0 sein, falls die interne Repräsentation des Tupels H_TYPE_MIXED ist; siehe tuple_type für weitere Details.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator 1 zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_is_real, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsReal := is_real(T)
```

Achtung

Auch falls alle Elemente des Tupels vom Typ real sind, liefert tuple_is_real 0 (false) zurück, falls die interne Repräsentation des Tupels vom Typ H_TYPE_MIXED ist. Um zu testen, ob die Elemente des Tupels vom Typ real sind, sollte der Operator tuple_is_real_elem verwendet werden.

Parameter

- ▷ **T** (input_control) ..... tuple(-array) ⇝ *real / integer / string / handle*
  Eingabetupel.
- ▷ **IsReal** (output_control) ..... number ⇝ *integer*
  Ist das Eingabetupel vom Typ real (Gleitkommazahl)?

Beispiel

```
tuple_is_real ([3.1416,'pi',3], IsRealA)
* IsRealA = false
tuple_is_real ([1.0,2.0,3.0], IsRealB)
* IsRealB = true
tuple_is_real ([], IsRealC)
* IsRealC = true
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert tuple_is_real den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_type, tuple_is_number, tuple_is_real_elem

Siehe auch

tuple_is_int, tuple_is_string, tuple_is_handle

Modul

Foundation

tuple_is_real_elem (: : T : IsReal)

Testen, ob die Elemente eines Tupels vom Typ `real` (Gleitkommazahl) sind.

`tuple_is_real_elem` testet die Elemente des Eingabetupels `T`. Für alle Elemente vom Typ `real` (Gleitkommazahl) wird 1 (true) zurückgegeben, ansonsten wird 0 (false) zurückgegeben.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_is_real_elem`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsReal := is_real_elem(T)
```

Achtung

`tuple_is_real_elem` gibt `true` zurück, falls der interne Datentyp der Elemente des Tupels `real` ist. Im Gegensatz zu `tuple_is_number` wird nicht zurückgegeben, ob die Elemente eines Tupel als `real` dargestellt werden könnten.

Parameter

- ▷ **T** (input_control) ..... tuple(-array) ⇝ *real / integer / string / handle*
  Eingabetupel.
- ▷ **IsReal** (output_control) ..... number(-array) ⇝ *integer*
  Sind die Elemente des Eingabetupels vom Typ `real` (Gleitkommazahl)?

Beispiel

```
tuple_is_real_elem ([3.1416,'pi',3,3.0], IsReal)
* IsReal = [true, false, false, true]
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_real_elem` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_type`, `tuple_type_elem`, `tuple_is_real`

Siehe auch

`tuple_is_mixed`, `tuple_is_int`, `tuple_is_real`, `tuple_is_string`, `tuple_is_mixed`, `tuple_is_int_elem`, `tuple_is_string_elem`, `tuple_is_number`, `tuple_is_handle_elem`, `tuple_is_handle`

Modul

Foundation

tuple_is_string (: : T : IsString)

Testen, ob die interne Repräsentation eines Tupels vom Typ `string` (Zeichenkette) ist.

`tuple_is_string` testet, ob die interne Repräsentation des Eingabetupels `T` vom Typ `string` (Zeichenkette) ist. In diesem Fall wird 1 (true) zurückgegeben, ansonsten wird 0 (false) zurückgegeben. Falls ein Tupel nur aus Elementen des Typs `string` besteht, kann `IsString` trotzdem gleich 0 sein, falls die interne Repräsentation des Tupels H_TYPE_MIXED ist; siehe `tuple_type` für weitere Details.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator 1 zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_is_string`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsString := is_string(T)
```

Achtung

Auch falls alle Elemente des Tupels vom Typ `string` sind, liefert `tuple_is_string` 0 (false) zurück, falls die interne Repräsentation des Tupels vom Typ H_TYPE_MIXED ist. Um zu testen, ob die Elemente des Tupels vom Typ `string` sind, sollte der Operator `tuple_is_string_elem` verwendet werden.

Parameter

- ▷ **T** (input_control) ..... tuple(-array) ⇝ *string* / real / integer / handle
  Eingabetupel.
- ▷ **IsString** (output_control) ..... number ⇝ *integer*
  Ist das Eingabetupel vom Typ `string` (Zeichenkette)?

Beispiel

```
tuple_is_string ([3.1416,'pi',3], IsStringA)
* IsStringA = false
tuple_is_string (['a','b','111'], IsStringB)
* IsStringB = true
tuple_is_string ([], IsStringC)
* IsStringC = true
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_string` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_type`, `tuple_is_number`, `tuple_is_string_elem`

Siehe auch

`tuple_is_int`, `tuple_is_real`, `tuple_is_handle`

Modul

Foundation

tuple_is_string_elem (: : T : IsString)

Testen, ob die Elemente eines Tupels vom Typ `string` (Zeichenkette) sind.

`tuple_is_string_elem` testet die Elemente des Eingabetupels `T`. Für alle Elemente vom Typ `string` (Zeichenkette) wird 1 (true) zurückgegeben, ansonsten wird 0 (false) zurückgegeben.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_is_string_elem`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
IsString := is_string_elem(T)
```

Parameter

▷ **T** (input_control) . . . . . tuple(-array) ⇝ *string* / real / integer / handle
Eingabetupel.

▷ **IsString** (output_control) . . . . . number(-array) ⇝ *integer*
Sind die Elemente des Eingabetupels vom Typ `string` (Zeichenkette)?

Beispiel

```
tuple_is_string_elem ([3.1416,'pi',3], IsString)
* IsString = [false, true, false]
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_string_elem` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_type`, `tuple_type_elem`, `tuple_is_string`

Siehe auch

`tuple_is_mixed`, `tuple_is_int`, `tuple_is_real`, `tuple_is_string`, `tuple_is_mixed`, `tuple_is_int_elem`, `tuple_is_real_elem`, `tuple_is_number`, `tuple_is_handle_elem`, `tuple_is_handle`

Modul

Foundation

tuple_is_valid_handle (: : Handle : IsValid)

Prüft, ob ein Handle gültig ist.

`tuple_is_valid_handle` gibt *1* in `IsValid` zurück, wenn `Handle` ein gültiges, nicht freigegebenes Handle enthält, andernfalls *0*.

Mehrere Handles können in `Handle` übergeben werden. In diesem Fall enthält `IsValid` einen Gültigkeitswert für jedes Eingabehandle.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

Parameter

▷ **Handle** (input_control) . . . . . tuple(-array) ⇝ *handle*
Handle, das auf Gültigkeit geprüft wird.

▷ **IsValid** (output_control) . . . . . integer(-array) ⇝ *integer*
Gültigkeit des Eingabehandles.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_is_valid_handle` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

tuple_sem_type (: : T : SemType)

Zurückgeben den semantischen Typs eines Tupels.

tuple_sem_type gibt den semantischen Typ des Eingabetupels T zurück. Der Typ wird als String in SemType zurückgegeben.

Falls T nur gültige Handles des gleichen Typs enthält, wird dessen semantischer Typ zurückgegeben (z.B. *'matrix'*). Falls das Eingabetupel nur gültige und ungültige Handles oder Handles verschiedenen Typs enthält, wird *'handle'* zurückgegeben. Falls T Elemente verschiedenen Typs enthält, wird *'any'* zurückgegeben.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator den Wert *'any'* zurück.

Parameter

- ▷ **T** (input_control) . . . . . tuple(-array) ⇝ *handle* / real / integer / string
  Eingabetupel.
- ▷ **SemType** (output_control) . . . . . string ⇝ *string*
  Semantischer Typ des Eingabetupels als String.

Beispiel

```
create_matrix (3, 3, 0, MatrixID)
tuple_sem_type (MatrixID, SemType)
* SemType == 'matrix' (Handle of specific type)
clear_handle (MatrixID)
tuple_sem_type (MatrixID, SemType)
* SemType == 'handle' (Cleared handle)
tuple_sem_type ([MatrixID,123], SemType)
* SemType == 'any' (Mixed types)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert tuple_sem_type den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_is_int, tuple_is_number, tuple_is_real, tuple_is_string, tuple_type

Siehe auch

get_obj_class

Modul

Foundation

tuple_sem_type_elem (: : T : SemTypes)

Zurückgeben der semantischen Typen der Elemente eines Tupels.

tuple_sem_type_elem gibt die semantischen Typen der Elemente des Eingabetupels T zurück. Die Typen werden als String in SemTypes zurückgegeben.

Falls ein Element ein gültiges Handle ist, wird der semantische Typ des Handles zurückgegeben. Für freigegebene oder ungültige Handles wird *'handle'* zurückgegeben.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

Parameter

▷ **T** (input_control) . . . . . tuple(-array) ⇝ *handle* / real / integer / string
Eingabetupel.

▷ **SemTypes** (output_control) . . . . . string(-array) ⇝ *string*
Semantische Typen der Elemente des Eingabetupels als Strings.

Beispiel

```
create_matrix (3, 3, 0, MatrixID)
tuple_sem_type_elem ([MatrixID,1,1.0,'HALCON',HNULL], SemTypeElem)
* SemTypeElem == ['matrix','integer','real','string','handle']
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_sem_type_elem` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_type`, `tuple_is_mixed`, `tuple_is_int`, `tuple_is_real`, `tuple_is_string`, `tuple_is_mixed`, `tuple_is_int_elem`, `tuple_is_real_elem`, `tuple_is_string_elem`, `tuple_type_elem`, `tuple_sem_type`

Siehe auch

`get_obj_class`

Modul

Foundation

tuple_type (: : T : Type)

Zurückgeben des Typs eines Tupels.

`tuple_type` gibt den Typ eines Eingabetupels `T` zurück. Der Typ wird als Ganzzahl im Ausgabeparameter `Type` zurückgegeben. In HDevelop stehen entsprechend definierte Konstanten zur Verfügung:

- H_TYPE_INT (1).
- H_TYPE_REAL (2).
- H_TYPE_STRING (4).
- H_TYPE_MIXED (8).
- H_TYPE_HANDLE (16).
- H_TYPE_ANY (31).

H_TYPE_MIXED wird in folgenden zwei Fällen zurückgegeben:

- Elemente des Tupels haben verschiedene Datentypen, z.B. `real` und `integer`.
- Mit dem Tupel `T` wurden Operationen durchgeführt, die den Datentyp einzelner Elemente geändert haben. Solche Tupel werden im Allgemeinen (aufgrund von Laufzeitgründen) nicht automatisch optimiert, selbst wenn die Datentypen aller Elemente in nachfolgenden Operationen wieder gleich werden. Siehe Operator `tuple_is_mixed` und das nachfolgende Beispiel für Informationen darüber wie solche Tupel optimiert werden können.

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator 31 (H_TYPE_ANY) zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_type`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Type := type(T)
```

Achtung

`tuple_type` gibt den internen Datentyp des Tupels zurück. Im Gegensatz zu `tuple_is_number` wird nicht zurückgegeben, ob ein Tupel als ein bestimmter Datentyp dargestellt werden könnte.

Parameter

- ▷ **T** (input_control) . . . . . tuple(-array) ⇝ *real* / integer / string / handle
  Eingabetupel.
- ▷ **Type** (output_control) . . . . . number ⇝ *integer*
  Typ des Eingabetupels als Ganzzahl.

Beispiel

```
tuple_type ([3.1416,'pi',3], TypeA)
* TypeA = H_TYPE_MIXED
tuple_type (['a','b','111'], TypeB)
* TypeB = H_TYPE_STRING
tuple_type ([], TypeC)
* TypeC = H_TYPE_ANY
tuple_type (HNULL, TypeD)
* TypeD = H_TYPE_HANDLE

TupleInt := [1,2,3,4]
TupleReal := [42.0]
TupleConcat := [TupleInt, TupleReal]
tuple_type (TupleConcat, TypeConcat)
* TypeConcat = H_TYPE_MIXED
* Now set 42.0 to 42
TupleConcat[4] := 42
tuple_type (TupleConcat, TypeConcat2)
* TypeConcat2 = H_TYPE_MIXED
* TupleConcat now consists of integers only, but the
* internal representation hasn't been updated. Optimize
* it by converting the tuple explicitly to an integer
* tuple.
tuple_int (TupleConcat, TupleConcatInt)
tuple_type (TupleConcatInt, TypeConcatInt)
* TypeConcatInt = H_TYPE_INT
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `tuple_type` den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_is_int`, `tuple_is_number`, `tuple_is_real`, `tuple_is_string`, `tuple_sem_type`

Siehe auch

`get_obj_class`

Modul

Foundation

tuple_type_elem (: : T : Types)

Zurückgeben des Typs der Elemente eines Tupels.

tuple_type_elem gibt den Typ der Elemente des Eingabetupels T zurück. Die Typen werden als Ganzzahl im Ausgabeparameter Types zurückgegeben. In HDevelop stehen entsprechend definierte Konstanten zur Verfügung:

- H_TYPE_INT (1).
- H_TYPE_REAL (2).
- H_TYPE_STRING (4).
- H_TYPE_HANDLE (16).

Sonderfall: Leeres Eingabetupel

Falls das Eingabetupel leer ist, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_type_elem, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Types := type_elem(T)
```

Parameter

▷ **T** (input_control) . . . . . tuple(-array) ⇝ *real* / integer / string / handle
Eingabetupel.

▷ **Types** (output_control) . . . . . number(-array) ⇝ *integer*
Typen der Elemente des Eingabetupels als Ganzzahl.

Beispiel

```
tuple_type_elem ([3.1416,'pi',3], Types)
* Types = [H_TYPE_REAL, H_TYPE_STRING, H_TYPE_INT]
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert tuple_type_elem den Wert TRUE.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_type, tuple_is_mixed, tuple_is_int, tuple_is_real, tuple_is_string, tuple_is_mixed, tuple_is_int_elem, tuple_is_real_elem, tuple_is_string_elem

Siehe auch

get_obj_class, tuple_is_number

Modul

Foundation

28.14 Vergleich

tuple_equal (: : T1, T2 : Equal)

Testen, ob zwei Tupel gleich sind.

tuple_equal testet, ob die beiden Eingabetupel T1 und T2 gleich sind. Die Eingabetupel werden elementweise verglichen. Zwei Tupelelemente gelten als gleich, wenn sie beide (ganze oder Gleitkomma-)Zahlen oder beide

Strings sind und dieselbe Belegung aufweisen. Darüber hinaus müssen beide Eingabetupel über dieselbe Anzahl an Elementen verfügen, um als gleich zu gelten.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_equal`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

`Equal := T1 == T2` oder `Equal := T1 = T2`

Parameter

- ▷ **`T1`** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **`T2`** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **`Equal`** (output_control) . . . . . integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_not_equal`, `tuple_less`, `tuple_greater`, `tuple_less_equal`, `tuple_greater_equal`, `tuple_equal_elem`, `tuple_not_equal_elem`, `tuple_less_elem`, `tuple_greater_elem`, `tuple_less_equal_elem`, `tuple_greater_equal_elem`

Modul

Foundation

`tuple_equal_elem` (: : T1, T2 : Equal)

Test, ob zwei Tupel elementweise gleich sind.

`tuple_equal_elem` testet, ob die beiden Eingabetupel `T1` und `T2` elementweise gleich sind. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel miteinander verglichen. Ansonsten muss entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels verglichen. Zwei Tupelelemente gelten als gleich, falls sie beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sind und dieselbe Belegung (Werte) aufweisen.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_equal_elem`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

`Equal := T1 [==] T2` oder `Equal := T1 [=] T2`

Parameter

- ▷ **`T1`** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **`T2`** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **`Equal`** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_not_equal_elem, tuple_less_elem, tuple_greater_elem, tuple_less_equal_elem, tuple_greater_equal_elem, tuple_equal, tuple_not_equal, tuple_less, tuple_greater, tuple_less_equal, tuple_greater_equal

Modul

Foundation

tuple_greater (: : T1, T2 : Greater)

Testen, ob ein Tupel größer als ein zweites Tupel ist.

tuple_greater gibt an, ob das Eingabetupel T1 größer als das Eingabetupel T2 ist. Ein Tupel T1 gilt als größer gegenüber einem Tupel T2, falls es sich entweder in einem elementweisen Vergleich als größer erweist oder falls T1 - im Falle, dass die ersten $min(|T1|, |T2|)$ Elemente gleich sind - mehr Elemente als T2 besitzt.

Beim elementweisen Vergleich werden die Elemente einzeln verglichen. Sind die beiden Elemente gleich, so wird das nächste Element geprüft. Ist das Element aus T1 größer als das aus T2, so ist das Ergebnis 1 und der Suchlauf wird abgebrochen. Ist das Element aus T1 kleiner als das aus T2, so ist das Ergebnis 0 und der Suchlauf wird abgebrochen. Sind alle Elemente durchlaufen und alle Elemente waren gleich so ist das Ergebnis 0.

Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht tuple_greater mit einer entsprechenden Fehlermeldung ab.

Sonderfall: Leeres Eingabetupel

Falls eines der Eingabetupel leer ist und das andere nicht, wird das Tupel, dass nicht leer ist, als größer als das leere Tupel interpretiert.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_greater, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Greater := T1 > T2
```

Parameter

- ▷ **T1** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Greater** (output_control) ..... integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_greater_equal, tuple_less, tuple_less_equal, tuple_equal, tuple_not_equal, tuple_equal_elem, tuple_not_equal_elem, tuple_less_elem, tuple_greater_elem, tuple_less_equal_elem, tuple_greater_equal_elem

Modul

Foundation

tuple_greater_elem (: : T1, T2 : Greater)

Test, ob ein Tupel elementweise größer als ein zweites Tupel ist.

tuple_greater_elem testet elementweise, ob das Eingabetupel T1 größer als das Eingabetupel T2 ist. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel miteinander verglichen. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels verglichen. Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht tuple_greater_elem mit einer entsprechenden Fehlermeldung ab.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_greater_elem, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Greater := T1 [>] T2
```

Parameter

- ▷ **T1** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Greater** (output_control) ..... integer(-array) ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_equal_elem, tuple_not_equal_elem, tuple_less_elem, tuple_less_equal_elem, tuple_greater_equal_elem, tuple_equal, tuple_not_equal, tuple_less, tuple_greater, tuple_less_equal, tuple_greater_equal

Modul

Foundation

tuple_greater_equal (: : T1, T2 : Greatereq)

Testen, ob ein Tupel größer oder gleich einem zweiten Tupel ist.

tuple_greater_equal gibt an, ob das Eingabetupel T1 größer oder gleich dem Tupel T2 ist. Ein Tupel T1 gilt als größer oder gleich einem Tupel T2, falls es nicht kleiner als T2 ist. Ein Tupel T1 gilt als kleiner gegenüber einem Tupel T2, falls es sich entweder in einem elementweisen Vergleich als kleiner erweist oder falls T1 - im Falle, dass die ersten $min(|T1|, |T2|)$ Elemente gleich sind - weniger Elemente als T2 besitzt.

Beim elementweisen Vergleich werden die Elemente einzeln verglichen. Sind die beiden Elemente gleich, so wird das nächste Element geprüft. Ist das Element aus T1 größer als das aus T2, so ist das Ergebnis 1 und der Suchlauf wird abgebrochen. Ist das Element aus T1 kleiner als das aus T2, so ist das Ergebnis 0 und der Suchlauf wird abgebrochen. Sind alle Elemente durchlaufen und alle Elemente waren gleich so ist das Ergebnis 1.

Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht tuple_greater_equal mit einer entsprechenden Fehlermeldung ab.

Sonderfall: Leeres Eingabetupel

Falls eines der Eingabetupel leer ist und das andere nicht, wird das Tupel, dass nicht leer ist, als größer als das leere Tupel interpretiert.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_greater_equal`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Greatereq := T1 >= T2
```

Parameter

- ▷ **T1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Greatereq** (output_control) . . . . . integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_greater`, `tuple_less`, `tuple_less_equal`, `tuple_equal`, `tuple_not_equal`, `tuple_equal_elem`, `tuple_not_equal_elem`, `tuple_less_elem`, `tuple_greater_elem`, `tuple_less_equal_elem`, `tuple_greater_equal_elem`

Modul

Foundation

tuple_greater_equal_elem (: : T1, T2 : Greatereq)

Test, ob ein Tupel elementweise größer oder gleich einem zweiten Tupel ist.

`tuple_greater_equal_elem` testet elementweise, ob das Eingabetupel `T1` größer oder gleich dem Eingabetupel `T2` ist. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel miteinander verglichen. Ansonsten muss entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels verglichen. Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht `tuple_greater_equal_elem` mit einer entsprechenden Fehlermeldung ab.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_greater_equal_elem`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Greatereq := T1 [>=] T2
```

Parameter

- ▷ **T1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Greatereq** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_equal_elem, tuple_not_equal_elem, tuple_less_elem, tuple_greater_elem, tuple_less_equal_elem, tuple_equal, tuple_not_equal, tuple_less, tuple_greater, tuple_less_equal, tuple_greater_equal

Modul

Foundation

tuple_less (: : T1, T2 : Less)

Testen, ob ein Tupel kleiner als ein zweites Tupel ist.

tuple_less gibt an, ob das Eingabetupel T1 kleiner als das Eingabetupel T2 ist. Ein Tupel T1 gilt als kleiner gegenüber einem Tupel T2, falls es sich entweder in einem elementweisen Vergleich als kleiner erweist oder falls T1 - im Falle, dass die ersten $min(|T1|, |T2|)$ Elemente gleich sind - weniger Elemente als T2 besitzt.

Beim elementweisen Vergleich werden die Elemente einzeln verglichen. Sind die beiden Elemente gleich, so wird das nächste Element geprüft. Ist das Element aus T1 kleiner als das aus T2, so ist das Ergebnis 1 und der Suchlauf wird abgebrochen. Ist das Element aus T1 größer als das aus T2, so ist das Ergebnis 0 und der Suchlauf wird abgebrochen. Sind alle Elemente durchlaufen und alle Elemente waren gleich so ist das Ergebnis 0.

Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht tuple_less mit einer entsprechenden Fehlermeldung ab.

Sonderfall: Leeres Eingabetupel

Falls eines der Eingabetupel leer ist und das andere nicht, wird das Tupel, dass leer ist, als kleiner als das nicht-leere Tupel interpretiert.

Sonderfall: Leeres Eingabetupel

Falls eines der Eingabetupel leer ist und das andere nicht, wird das Tupel, dass leer ist, als kleiner als das nicht-leere Tupel interpretiert.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_less, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Less := T1 < T2
```

Parameter

- ▷ **T1** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Less** (output_control) ..... integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_less_equal, tuple_greater, tuple_greater_equal, tuple_equal, tuple_not_equal, tuple_equal_elem, tuple_not_equal_elem, tuple_less_elem, tuple_greater_elem, tuple_less_equal_elem, tuple_greater_equal_elem

Modul

Foundation

tuple_less_elem (: : T1, T2 : Less)

Test, ob ein Tupel elementweise kleiner als ein zweites Tupel ist.

tuple_less_elem testet elementweise, ob das Eingabetupel T1 kleiner als das Eingabetupel T2 ist. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel miteinander verglichen. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels verglichen. Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht tuple_less_elem mit einer entsprechenden Fehlermeldung ab.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_less_elem, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Less := T1 [<] T2
```

Parameter

- ▷ **T1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Less** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_equal_elem, tuple_not_equal_elem, tuple_greater_elem, tuple_less_equal_elem, tuple_greater_equal_elem, tuple_equal, tuple_not_equal, tuple_less, tuple_greater, tuple_less_equal, tuple_greater_equal

Modul

Foundation

tuple_less_equal (: : T1, T2 : Lesseq)

Testen, ob ein Tupel kleiner oder gleich einem zweiten Tupel ist.

tuple_less_equal gibt an, ob das Eingabetupel T1 kleiner oder gleich dem Tupel T2 ist. Ein Tupel T1 gilt als kleiner oder gleich einem Tupel T2, falls es nicht größer als T2 ist. Ein Tupel T1 gilt als größer gegenüber einem Tupel T2, falls es sich entweder in einem elementweisen Vergleich als größer erweist oder falls T1 - im Falle, dass die ersten $min(|T1|, |T2|)$ Elemente gleich sind - mehr Elemente als T2 besitzt.

Beim elementweisen Vergleich werden die Elemente einzeln verglichen. Sind die beiden Elemente gleich, so wird das nächste Element geprüft. Ist das Element aus `T1` kleiner als das aus `T2`, so ist das Ergebnis 1 und der Suchlauf wird abgebrochen. Ist das Element aus `T1` größer als das aus `T2`, so ist das Ergebnis 0 und der Suchlauf wird abgebrochen. Sind alle Elemente durchlaufen und alle Elemente waren gleich so ist das Ergebnis 1.

Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht `tuple_less_equal` mit einer entsprechenden Fehlermeldung ab.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_less_equal`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Lesseq := T1 <= T2
```

Parameter

- ▷ **`T1`** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **`T2`** (input_control) ..... tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **`Lesseq`** (output_control) ..... integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

`tuple_less`, `tuple_greater`, `tuple_greater_equal`, `tuple_equal`, `tuple_not_equal`, `tuple_equal_elem`, `tuple_not_equal_elem`, `tuple_less_elem`, `tuple_greater_elem`, `tuple_less_equal_elem`, `tuple_greater_equal_elem`

Modul

Foundation

`tuple_less_equal_elem` (: : T1, T2 : Lesseq)

Test, ob ein Tupel elementweise kleiner oder gleich einem zweiten Tupel ist.

`tuple_less_equal_elem` testet elementweise, ob das Eingabetupel `T1` kleiner oder gleich dem Eingabetupel `T2` ist. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel miteinander verglichen. Ansonsten muss entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels verglichen. Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht `tuple_less_equal_elem` mit einer entsprechenden Fehlermeldung ab.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für `tuple_less_equal_elem`, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

```
Lesseq := T1 [<=] T2
```

Parameter

- ▷ **T1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Lesseq** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_equal_elem, tuple_not_equal_elem, tuple_greater_elem, tuple_less_elem, tuple_greater_equal_elem, tuple_equal, tuple_not_equal, tuple_less, tuple_greater, tuple_less_equal, tuple_greater_equal

Modul

Foundation

tuple_not_equal (: : T1, T2 : Nequal)

Testen, ob sich zwei Tupel unterscheiden.

tuple_not_equal testet, ob die beiden Eingabetupel T1 und T2 verschieden sind. Zwei Tupel gelten als verschieden, sobald sie unterschiedlich viele Elemente beinhalten oder sich in mindestens einer Position beim elementweisen Vergleich unterscheiden. Zwei miteinander verglichene Tupelelemente gelten als verschieden, falls sie entweder aus nicht vergleichbaren Typen bestehen (zum Beispiel eine Gleitkommazahl und ein String), oder falls sie zwar vergleichbare Typen besitzen, jedoch eine unterschiedliche Belegung (Werte) aufweisen.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_not_equal, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

Nequal := T1 != T2 oder Nequal := T1 # T2

Parameter

- ▷ **T1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Nequal** (output_control) . . . . . integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_equal, tuple_less, tuple_greater, tuple_less_equal, tuple_greater_equal, tuple_equal_elem, tuple_not_equal_elem, tuple_less_elem, tuple_greater_elem, tuple_less_equal_elem, tuple_greater_equal_elem

Modul

Foundation

tuple_not_equal_elem (: : T1, T2 : Nequal)

Test, ob sich zwei Tupel elementweise unterscheiden.

tuple_not_equal_elem testet, ob die beiden Eingabetupel T1 und T2 elementweise verschieden sind. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel miteinander verglichen. Ansonsten muss entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels verglichen. Zwei Tupelelemente gelten als verschieden, falls sie entweder aus nicht vergleichbaren Typen bestehen (zum Beispiel eine Gleitkommazahl und ein String), oder falls sie zwar vergleichbare Typen besitzen, jedoch eine unterschiedliche Belegung (Werte) aufweisen.

Sonderfall: Leere Eingabetupel

Falls eines oder beide der Eingabetupel leer sind, gibt der Operator ein leeres Tupel zurück.

HDevelop Inline-Operation

HDevelop unterstützt eine Inline-Operation für tuple_not_equal_elem, die innerhalb eines Ausdrucks in der folgenden Syntax verwendet werden kann:

Nequal := T1 [!=] T2 oder Nequal := T1 [#] T2

Parameter

- ▷ **T1** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . tuple(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Nequal** (output_control) . . . . . integer(-array) ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Ausführungsinformationen

- Multithreading-Typ: independent (läuft parallel auch zu exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Alternativen

tuple_equal_elem, tuple_less_elem, tuple_greater_elem, tuple_less_equal_elem, tuple_greater_equal_elem, tuple_equal, tuple_not_equal, tuple_less, tuple_greater, tuple_less_equal, tuple_greater_equal

Modul

Foundation

Kapitel 29

XLD

29.1 Generierung

gen_circle_contour_xld (: ContCircle : Row, Column, Radius, StartPhi, EndPhi, PointOrder, Resolution :)

Erzeugen von XLD-Konturen in Form von Kreisen oder Kreisbögen.

gen_circle_contour_xld erzeugt einen oder mehrere Kreisbögen oder geschlossene Kreise. Kreise werden durch ihren Mittelpunkt (Row, Column) und ihren Radius beschrieben. Kreisbögen werden zusätzlich durch den Anfangs- und Endwinkel StartPhi und EndPhi des Start- bzw. Endpunktes, sowie den zugehörigen Umlaufsinn PointOrder charakterisiert. Die Auflösung der resultierenden XLD Konturen ContCircle wird über den Parameter Resolution festgelegt: Er bestimmt den euklidischen Abstand in Pixeln zwischen benachbarten Konturpunkten. Der Abstand zwischen dem vorletzten und dem letzten Konturpunkt ist im Allgemeinen kleiner als Resolution.

Parameter

- ▷ **ContCircle** (output_object) ..... xld_cont(-array) ⇝ *object*
  Ausgabe-Konturen.
- ▷ **Row** (input_control) ..... circle.center.y(-array) ⇝ *real*
  Zeilenkoordinaten der Mittelpunkte der Kreise oder Kreisbögen.
  Defaultwert: 200.0
- ▷ **Column** (input_control) ..... circle.center.x(-array) ⇝ *real*
  Spaltenkoordinaten der Mittelpunkte der Kreise oder Kreisbögen.
  Defaultwert: 200.0
- ▷ **Radius** (input_control) ..... circle.radius(-array) ⇝ *real*
  Radien der Kreise oder Kreisbögen.
  Defaultwert: 100.0
  Restriktion: Radius > 0
- ▷ **StartPhi** (input_control) ..... angle.rad(-array) ⇝ *real*
  Winkel der Startpunkte der Kreise oder Kreisbögen [rad].
  Defaultwert: 0.0
- ▷ **EndPhi** (input_control) ..... angle.rad(-array) ⇝ *real*
  Winkel der Endpunkte der Kreise oder Kreisbögen [rad].
  Defaultwert: 6.28318
- ▷ **PointOrder** (input_control) ..... string(-array) ⇝ *string*
  Mathematischer Umlaufsinn der Kreise oder Kreisbögen.
  Defaultwert: 'positive'
  Werteliste: PointOrder ∈ {'positive', 'negative'}
- ▷ **Resolution** (input_control) ..... real ⇝ *real*
  Abstand zwischen benachbarten Konturpunkten.
  Defaultwert: 1.0
  Restriktion: Resolution >= 0.00001

Beispiel

```
draw_circle (WindowHandle, Row, Column, Radius)
gen_circle_contour_xld (ContCircle, Row, Column, Radius, 0, \
                        rad(360) , 'positive', 1.0)
gen_region_contour_xld (ContCircle, Region, 'filled')
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_circle_contour_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`draw_circle`

Nachfolger

`disp_xld`

Alternativen

`gen_ellipse_contour_xld`

Modul

Foundation

gen_contour_nurbs_xld (: Contour : Rows, Cols, Knots, Weights, Degree, MaxError, MaxDistance :)

Erzeugen einer XLD Kontur anhand einer NURBS Kurve.

Der Operator `gen_contour_nurbs_xld` erzeugt eine XLD Kontur `Contour`, die eine NURBS-Kurve (nicht uniformer, rationaler B-Spline) approximiert. Eine NURBS-Kurve wird spezifiziert durch ihren Grad `Degree`, ihr Kontrollpolygon (`Rows`,`Cols`), ihren Gewichtsvektor `Weights` und ihren Knotenvektor `Knots`.

Der Grad `Degree` der Kurve bestimmt den Grad ihrer Differenzierbarkeit. Die Kurve liegt innerhalb der konvexen Hülle ihres Kontrollpolygons (`Rows`,`Cols`). Die Gewichte in `Weights` beschreiben, wie stark die Kurve von den jeweiligen Kontrollpunkten angezogen wird. Wird *'auto'* als Gewichtsvektor `Weights` ausgewählt, erhalten alle Kontrollpunkte das gleiche Gewicht. Der Knotenvektor `Knots` beschreibt den Träger der Kurve und kann im Normalfall auf *'auto'* gesetzt werden.

Die Genauigkeit der approximierenden Kontur `Contour` wird durch die Parameter `MaxDistance` und `MaxError` bestimmt. Ersterer ist die maximale Distanz von zwei aufeinanderfolgenden Punkten in der approximierenden Kontur `Contour`. Der zweite Parameter entspricht dem maximalen Abstand der Approximation `Contour` von der eigentlichen Kurve. Normalerweise muss die Approximation beide Bedingungen erfüllen, es ist aber möglich, durch Setzen des Wertes auf *'omit'* eine der beiden Bedingungen wegzulassen. Je niedriger diese beiden Parameter gewählt werden, desto besser ist die Approximation der Kurve. Unabhängig von der Wahl der Parameter liegen alle Punkte der Kontur `Contour` genau auf der Kurve (bis auf numerische Ungenauigkeiten).

Definition

Eine NURBS Kurve C von Grad p is definiert durch

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i}, u \in [u_p, u_n].$$

Die Basis Funktionen N_i, p sind gegeben durch

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{sonst} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

Eine NURBS-Kurve ist also definiert durch ihren Grad p (`Degree`), ihr Kontrollpolygon $P_0, ..., P_n - 1$ (`Rows` und `Cols`), ihren Gewichtsvektor $w = (w_0, ..., w_n - 1)$ (`Weights`) und ihren Knotenvektor $u = (u_0, ..., u_n + p)$ (`Knots`).

Der Grad der Differenzierbarkeit der Kurve wird bestimmt durch p. Mindestens $p + 1$ Kontrollpunkte werden benötigt, um eine gültige Kurve zu definieren. Der Knotenvektor u beschreibt den Träger der Kurve. Die Länge dieses Vektors muss genau $n + p + 1$ betragen und die Einträge müssen monoton steigend sein. Jeder Eintrag u_i entspricht dem Knoten zwischen den beiden Kurvensegmenten $[u_i - 1, u_i)$ und $[u_i, u_i + 1)$. Jedes Kurvensegment $[u_i, u_i + 1)$ liegt innerhalb der konvexen Hülle der Kontrollpunkte $P_i - p, ..., P_i$. Für jedes Intervall, dessen Länge null ist, sinkt der Grad der Differenzierbarkeit des Knotens k. Die Anzahl der Vorkommen eines Knotens im Knotenvektor wird als Multiplizität bezeichnet. Die maximale Multiplizität eines Knotens ist p. Ausnahme sind Start- und Endpunkt, deren Multiplizität $p + 1$ betragen kann. Das ist auch die Regel, da in diesem Fall Start- und Endpunkt der Kurve im ersten bzw. letzten Punkt des Kontrollpolygons liegen. Falls Start- oder Endpunkt nicht volle Multiplizität besitzen, ist der Träger der Kurve auf das Intervall $[u_p, u_n]$ beschränkt. Die Gewichte w_i bestimmen den Einfluss der jeweiligen Kontrollpunkte P_i auf die Kurve.

Geschlossene Kurven. Sind Start- und Endpunkt gleich ($P_0 = P_n - 1$) und der Knotenvektor `Knots` auf *'auto'* gesetzt, wird die Kurve automatisch geschlossen, so dass die Multiplizität aller Knoten eins ist. Falls ein Gewichtsvektor `Weights` angegeben wird, muss dessen Länge in diesem Fall $n - 1$ betragen.

Parameter

▷ **`Contour`** (output_object) . . . . . xld_cont ⇝ *object*
Die Kontur, die die Kurve approximiert.

▷ **`Rows`** (input_control) . . . . . coordinates.y-array ⇝ *real*
Zeilenkoordinaten des Kontrollpolygons.
Parameteranzahl: Rows == Cols && Rows >= Degree + 1

▷ **`Cols`** (input_control) . . . . . coordinates.x-array ⇝ *real*
Spaltenkoordinaten des Kontrollpolygons.
Parameteranzahl: Cols == Rows && Cols >= Degree + 1

▷ **`Knots`** (input_control) . . . . . string(-array) ⇝ *string* / real
Der Knotenvektor u.
Defaultwert: 'auto'
Wertevorschläge: `Knots` ∈ {'auto'}

▷ **`Weights`** (input_control) . . . . . string(-array) ⇝ *string* / real
Der Gewichtsvektor w.
Parameteranzahl: Weights == Rows
Defaultwert: 'auto'
Wertevorschläge: `Weights` ∈ {'auto'}
Restriktion: Weights > 0.0

▷ **`Degree`** (input_control) . . . . . integer ⇝ *integer*
Grad p der Kurve.
Defaultwert: 3
Wertevorschläge: `Degree` ∈ {2, 3, 4, 5}
Restriktion: Degree >= 2

▷ **`MaxError`** (input_control) . . . . . real ⇝ *real* / string
Maximale Distanz zwischen Approximation und Kurve.
Defaultwert: 1.0
Wertevorschläge: `MaxError` ∈ {'omit', 1.0, 2.0, 3.0, 4.0, 5.0}
Restriktion: MaxError > 0.0

▷ **`MaxDistance`** (input_control) . . . . . real ⇝ *real* / string
Maximaler Abstand zweier aufeinander folgender Punkte in der approximierten Kontur `Contour`.
Defaultwert: 5.0
Wertevorschläge: `MaxDistance` ∈ {'omit', 1.0, 2.0, 3.0, 4.0, 5.0}
Restriktion: MaxDistance > 0.0

Beispiel

```
* use a polygon XLD contour as control polygon and approximate
* the NURBS curve as contour

get_contour_xld(Polygon, Row, Col)
gen_contour_nurbs_xld(NURBSContour, Row, Col, 'auto', 'auto', 3, 1.0, 10.0)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_nurbs_interp

Nachfolger

gen_polygons_xld

Literatur

L. Piegl, W. Tiller: “The NURBS Book“, 2nd Edition, Springer, 1997.

Modul

Foundation

gen_contour_polygon_rounded_xld (: Contour : Row, Col, Radius, SamplingInterval :)

Erzeugen einer XLD-Kontur mit gerundeten Ecken aus einem Polygon (als Tupel).

gen_contour_polygon_rounded_xld erzeugt aus einem Polygon, das in den Tupeln Row und Col übergeben wird, eine XLD-Kontur Contour mit abgerundeten Ecken. Die Rundungen werden durch Kreisbögen beschrieben, die durch den Parameter Radius definiert sind. Für jeden mittels Row und Col festgelegten Polygonpunkt muss ein korrespondierender Radius spezifiziert werden. Im Falle eines geschlossenen Polygons wird der Anfangspunkt des Polygons gleich dem Endpunkt definiert wobei die korrespondierenden Radien dieser Punkte gleich sein müssen. Bei offenen Polygonen hingegen werden die Radien für den Anfangs- und Endpunkt ignoriert. Der Parameter SamplingInterval schließlich bezeichnet den Abstand der Stützpunkte der entstehenden Kontur Contour.

Parameter

▷ **Contour** (output_object) ..... xld_cont ⇝ *object*
Ausgabe-Kontur.

▷ **Row** (input_control) ..... coordinates.y-array ⇝ *real* / integer
Zeilen-Koordinaten des Polygons.
Defaultwert: [20,80,80,20,20]
Wertevorschläge: Row ∈ {0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500}

▷ **Col** (input_control) ..... coordinates.x-array ⇝ *real* / integer
Spalten-Koordinaten des Polygons.
Defaultwert: [20,20,80,80,20]
Wertevorschläge: Col ∈ {0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500}

▷ **Radius** (input_control) ..... number-array ⇝ *real* / integer
Radien der abgerundeten Ecken.
Defaultwert: [20,20,20,20,20]
Wertevorschläge: Radius ∈ {0, 1, 2, 5, 10, 20, 50}

▷ **SamplingInterval** (input_control) ..... number ⇝ *real* / integer
Abstand der Abtastpunkte.
Defaultwert: 1.0
Wertevorschläge: SamplingInterval ∈ {0.5, 1.0, 2.0, 5.0}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_region_contour`

Nachfolger

`smooth_contours_xld`, `gen_polygons_xld`

Siehe auch

`gen_contour_polygon_xld`, `gen_contours_skeleton_xld`

Modul

Foundation

gen_contour_polygon_xld (: Contour : Row, Col :)

Erzeugen einer XLD-Kontur aus einem Polygon (als Tupel).

`gen_contour_polygon_xld` erzeugt aus einem Polygon, das in den Tupeln `Row` und `Col` übergeben wird, eine XLD-Kontur `Contour`. Dieser Operator ist dazu gedacht, Segmentierungsergebnisse, die nicht in der Kern-Bibliothek erzeugt worden sind, einzuspeichern, um auf ihnen Funktionen wie Polygonapproximation oder Erkennung von Parallelen auszuführen.

Parameter

▷ **Contour** (output_object) ..... xld_cont ⇝ *object*
Ausgabe-Kontur.

▷ **Row** (input_control) ..... coordinates.y-array ⇝ *real* / integer
Zeilen-Koordinaten des Polygons.
Defaultwert: [0,1,2,2,2]
Wertevorschläge: `Row` ∈ {0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500}

▷ **Col** (input_control) ..... coordinates.x-array ⇝ *real* / integer
Spalten-Koordinaten des Polygons.
Defaultwert: [0,0,0,1,2]
Wertevorschläge: `Col` ∈ {0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`get_region_contour`

Nachfolger

`smooth_contours_xld`, `gen_polygons_xld`

Siehe auch

`gen_contours_skeleton_xld`, `get_contour_xld`

Modul

Foundation

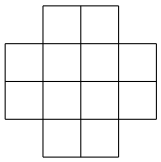
gen_contour_region_xld (Regions : Contours : Mode :)

Erzeugen von XLD-Konturen aus Regionen.

gen_contour_region_xld erzeugt aus einer oder mehreren Regionen die XLD-Konturen Contours. Dieser Operator ist dazu gedacht, Segmentierungsergebnisse, die als Regionen erzeugt worden sind, umzuwandeln, um auf ihnen Funktionen wie Polygonapproximation oder Erkennung von Parallelen auszuführen. Für jede Zusammenhangskomponente der Eingaberegionen wird eine geschlossene Kontur des Randes erzeugt. Der Parameter Mode kann folgende Werte annehmen:

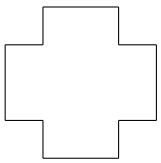
- *'center'*: Die Mittelpunkte der Randpixel werden als Konturpunkte verwendet.
- *'border'*: Der äußere Rand der Randpixel wird als Konturpunkte verwendet.
- *'border_holes'*: Zusätzlich werden noch die Konturen aller Löcher der Regionen erzeugt, wobei der äußere Rand der Randpixel als Konturpunkt verwendet wird .

Der Unterschied zwischen den zwei Modi *'border'* und *'center'* kann am Beispiel der folgenden Region leicht gesehen werden:

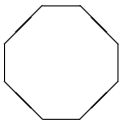


wobei □ ein einzelnes Pixel symbolisiert.

Die mit *'border'* und *'center'* berechneten Konturen sehen wie folgt aus:



'border'



'center'

Das bedeutet zum Beispiel, dass Konturen, die mit *'border'* erzeugt werden, im allgemeinen eine viel größere Euklidsche Länge (siehe length_xld) haben als Konturen, die mit *'center'* erzeugt werden. Dies liegt daran, dass für diagonale Randelemente bei *'border'* zwei Kontursegmente der Länge 1 erzeugt werden, während bei *'center'* ein einzelnes Element der Länge $\sqrt{2}$ verwendet wird. Offensichtlich haben auch andere Merkmale, z.B. die Fläche (siehe area_center_xld), verschiedene Werte.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *object*
Eingaberegionen.

▷ **Contours** (output_object) ..... xld_cont(-array) ⇝ *object*
Ausgabekonturen.

▷ **Mode** (input_control) ..... string ⇝ *string*
Art der Konturerzeugung.
Defaultwert: 'border'
Werteliste: Mode ∈ {'border', 'border_holes', 'center'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

smooth_contours_xld, gen_polygons_xld

Alternativen

gen_contour_polygon_xld, get_region_contour

Siehe auch

gen_contours_skeleton_xld

Modul

Foundation

gen_contours_skeleton_xld (Skeleton : Contours : Length, Mode :)

Umwandeln eines Skeletts in XLD-Konturen.

Mit gen_contours_skeleton_xld wird das in Skeleton als Region übergebene Skelett (z.B. Bildkanten) in die Konturdarstellung umgerechnet. Es wird angenommen, dass der Großteil der Region nur ein Pixel breit ist (siehe skeleton).

Der Algorithmus versucht zunächst, die Region so zu transformieren, dass sie in 8-Nachbarschaft nur noch Linienzüge enthält. In einem zweiten Schritt werden die Kreuzungspunkte markiert. Anschließend versucht gen_contours_skeleton_xld Konturen zu erzeugen, die in End- und Kreuzungspunkten enden, insbesondere für Kreuzungspunkte in den folgenden Konfigurationen (in allen vier Rotationen):

$$\begin{matrix}1&0&1\\0&2&0\\0&0&1\end{matrix}\quad\begin{matrix}1&0&1\\0&2&0\\1&0&1\end{matrix}\quad\begin{matrix}1&0&0\\0&2&1\\0&1&0\end{matrix}\quad\begin{matrix}1&0&0\\0&2&1\\1&0&0\end{matrix}\quad\begin{matrix}0&1&0\\0&2&1\\0&1&0\end{matrix}\quad\begin{matrix}0&1&0\\1&2&1\\0&1&0\end{matrix}$$

wobei 0 = Hintergrund, 1 = Vordergrund und 2 = Kreuzungspunkt.

Anschließend werden die Konturen zurückgegeben, die einschließlich End- und Kreuzungspunkten mindestens Length Punkte lang sind. Die Unterbrechung der Konturen an Kreuzungspunkten hat zur Folge, dass auch längere Linienzüge durch Kreuzen von kurzen Linien (auch wenn diese weniger als Length Punkte lang sind) in mehrere Konturen zerfallen (Mode *'filter'*). Dies wird mit dem Filtermodus (Mode) *'generalize1'* vermieden. In diesem Fall werden die Konturen so erzeugt, wie wenn die Konturen kürzer als Length Punkte nicht vorhanden wären. Damit auch Linienzüge, die durch Kreuzen von kurzen Linien in sehr kurze Konturen zerfallen, erhalten bleiben, kann der Filtermodus (Mode) *'generalize2'* eingestellt werden. In diesem Fall bleiben Linienstücke erhalten, deren beide Endpunkte Kreuzungspunkte mit anderen Linien sind, auch wenn sie kürzer als Length Punkte sind.

Parameter

▷ **Skeleton** (input_object) ..... region ⇝ *object*
Skelett für die Konturberechnung.

▷ **Contours** (output_object) ..... xld_cont-array ⇝ *object*
Ausgabe-Konturen.

▷ **Length** (input_control) ..... integer ⇝ *integer*
Mindestlänge zu erfassender Konturen.
Defaultwert: 1
Wertevorschläge: Length ∈ {1, 2, 3, 5, 10, 20}

▷ **Mode** (input_control) ..... string ⇝ *string*
Kontur-Filtermodus.
Defaultwert: 'filter'
Werteliste: Mode ∈ {'filter', 'generalize1', 'generalize2'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

skeleton

Nachfolger

smooth_contours_xld, get_contour_xld, gen_polygons_xld

Siehe auch

edges_image, threshold, get_region_contour

Modul

Foundation

gen_cross_contour_xld (: Cross : Row, Col, Size, Angle :)

Erzeugen einer XLD-Kontur in Form eines Kreuzes zu jedem Eingabepunkt.

gen_cross_contour_xld erzeugt zu jedem Eingabepunkt (Row,Col) eine kreuzförmige-XLD Kontur. Konzeptionell besteht diese XLD-Kontur aus zwei Linien der Länge Size, die sich genau im Eingabepunkt schneiden, und deren Orientierung von Angle vorgegeben wird. Das Kreuz wird in Cross zurückgegeben. Bei der Darstellung von mehr als einem Kreuz müssen die Koordinaten dabei jeweils in Form eines Tupels übergeben werden.

Parameter

▷ **Cross** (output_object) ..... xld_cont(-array) ⇝ *object*
Erzeugte XLD-Konturen.

▷ **Row** (input_control) ..... point.y(-array) ⇝ *real* / integer
Zeilenkoordinaten der Eingabepunkte.

▷ **Col** (input_control) ..... point.x(-array) ⇝ *real* / integer
Spaltenkoordinaten der Eingabepunkte.
Restriktion: number(Col) == number(Row)

▷ **Size** (input_control) ..... number ⇝ *real* / integer
Länge der Schenkel des Kreuzes.
Defaultwert: 6.0
Wertevorschläge: Size ∈ {4.0, 6.0, 8.0, 10.0}
Restriktion: 0.0 <= Size

▷ **Angle** (input_control) ..... angle.rad ⇝ *real*
Orientierung der Kreuze.
Defaultwert: 0.785398
Wertevorschläge: Angle ∈ {0.0, 0.785398}

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert gen_cross_contour_xld den Wert TRUE. durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

points_foerstner, points_harris, critical_points_sub_pix, local_max_sub_pix, local_min_sub_pix, saddle_points_sub_pix

Alternativen

disp_cross

Modul

Foundation

gen_ellipse_contour_xld (: ContEllipse : Row, Column, Phi, Radius1, Radius2, StartPhi, EndPhi, PointOrder, Resolution :)

Erzeugen einer XLD-Kontur aus einem Ellipsenbogen.

gen_ellipse_contour_xld generiert einen oder mehrere Ellipsenbögen bzw. geschlossene Ellipsen. Die Ellipsen werden dabei durch ihren Mittelpunkt (Row, Column), die Orientierung der Hauptachse Phi und die Länge der großen Radius1 bzw. kleinen Halbachse Radius2 beschrieben. Ellipsenbögen werden zusätzlich durch den Anfangs- und Endwinkel StartPhi und EndPhi des Start- bzw. Endpunktes, sowie den zugehörigen Umlaufsinn PointOrder charakterisiert. Die Anfangs- und Endwinkel werden von der positiven großen Halbachse aus gemessen, deren Orientierung durch Phi gegeben ist. Sie beziehen sich auf den kleinsten die Ellipse umgebenden Kreis. Der tatsächliche Anfangs- und Endpunkt der Ellipse ergibt sich aus dem Schnittpunkt der Ellipse mit der Orthogonalprojektion des entsprechenden Kreispunktes auf die große Halbachse. Beide Winkel werden dabei im

Ellipsenkoordinatensystem bezogen auf die Hauptachse der Ellipse mathematisch positiv angegeben. Die beiden Hauptpole der Ellipse haben demnach die Winkel 0 bzw. π, die Nebenpole die Winkel $\pi/2$ bzw. f3$\pi/2$. Um eine geschlossene Ellipse zu erhalten, müssen die Werte 0 und 2π (bei positivem Umlaufsinn) übergeben werden. Für alle Winkel sind beliebige Werte zugelassen, da diese intern auf das Intervall $[0, 2\pi]$ abgebildet werden. Die Auflösung der resultierenden XLD Konturen `ContEllipse` wird über den Parameter `Resolution` festgelegt: Er bestimmt den maximalen euklidischen Abstand zwischen benachbarten Konturpunkten. Die Auflösung wird auf den kleinsten zulässigen Wert gesetzt, wenn der Eingabewert diesen Wert unterschreitet.

Parameter

- ▷ **`ContEllipse`** (output_object) ..... xld_cont(-array) ⇝ *object*
  Ausgabe-Kontur.
- ▷ **`Row`** (input_control) ..... ellipse.center.y(-array) ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Ellipse.
  Defaultwert: 200.0
- ▷ **`Column`** (input_control) ..... ellipse.center.x(-array) ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Ellipse.
  Defaultwert: 200.0
- ▷ **`Phi`** (input_control) ..... ellipse.angle.rad(-array) ⇝ *real*
  Orientierung der Hauptachse [rad].
  Defaultwert: 0.0
- ▷ **`Radius1`** (input_control) ..... ellipse.radius1(-array) ⇝ *real*
  Länge der großen Halbachse.
  Defaultwert: 100.0
  Restriktion: Radius1 > 0
- ▷ **`Radius2`** (input_control) ..... ellipse.radius2(-array) ⇝ *real*
  Länge der kleinen Halbachse.
  Defaultwert: 50.0
  Restriktion: Radius2 >= 0 && Radius2 <= Radius1
- ▷ **`StartPhi`** (input_control) ..... angle.rad(-array) ⇝ *real*
  Winkel des Startpunktes auf dem kleinsten umgebenden Kreis [rad].
  Defaultwert: 0.0
- ▷ **`EndPhi`** (input_control) ..... angle.rad(-array) ⇝ *real*
  Winkel des Endpunktes auf dem kleinsten umgebenden Kreis [rad].
  Defaultwert: 6.28318
- ▷ **`PointOrder`** (input_control) ..... string(-array) ⇝ *string*
  mathematischer Umlaufsinn.
  Defaultwert: 'positive'
  Werteliste: `PointOrder` ∈ {'positive', 'negative'}
- ▷ **`Resolution`** (input_control) ..... real ⇝ *real*
  Auflösung: Maximaler Abstand zwischen benachbarten Konturpunkten.
  Defaultwert: 1.5
  Restriktion: Resolution >= 1.192e-7

Beispiel

```
draw_ellipse(WindowHandle,Row,Column,Phi,Radius1,Radius2)
gen_ellipse_contour_xld(Ellipse,Row,Column,Phi,Radius1,Radius2,0,6.28319, \
                        'positive',1.5)
length_xld(Ellipse,Length)
* Transform StartPhi so that the start point of the ellipse actually
* intersects the line through origin at the angle StartPhi measured from
* the positive main axis
affine_trans_point_2d ([1.0,0.0,0.0,0.0,1.0*Radius2/Radius1,0.0], \
                       sin(StartPhi), cos(StartPhi), Qx, Qy)
StartPhiEllipse := atan2(Qx,Qy)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_ellipse_contour_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

draw_ellipse

Nachfolger

disp_xld, get_points_ellipse

Alternativen

gen_circle_contour_xld

Modul

Foundation

gen_nurbs_interp (: : Rows, Cols, Tangents, Degree : CtrlRows, CtrlCols, Knots)

Erzeugen von Kontrolldaten einer NURBS Kurve, die gegebene Punkte interpoliert.

gen_nurbs_interp erzeugt die Kontrolldaten CtrlRows, CtrlCols und Knots einer NURBS Kurve, die die Eingabepunkte (Rows, Cols) interpoliert. Falls die Eingabepunktliste nicht geschlossen ist (das heißt, der erste und der letzte Punkt sind identisch), müssen zusätzlich die Tangenten im ersten und im letzten Punkt in der Reihenfolge $[drow_0, dcol_0, drow_n - 1, dcol_n - 1]$ in Tangents übergeben werden (für geschlossene Punktlisten muss Tangents ein leeres Tupel sein). Darüberhinaus muss der gewünschte Grad für die NURBS Kurve in Degree angegeben werden.

Die Ausgabe von gen_nurbs_interp kann direkt in gen_contour_nurbs_xld verwendet werden, wenn der Gewichtsvektor auf *'auto'* gesetzt wird.

Ausführliche Informationen über NURBS Kurven befinden sich in der Dokumentation von gen_contour_nurbs_xld.

Parameter

▷ **Rows** (input_control) . . . . . coordinates.y-array ⇝ *real*
Zeilenkoordinaten der Eingabepunkte.
Parameteranzahl: Rows == Cols

▷ **Cols** (input_control) . . . . . coordinates.x-array ⇝ *real*
Spaltenkoordinaten der Eingabepunkte.
Parameteranzahl: Cols == Rows

▷ **Tangents** (input_control) . . . . . real-array ⇝ *real*
Tangenten des ersten und letzten Punktes.
Defaultwert: []

▷ **Degree** (input_control) . . . . . integer ⇝ *integer*
Grad der Ausgabekurve.
Defaultwert: 3
Wertevorschläge: Degree ∈ {2, 3, 4, 5}
Restriktion: Degree >= 2

▷ **CtrlRows** (output_control) . . . . . coordinates.y-array ⇝ *real*
Zeilenkoordinaten des Kontrollpolygons.

▷ **CtrlCols** (output_control) . . . . . coordinates.x-array ⇝ *real*
Spaltenkoordinaten des Kontrollpolygons.
Parameteranzahl: Cols == Rows

▷ **Knots** (output_control) . . . . . real-array ⇝ *real*
Knotenvektor der Ausgabekurve.

Ergebnis

Sind alle Eingabeparameter korrekt, gibt gen_nurbs_interp den Wert TRUE zurück.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Nachfolger

gen_contour_nurbs_xld

Siehe auch

draw_nurbs_interp, draw_nurbs_interp_mod

Literatur

L. Piegl, W. Tiller: "The NURBS Book", 2nd Edition, Springer, 1997.

Modul

Foundation

gen_parallels_xld (Polygons : Parallels : Len, Dist, Alpha, Merge :)

Suchen nach parallelen XLD-Polygonen.

Mit gen_parallels_xld werden die in Polygons übergebenen Polygone auf Parallelität hin untersucht. Die resultierenden parallelen Teilstücke der Polygone werden unter Parallels ausgegeben. Falls der Parameter Merge auf *'true'* gesetzt wird, werden nebeneinanderliegende parallele Teilstücke in einer einzigen Relation gespeichert. Ansonsten wird für jedes Paar von parallelen Liniensegmenten eine separate Parallelenrelation ausgegeben.

Es werden nur Polygonsegmente mit einer Länge von mindestens Len berücksichtigt. Ob zwei Polygonsegmente als parallel angesehen werden, hängt vor allem von der maximalen Winkeltoleranz Alpha (Bogenmaß) ab. Ferner müssen sich die Projektionen der beiden Segmente auf ihre Winkelhalbierende überlappen. Der Überlappungsbereich auf der Winkelhalbierenden wird zurück auf die Segmente projiziert. Ist der maximale Abstand der Rückprojektionen kleiner als Dist, werden die Segmente in Parallels gespeichert.

Als Nebeneffekt wird zu jedem parallelen Linienpaar ein sog. Qualitätsfaktor berechnet, siehe mod_parallels_xld. Dieser ergibt sich aus der normierten Winkeldifferenz und der normierten Länger der Überlappung:

$$q = \frac{\pi/2 - \delta\alpha}{\pi/2} \frac{2overlap}{len_1 + len_2} \qquad \text{mit} \quad 0 \leq q \leq 1$$

Dabei ist $\delta\alpha$ die Winkeldifferenz der Polygonsegmente, $overlap$ der Überlappungsbereich, len_1 und len_2 die Länge der Polygonsegmente und q der errechnete Qualitätsfaktor. Dieser Wert stellt ein Maß für die Parallelität dar (je größer dieser Wert, desto „paralleler" die Linien). Anschließend werden die Qualitäts-Faktoren aller parallelen Linien in gemeinsamen Polygonen, gewichtet nach ihrem Anteil an der gemeinsamen Überlappung, aufsummiert. Die Spanne für den errechneten Qualitätsfaktor kann mit info_parallels_xld abgerufen werden.

Parameter

▷ **Polygons** (input_object) ..... xld_poly-array ⇝ *object*
Eingabepolygone.

▷ **Parallels** (output_object) ..... xld_para-array ⇝ *object*
Parallelenrelationen.

▷ **Len** (input_control) ..... number ⇝ *real* / integer
Mindestlänge der einzelnen Polygonlinien.
Defaultwert: 10.0
Wertevorschläge: Len ∈ {5.0, 10.0, 15.0, 20.0}
Restriktion: Len > 0.0

▷ **Dist** (input_control) . . . . . number ⇝ *real* / integer
Maximalabstand der einzelnen Polygonlinien.
Defaultwert: 30.0
Wertevorschläge: Dist ∈ {20.0, 25.0, 30.0, 40.0, 50.0, 75.0}
Restriktion: Dist > 0.0

▷ **Alpha** (input_control) . . . . . number ⇝ *real* / integer
Maximale Winkeltoleranz der einzelnen Polygonlinien.
Defaultwert: 0.15
Wertevorschläge: Alpha ∈ {0.05, 0.10, 0.15, 0.20, 0.30}
Restriktion: 0 <= Alpha && Alpha <= pi / 2

▷ **Merge** (input_control) . . . . . string ⇝ *string*
Sollen nebeneinanderliegende Parallelenrelationen zusammengefügt werden?
Defaultwert: 'true'
Werteliste: Merge ∈ {'true', 'false'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_polygons_xld

Nachfolger

mod_parallels_xld, get_parallels_xld, info_parallels_xld

Modul

Foundation

gen_polygons_xld (Contours : Polygons : Type, Alpha :)

Approximieren von Polygonen an XLD-Konturen.

Mit gen_polygons_xld werden, unter Angabe eines Approximationsverfahrens Type und eines Schwellenwertes Alpha, XLD-Konturen, die in Contours abgespeichert sind, approximiert. Es werden Polygone berechnet, mit deren Stützpunkten die Konturen angenähert werden können. Es können dabei sowohl offene als auch geschlossene Konturen approximiert werden. Die gewonnenen Stützpunkte werden in dem Parameter Polygons zurückgegeben.

Es steht das Verfahren von Ramer, in welchem die Konturen so approximiert werden, dass der euklidische Abstand eines jeden Konturpunktes von der Polygonlinie höchstens Alpha Pixeleinheiten beträgt.

Parameter

▷ **Contours** (input_object) . . . . . xld_cont-array ⇝ *object*
Konturen, die approximiert werden sollen.

▷ **Polygons** (output_object) . . . . . xld_poly-array ⇝ *object*
Approximierende Polygone.

▷ **Type** (input_control) . . . . . string ⇝ *string*
Name des Polygonverfahrens.
Defaultwert: 'ramer'
Werteliste: Type ∈ {'ramer'}

▷ **Alpha** (input_control) . . . . . number ⇝ *real* / integer
Schwellenwert der Polygonapproximation.
Defaultwert: 2.0
Wertevorschläge: Alpha ∈ {1.0, 1.5, 2.0, 3.0, 4.0}
Restriktion: Alpha > 0.0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Nachfolger

gen_parallels_xld, split_contours_xld

Siehe auch

get_region_polygon

Modul

Foundation

gen_rectangle2_contour_xld (: Rectangle : Row, Column, Phi, Length1, Length2 :)

Erzeugen einer XLD-Kontur in Form eines Rechtecks.

gen_rectangle2_contour_xld erzeugt eine oder mehrere XLD-Konturen in Form eines beliebig orientierten Rechtecks. Das Rechteck hat den Mittelpunkt (Row, Column), die Orientierung Phi und die halben Kantenlängen Length1 und Length2. Der Winkel Phi wird im Bogenmaß spezifiziert und gibt den Winkel der horizontalen Achse und der Kante mit der halben Länge Length1 im mathematisch positiven Sinn (gegen den Uhrzeigersinn) an. Es kann mehr als eine XLD-Kontur erzeugt werden, indem ein Tupel von Rechteckparametern übergeben wird.

Parameter

▷ **Rectangle** (output_object) ..... xld_cont(-array) ⇝ *object*
Rechteckskontur.

▷ **Row** (input_control) ..... rectangle2.center.y(-array) ⇝ *real*
Zeilenkoordinate des Mittelpunktes des Rechtecks.
Defaultwert: 300.0

▷ **Column** (input_control) ..... rectangle2.center.x(-array) ⇝ *real*
Spaltenkoordinate des Mittelpunktes des Rechtecks.
Defaultwert: 200.0

▷ **Phi** (input_control) ..... rectangle2.angle.rad(-array) ⇝ *real*
Orientierung der Hauptachse des Rechtecks in Bogenmaß.
Defaultwert: 0.0
Restriktion: - pi / 2 < Phi && Phi <= pi / 2

▷ **Length1** (input_control) ..... rectangle2.hwidth(-array) ⇝ *real*
Erster Halbmesser (halbe Länge) des Rechtecks.
Defaultwert: 100.5

▷ **Length2** (input_control) ..... rectangle2.hheight(-array) ⇝ *real*
Zweiter Halbmesser (halbe Breite) des Rechtecks.
Defaultwert: 20.5

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_rectangle2_contour_xld den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

fit_rectangle2_contour_xld

Alternativen

gen_rectangle2

Siehe auch

gen_contour_polygon_xld

Modul

Foundation

mod_parallels_xld (Parallels, Image : ModParallels, ExtParallels : Quality, MinGray, MaxGray, MaxStandard :)

Suchen nach parallelen Polygonen mit eingeschlossener homogener Fläche.

mod_parallels_xld überprüft die Grauwerte im Bild Image zwischen zwei parallelen Linien (Überlappungsbereich), die unter Parallels abgespeichert sind, auf Homogenität.

Es werden nur parallele Polygone betrachtet, deren Quality-Faktor größer gleich Quality ist. Zwischen je zwei parallelen Linien wird deren gemeinsame zentrale Überlappungsstrecke entlanggelaufen und jeweils der dazu senkrechte Grauwertschnitt errechnet (cross section).

In einem ersten Durchlauf wird der mittlere Grauwert für jede Schnittgerade ermittelt. In einem zweiten Durchlauf werden nun die Standardabweichungen jeder Schnittgeraden bezüglich ihres mittleren, empirisch ermittelten Grauwertes berechnet.

Falls der mittlere Grauwert einer betrachteten Fläche im Intervall [MinGray,MaxGray] liegt und das Mittel aus allen Standardabweichungen kleiner als die obere Schranke MaxStandard ist, so werden die parallelen Linien in ModParallels als modifizierte parallele Polygonteilstücke zurückgegeben.

Eine zweite Routine ist in diese Funktion eingebettet. Diese überprüft auf die gleiche Art wie oben erwähnt alle Anschlusslinien, also alle Polygonlinien, die Anschlussstücke der gefundenen Parallelrelationen sind, auf Homogenität. Dabei wird entlang dieser Anschlusslinien gelaufen und ein Bereich untersucht, der genauso breit ist, wie die Breite des letzten homogenen Parallelenstücks, untersucht, also die Seite der Linie, auf der auch das in der Relation zugehörige Polygon liegt. Ist auch hier der Grauwertverlauf im Sinne von oben homogen, so wird der Bereich der nächsten Anschlusslinie untersucht. Die so gefundenen Endpunkte jedes Polygons aus der modifizierten Parallelenrelation werden in ExtParallels als erweiterte Parallelen zurückgegeben.

Parameter

- ▷ **Parallels** (input_object) ..... xld_para-array ⇝ *object*
  Parallele Polygonteilstücke.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object : byte*
  Zu untersuchendes Eingabebild.
- ▷ **ModParallels** (output_object) ..... xld_mod_para-array ⇝ *object*
  Modifizierte parallele Polygonteilstücke.
- ▷ **ExtParallels** (output_object) ..... xld_ext_para-array ⇝ *object*
  Erweiterte parallele Polygonteilstücke.
- ▷ **Quality** (input_control) ..... number ⇝ *real* / integer
  Minimale Schwelle des Quality-Faktors (Maß für Parallelität).
  Defaultwert: 0.4
  Wertevorschläge: Quality ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6}
  Restriktion: 0.0 <= Quality && Quality <= 1.0
- ▷ **MinGray** (input_control) ..... integer ⇝ *integer*
  Minimale Schwelle des mittleren Grauwertes.
  Defaultwert: 160
  Wertevorschläge: MinGray ∈ {80, 100, 120, 140, 160, 180}
  Restriktion: 0 <= MinGray && MinGray <= 255
- ▷ **MaxGray** (input_control) ..... integer ⇝ *integer*
  Maximale Schwelle des mittleren Grauwertes.
  Defaultwert: 220
  Wertevorschläge: MaxGray ∈ {140, 160, 180, 200, 220, 240}
  Restriktion: 0 <= MaxGray && MaxGray <= 255 && MaxGray >= MinGray

▷ **MaxStandard** (input_control) . . . . . number ⇝ *real* / integer
Maximal zugelassene gemittelte Standardabweichung bzgl. der mittleren Grauwerte.
Defaultwert: 10.0
Wertevorschläge: `MaxStandard` ∈ {5.0, 10.0, 15.0, 20.0}
Restriktion: MaxStandard >= 0.0

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_parallels_xld`

Nachfolger

`max_parallels_xld`

Siehe auch

`info_parallels_xld`

Modul

Foundation

29.2 Geometrische Transformationen

```
affine_trans_contour_xld (
    Contours : ContoursAffineTrans : HomMat2D : )
```

Anwenden einer beliebigen affinen 2D-Transformation auf XLD-Konturen.

`affine_trans_contour_xld` wendet eine beliebige affine 2D-Transformation (Skalierung, Drehung, Verschiebung, Scherung) auf die in `Contours` übergebenen XLD-Konturen an und liefert die transformierten Konturen in `ContoursAffineTrans` zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in `HomMat2D` übergeben wird. Diese kann mit Hilfe der Operatoren `hom_mat2d_identity`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_translate` etc. aufgebaut werden oder das Resultat von Operatoren wie `vector_angle_to_rigid` sein.

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, dass die *Zeilen*-Koordinate des Bildes der x-Koordinate und die *Spalten*-Koordinate des Bildes der y-Koordinate des Koordinatensystems entspricht, in dem die Transformationsmatrix definiert wurde. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Achtung

`affine_trans_contour_xld` verwendet nicht das HALCON-Standard-Koordinatensystem (mit dem Ursprung im Mittelpunkt des linken oberen Pixels) sondern das gleiche Koordinatensystem wie `affine_trans_pixel`, d.h. der Ursprung liegt in der linken oberen Ecke des linken oberen Pixels. Eine Transformation mit `affine_trans_contour_xld` entspricht für jeden Punkt der Kontur einer Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt, siehe `affine_trans_pixel`). Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von der Kontur mit Operatoren wie z.B. `area_center_xld` abgeleitet wurden. Wenn z.B. eine rotationssymmetrische XLD Kontur unter Verwendung von `hom_mat2d_rotate` um ihren mit `area_center_xld` bestimmten Schwerpunkt rotiert wird, so liegt die transformierte Kontur nicht auf der Originalkontur. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von `affine_trans_contour_xld` die Transformationsmatrix `HomMat2D` durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
```

```
affine_trans_contour_xld(Contours, ContoursAffineTrans, HomMat2DAdapted)
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabe-XLD-Konturen.
- ▷ **ContoursAffineTrans** (output_object) ..... xld_cont(-array) ⇝ *object*
  Transformierte XLD-Konturen.
- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.

Ergebnis

Falls die Matrix HomMat2D eine affine Abbildung darstellt (d.h. keine projektive Abbildung), liefert affine_trans_contour_xld den Wert TRUE zurück. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_rotate, hom_mat2d_scale, hom_mat2d_reflect

Siehe auch

affine_trans_image, affine_trans_region

Modul

Foundation

affine_trans_polygon_xld (Polygons : PolygonsAffineTrans : HomMat2D :)

Anwenden einer beliebigen affinen Transformation auf XLD-Polygone

affine_trans_polygon_xld wendet eine beliebige affine Transformation (Skalierung, Drehung, Verschiebung, Scherung) auf die in Polygons übergebenen XLD-Polygone an und liefert die transformierten Polygone in PolygonsAffineTrans zurück. Die Transformation wird durch eine homogene Transformationsmatrix beschrieben, die in HomMat2D übergeben wird. Diese kann mit Hilfe der Operatoren hom_mat2d_identity, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_translate etc. aufgebaut werden oder das Resultat von Operatoren wie vector_angle_to_rigid sein. Die transformierten Polygone werden in PolygonsAffineTrans zurückgeliefert.

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, dass die *Zeilen*-Koordinate des Bildes der x-Koordinate und die *Spalten*-Koordinate des Bildes der y-Koordinate des Koordinatensystems entspricht, in dem die Transformationsmatrix definiert wurde. Dies ist notwendig, um für das Bild ein rechtshändiges Koordinatensystem zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn ausgeführt. Die Koordinatenreihenfolge (x,y) der Matrizen entspricht dann der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder.

Achtung

Die XLD-Konturen, die möglicherweise von Polygons referenziert werden, werden weder transformiert noch mit den Ergebnispolygonen abgespeichert, da dies im allgemeinen nicht möglich ist, ohne Inkonsistenzen bei den Attributen der XLD-Konturen zu erzeugen. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen, z.B. split_contours_xld, zu Fehlermeldungen kommen.

affine_trans_polygon_xld verwendet nicht das HALCON-Standard-Koordinatensystem (mit dem Ursprung im Mittelpunkt des linken oberen Pixels) sondern das gleiche Koordinatensystem wie

affine_trans_pixel, d.h. der Ursprung liegt in der linken oberen Ecke des linken oberen Pixels. Eine Transformation mit affine_trans_polygon_xld entspricht für jeden Punkt des Polygons einer Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt, siehe affine_trans_pixel). Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von dem Polygon mit Operatoren wie z.B. area_center_xld abgeleitet wurden. Wenn z.B. ein rotationssymmetrisches Polygon unter Verwendung von hom_mat2d_rotate um seinen mit area_center_xld bestimmten Schwerpunkt rotiert wird, so liegt das transformierte Polygon nicht auf dem Originalpolygon. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von affine_trans_polygon_xld die Transformationsmatrix HomMat2D durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
affine_trans_polygon_xld(Polygons, PolygonsAffineTrans, HomMat2DAdapted)
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

- ▷ **Polygons** (input_object) ..... xld_poly(-array) ⇝ *object*
  Eingabe-XLD-Polygone.
- ▷ **PolygonsAffineTrans** (output_object) ..... xld_poly(-array) ⇝ *object*
  Transformierte XLD-Polygone.
- ▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
  Eingabe-Transformationsmatrix.

Ergebnis

Falls die Matrix HomMat2D eine affine Abbildung darstellt (d.h. keine projektive Abbildung), liefert affine_trans_polygon_xld den Wert TRUE zurück. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_rotate, hom_mat2d_scale, hom_mat2d_reflect

Siehe auch

affine_trans_image, affine_trans_region, affine_trans_contour_xld

Modul

Foundation

gen_parallel_contour_xld (Contours : ParallelContours : Mode, Distance :)

Berechnen der Parallelkontur einer XLD-Kontur

gen_parallel_contour_xld berechnet zu jeder der Eingabekonturen Contours eine parallele Kontur mit dem Abstand Distance. Die berechneten Konturen werden in ParallelContours zurückgegeben. Zur Berechnung der Parallelkontur wird der Normalenvektor der Eingabekontur in jedem Konturpunkt benötigt. Der Parameter Mode gibt an, wie die Normalenvektoren bestimmt werden. Falls Mode = *'gradient'*, wird angenommen, dass die Eingabekonturen Kanten sind, und die Normaleninformation aus der Gradientenrichtung der Kante gewonnen (siehe edges_sub_pix). Dazu muss bei der Eingabekontur das Attribut *'edge_direction'* definiert sein (siehe get_contour_attrib_xld). Falls Mode = *'contour_normal'*, wird eine eventuell schon vorhandene Normaleninformation zur Berechnung der Normalen verwendet. Dazu muss das Konturattribut *'angle'* definiert

sein (siehe lines_gauss oder edges_sub_pix). Falls schließlich Mode = *'regression_normal'*, wird die Normalenrichtung aus einer lokalen Regressionsgeraden in jedem Konturpunkt bestimmt. Dabei werden die Normalenvektoren so ausgerichtet, dass sie auf die rechte Seite der Kontur zeigen. Dieser Modus kann, im Gegensatz zu den ersten zwei Modi, für alle XLD-Konturen angewendet werden, ganz gleich wie sie erzeugt wurden.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *object*
  Zu transformierende Konturen.
- ▷ **ParallelContours** (output_object) ..... xld_cont-array ⇝ *object*
  Parallelkonturen.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus, mit dem die Richtungsinformation gewonnen wird.
  Defaultwert: 'regression_normal'
  Wertevorschläge: Mode ∈ {'gradient', 'contour_normal', 'regression_normal'}
- ▷ **Distance** (input_control) ..... number ⇝ *real* / integer
  Abstand der Parallelkontur.
  Defaultwert: 1
  Wertevorschläge: Distance ∈ {0.2, 0.4, 0.6, 0.8, 1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 50}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix, threshold_sub_pix

Nachfolger

gen_polygons_xld

Siehe auch

get_contour_xld

Modul

Foundation

polar_trans_contour_xld (Contour : PolarTransContour : Row, Column, AngleStart, AngleEnd, RadiusStart, RadiusEnd, Width, Height :)

Anwenden einer Polarkoordinatentransformation auf eine Kontur.

polar_trans_contour_xld transformiert eine Kontur Contour im durch den Mittelpunkt (Row, Column), die Radien RadiusStart und RadiusEnd und die Winkel AngleStart und AngleEnd definierten Kreisring auf dessen Polarkoordinatendarstellung in einem virtuellen Bild mit den Dimensionen Width × Height. Die Ausgabekontur wird an den Rändern dieses virtuellen Bildes beschnitten.

Die Koordinate $(0, 0)$ in der Ausgabekontur PolarTransContour entspricht immer dem Konturpunkt, der durch RadiusStart und AngleStart spezifiziert wird. Analog entspricht die Koordinate Height − 1, Width − 1 in der Ausgabekontur immer dem Konturpunkt, der durch RadiusEnd und AngleEnd spezifiziert wird, auch wenn diese Punkte nicht in der Kontur Contour enthalten sind. Im Normalfall (AngleStart < AngleEnd und RadiusStart < RadiusEnd) wird die Polarkoordinatentransformation also in mathematisch positiver Durchlaufrichtung (entgegen dem Uhrzeigersinn) durchgeführt. Dabei liegen die Punkte mit kleineren Radien im virtuellen Ausgabebild weiter oben. Durch geeignetes Vertauschen der Werte der Parameter (z.B. AngleStart > AngleEnd oder RadiusStart > RadiusEnd) kann jede beliebige Orientierung der Ausgabekontur erreicht werden.

Die Winkel sind frei als reelle Zahlen wählbar, Mittelpunkt und Radien können ebenfalls frei als reelle Zahlen gewählt werden.

PolarTransContour kann aus mehreren Konturen bestehen, da die Kontur Contour an den Winkelgrenzen AngleStart bzw. AngleEnd geteilt werden kann. Außerdem kann die Länge des Winkelintervalls [AngleStart, AngleEnd] größer sein als $2pi$.

Werden mehrere Konturen in Contour übergeben, werden deren Polartransformationen individuell berechnet und als Tupel in PolarTransContour abgespeichert. Da eine Kontur bei der Transformation in mehrere Konturen geteilt werden kann, besteht kein Zusammenhang zwischen den Indizes des Eingabetupels Contour und den Indizes des Ausgabetupels PolarTransContour.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Achtung

Es werden nur die Konturpunkte transformiert. Da die Polartransformation keine affine Abbildung ist, liefert der Operator polar_trans_contour_xld nur dann brauchbare Ergebnisse, wenn die Abstände der Konturpunkte klein sind. Eventuelle Attribute werden nicht mittransformiert.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabekontur.
- ▷ **PolarTransContour** (output_object) ..... xld_cont(-array) ⇝ *object*
  Ergebniskontur.
- ▷ **Row** (input_control) ..... number ⇝ *real* / integer
  Zeilenkoordinate des Kreismittelpunktes.
  Defaultwert: 256
  Wertevorschläge: Row ∈ {0, 16, 32, 64, 128, 240, 256, 480, 512}
  Typischer Wertebereich: 0 ≤ Row ≤ 32767
  Restriktion: Row >= -131068 && Row <= 131068
- ▷ **Column** (input_control) ..... number ⇝ *real* / integer
  Spaltenkoordinate des Kreismittelpunktes.
  Defaultwert: 256
  Wertevorschläge: Column ∈ {0, 16, 32, 64, 128, 256, 320, 512, 640}
  Typischer Wertebereich: 0 ≤ Column ≤ 32767
  Restriktion: Column >= -131068 && Column <= 131068
- ▷ **AngleStart** (input_control) ..... angle.rad ⇝ *real*
  Winkel, der auf die Spaltenkoordinate *0* in PolarTransContour abgebildet wird.
  Defaultwert: 0.0
  Wertevorschläge: AngleStart ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853, 12.566370616}
  Typischer Wertebereich: -6.2831853 ≤ AngleStart ≤ 6.2831853
- ▷ **AngleEnd** (input_control) ..... angle.rad ⇝ *real*
  Winkel, der auf die Spaltenkoordinate Width − 1 in PolarTransContour abgebildet wird.
  Defaultwert: 6.2831853
  Wertevorschläge: AngleEnd ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853, 12.566370616}
  Typischer Wertebereich: -6.2831853 ≤ AngleEnd ≤ 6.2831853
- ▷ **RadiusStart** (input_control) ..... number ⇝ *real* / integer
  Kreisradius, der auf die Zeilenkoordinate *0* in PolarTransContour abgebildet wird.
  Defaultwert: 0
  Wertevorschläge: RadiusStart ∈ {0, 16, 32, 64, 100, 128, 256, 512}
  Typischer Wertebereich: 0 ≤ RadiusStart ≤ 32767
- ▷ **RadiusEnd** (input_control) ..... number ⇝ *real* / integer
  Kreisradius, der auf die Zeilenkoordinate Height − 1 in PolarTransContour abgebildet wird.
  Defaultwert: 100
  Wertevorschläge: RadiusEnd ∈ {0, 16, 32, 64, 100, 128, 256, 512}
  Typischer Wertebereich: 0 ≤ RadiusEnd ≤ 32767
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des virtuellen Zielbildes.
  Defaultwert: 512
  Wertevorschläge: Width ∈ {256, 320, 512, 640, 800, 1024}
  Typischer Wertebereich: 0 ≤ Width ≤ 32767

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des virtuellen Zielbildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {240, 256, 480, 512, 600, 1024}
Typischer Wertebereich: 0 ≤ Height ≤ 32767

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Siehe auch

polar_trans_image_ext, polar_trans_image_inv, polar_trans_region, polar_trans_region_inv, polar_trans_contour_xld_inv

Modul

Foundation

polar_trans_contour_xld_inv (PolarContour : XYTransContour : Row, Column, AngleStart, AngleEnd, RadiusStart, RadiusEnd, WidthIn, HeightIn, Width, Height :)

Rücktransformieren einer Kontur in Polarkoordinaten

polar_trans_contour_xld_inv transformiert die Polarkoordinatendarstellung eines Kontur, gespeichert in PolarContour, zurück auf einen Kreisring in kartesischen Koordinaten, der durch die Radien RadiusStart und RadiusEnd sowie durch die Winkel AngleStart und AngleEnd beschrieben ist und dessen Mittelpunkt in (Row, Column) liegt. Alle diese Werte können frei aus den reellen Zahlen gewählt werden. Die Gesamtgröße des Ausgabebildes beträgt Width × Height Pixel. Zusätzlich müssen die Dimensionen des virtuellen Eingabebildes, welches die Kontur PolarContour enthält, in WidthIn und HeightIn übergeben werden. WidthIn − 1 ist die Spaltenkoordinate, die AngleEnd entspricht und HeightIn − 1 ist die Zeilenkoordinate, die RadiusEnd entspricht. AngleStart bzw. RadiusStart entspricht der Spalten- bzw. Zeilenkoordinate *0*. Außerdem müssen die Dimensionen des virtuellen Ausgabebildes, welches die Region XYTransContour enthält, in Width und Height übergeben werden. An dessen Rändern wird die Ausgabekontur beschnitten.

polar_trans_contour_xld_inv ist die exakte Umkehrfunktion zu polar_trans_contour_xld.

Die Aufrufsequenz:

```
polar_trans_contour_xld(Contour, PolarContour, Row, Column, rad(360), 0, 0,
Radius, Width, Height)
polar_trans_contour_xld_inv(PolarContour, XYTransContour, Row, Column,
rad(360), 0, 0, Radius, Width, Height, WidthOut, HeightOut)
```

liefert in der Ausgabekontur XYTransCont die Kontur Image eingeschränkt auf den Kreis um (Row, Column) mit Radius Radius zurück.

XYTransContour kann aus mehreren Konturen bestehen, da die Kontur PolarContour an den Rändern des virtuellen Ausgabebildes, das durch Width und Height definiert wird, beschnitten wird.

Werden mehrere Konturen in PolarContour übergeben, werden deren Transformationen individuell berechnet und als Tupel in XYTransContour abgespeichert. Da eine Kontur bei der Transformation in mehrere Konturen geteilt werden kann, besteht kein Zusammenhang zwischen den Indizes des Eingabetupels PolarContour und den Indizes des Ausgabetupels XYTransContour.

Weitere Informationen

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Achtung

Es werden nur die Konturpunkte transformiert. Da die Polartransformation keine affine Abbildung ist, liefert der

Operator `polar_trans_contour_xld_inv` nur dann brauchbare Ergebnisse, wenn die Abstände der Konturpunkte klein sind. Eventuelle Attribute werden nicht mittransformiert.

Parameter

▷ **`PolarContour`** (input_object) . . . . . xld_cont(-array) ⇝ *object*
Eingabekontur.

▷ **`XYTransContour`** (output_object) . . . . . xld_cont(-array) ⇝ *object*
Ergebniskontur.

▷ **`Row`** (input_control) . . . . . number ⇝ *real* / integer
Zeilenkoordinate des Kreismittelpunktes.
Defaultwert: 256
Wertevorschläge: `Row` ∈ {0, 16, 32, 64, 128, 240, 256, 480, 512}
Typischer Wertebereich: 0 ≤ `Row` ≤ 32767
Restriktion: Row >= -131068 && Row <= 131068

▷ **`Column`** (input_control) . . . . . number ⇝ *real* / integer
Spaltenkoordinate des Kreismittelpunktes.
Defaultwert: 256
Wertevorschläge: `Column` ∈ {0, 16, 32, 64, 128, 256, 320, 512, 640}
Typischer Wertebereich: 0 ≤ `Column` ≤ 32767
Restriktion: Column >= -131068 && Column <= 131068

▷ **`AngleStart`** (input_control) . . . . . angle.rad ⇝ *real*
Winkel, auf den die Spaltenkoordinate *0* von `PolarContour` abgebildet wird.
Defaultwert: 0.0
Wertevorschläge: `AngleStart` ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853}
Typischer Wertebereich: -6.2831853 ≤ `AngleStart` ≤ 6.2831853

▷ **`AngleEnd`** (input_control) . . . . . angle.rad ⇝ *real*
Winkel, auf den die Spaltenkoordinate `WidthIn` − 1 in `PolarContour` abgebildet wird.
Defaultwert: 6.2831853
Wertevorschläge: `AngleEnd` ∈ {0.0, 0.78539816, 1.57079632, 3.141592654, 6.2831853}
Typischer Wertebereich: -6.2831853 ≤ `AngleEnd` ≤ 6.2831853

▷ **`RadiusStart`** (input_control) . . . . . number ⇝ *real* / integer
Kreisradius, auf den die Zeilenkoordinate *0* in `PolarContour` abgebildet wird.
Defaultwert: 0
Wertevorschläge: `RadiusStart` ∈ {0, 16, 32, 64, 100, 128, 256, 512}
Typischer Wertebereich: 0 ≤ `RadiusStart` ≤ 32767

▷ **`RadiusEnd`** (input_control) . . . . . number ⇝ *real* / integer
Kreisradius, auf den die Zeilenkoordinate `HeightIn` − 1 in `PolarContour` abgebildet wird.
Defaultwert: 100
Wertevorschläge: `RadiusEnd` ∈ {0, 16, 32, 64, 100, 128, 256, 512}
Typischer Wertebereich: 0 ≤ `RadiusEnd` ≤ 32767

▷ **`WidthIn`** (input_control) . . . . . extent.x ⇝ *integer*
Breite des virtuellen Eingabebildes.
Defaultwert: 512
Wertevorschläge: `WidthIn` ∈ {256, 320, 512, 640, 800, 1024}
Typischer Wertebereich: 0 ≤ `WidthIn` ≤ 32767

▷ **`HeightIn`** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des virtuellen Eingabebildes.
Defaultwert: 512
Wertevorschläge: `HeightIn` ∈ {240, 256, 480, 512, 600, 1024}
Typischer Wertebereich: 0 ≤ `HeightIn` ≤ 32767

▷ **`Width`** (input_control) . . . . . extent.x ⇝ *integer*
Breite des virtuellen Zielbildes.
Defaultwert: 512
Wertevorschläge: `Width` ∈ {256, 320, 512, 640, 800, 1024}
Typischer Wertebereich: 0 ≤ `Width` ≤ 32767

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des virtuellen Zielbildes.
Defaultwert: 512
Wertevorschläge: Height ∈ {240, 256, 480, 512, 600, 1024}
Typischer Wertebereich: 0 ≤ Height ≤ 32767

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Siehe auch

polar_trans_image_ext, polar_trans_image_inv, polar_trans_region, polar_trans_region_inv, polar_trans_contour_xld

Modul

Foundation

projective_trans_contour_xld (
Contours : ContoursProjTrans : HomMat2D :)

Anwenden einer projektiven Transformation auf eine XLD-Kontur.

projective_trans_contour_xld wendet die durch die homogene Matrix HomMat2D beschriebene projektive Transformation auf die Konturen in Contours an und liefert die Ergebniskonturen in ContoursProjTrans zurück.

Zur Berechnung und Interpretation der Matrix siehe projective_trans_image.

Achtung

Der Transformation liegt das gleiche Koordinatensystem zugrunde, das auch bei dem Operator affine_trans_pixel verwendet wird. Das bedeutet, dass die Transformation nicht mit der Transformationsmatrix HomMat2D, sondern mit einer modifizierte Version davon durchgeführt wird. Eine Transformation mit projective_trans_contour_xld entspricht für jeden Punkt (Row_i, Col_i) der Kontur der folgenden Kette von Transformationen (Ein- und Ausgabekoordinaten als homogene Vektoren dargestellt):

$$\begin{pmatrix} RowTrans_i \\ ColTrans_i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & -0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \texttt{HomMat2D} \cdot \begin{bmatrix} 1 & 0 & +0.5 \\ 0 & 1 & +0.5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} Row_i \\ Col_i \\ 1 \end{pmatrix}$$

Dies kann zu unerwarteten Ergebnissen führen, wenn die Transformationsmatrix unter Verwendung von Koordinaten erstellt wurde, die von der Kontur mit Operatoren wie z.B. area_center_xld abgeleitet wurden. Wenn z.B. eine rotationssymmetrische XLD Kontur unter Verwendung von hom_mat2d_rotate um ihren mit area_center_xld bestimmten Schwerpunkt rotiert wird, so liegt die transformierte Kontur nicht auf der Originalkontur. Dieser Effekt kann dadurch kompensiert werden, dass vor dem Aufruf von projective_trans_contour_xld die Transformationsmatrix HomMat2D durch die beiden folgenden Translationen modifiziert wird:

```
hom_mat2d_translate(HomMat2D, 0.5, 0.5, HomMat2DTmp)
hom_mat2d_translate_local(HomMat2DTmp, -0.5, -0.5, HomMat2DAdapted)
projective_trans_contour_xld(Contours, ContoursAffineTrans,
HomMat2DAdapted)
```

Ein Überblick zu den verschiedenen in HALCON verwendeten 2D Koordinatensystemen ist in der Einleitung zum Kapitel Transformationen / 2D-Transformationen gegeben.

Parameter

▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
Eingabekonturen.

▷ **ContoursProjTrans** (output_object) ..... xld_cont(-array) ⇝ *object*
Ausgabekonturen.

▷ **HomMat2D** (input_control) ..... hom_mat2d ⇝ *real*
Homogene projektive Transformationsmatrix.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

vector_to_proj_hom_mat2d, hom_vector_to_proj_hom_mat2d, proj_match_points_ransac, proj_match_points_ransac_guided, hom_mat3d_project

Siehe auch

projective_trans_image, projective_trans_image_size, projective_trans_region, projective_trans_point_2d, projective_trans_pixel

Modul

Foundation

29.3 Mengen-Operationen

```
difference_closed_contours_xld ( Contours,
    Sub : ContoursDifference : : )
```

Berechnen der Differenzmenge von geschlossenen Konturen.

Der Operator difference_closed_contours_xld berechnet die Differenzmenge der von den geschlossenen Eingabekonturen umgebenen Regionen. Der Rand dieser Differenzmenge wird in ContoursDifference zurückgegeben.

Wenn die einzelnen Eingabekonturen nicht geschlossen sind, werden sie automatisch durch Verbinden von Anfangs- und Endpunkt geschlossen.

Intern werden zuerst die Regionen, die von den Konturen Contours bzw. Sub umschlossen sind, getrennt voneinander bestimmt (siehe unten). Danach wird die Differenzmenge der beiden sich ergebenden Regionen berechnet.

Im folgenden werden die beiden Eingabekonturen als *Menge von Rändern* der zu schneidenden Regionen betrachtet. Jede dieser Mengen besteht aus einer beliebigen Anzahl von Rändern (die einzelnen Konturen aus Contours bzw. Sub). Jeder Rand kann konvex oder konkav sein und sich selbst schneiden. Löcher können durch Verschachtelung der Ränder gebildet werden.

Die von allen Rändern einer solchen Menge von Rändern umschlossene Region ist durch die sogenannte Even-Odd-Rule definiert. Sie besteht also aus der Menge aller Punkte, die folgende Eigenschaft besitzen: Die Strecke, die man durch Verbinden des Punktes mit einem außerhalb der Region liegenden Referenzpunkt erhält, schneidet eine ungerade Anzahl von Rändern.

Anschaulich bedeutet dies, dass ein Rand, der vollständig innerhalb eines anderen Randes liegt, ein Loch in der Region erzeugt, die durch den ersten Rand umschlossen ist. Analog dazu wird auch der Überlappungsbereich zweier Regionen als "Loch"betrachtet, d.h., er gehört nicht zu der Region, die von den beiden Rändern umschlossen wird. Die Region, die von einem Rand umschlossen wird, ist dabei unabhängig von der Umlaufrichtung des Randes.

Ein sich selbst schneidender Rand kann entweder am Schnittpunkt aufgetrennt werden oder er wird so angeordnet, dass er sich selbst am Schnittpunkt berührt. Die sich ergebenden Ränder werden als unabhängige Ränder betrachtet.

Die resultierenden Konturen ContoursDifference werden automatisch in Konturen, die Regionen umschließen, und Konturen, die Löcher umschließen, klassifiziert. Diese Information wird im globalen Attribut *'is_hole'*

gespeichert. Für Konturen, die Löcher umschließen, wird das globale Attribut *'is_hole'* auf *1* gesetzt, ansonsten auf *0*.

Aufeinanderfolgende Punkte, die sich in Zeilen- und Spaltenrichtung um weniger als 1e-06 Pixel unterscheiden, werden als ein Punkt angesehen.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die Region umschließen, von der die zweite Region abgezogen wird.
- ▷ **Sub** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die Region umschließen, die von der ersten Region abgezogen wird.
- ▷ **ContoursDifference** (output_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die Differenzmenge umschließen.

Ergebnis

`difference_closed_contours_xld` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contour_region_xld`, `zero_crossing_sub_pix`, `threshold_sub_pix`

Alternativen

`difference`

Siehe auch

`intersection_closed_contours_xld`, `symm_difference_closed_contours_xld`, `union2_closed_contours_xld`, `difference_closed_polygons_xld`

Modul

Foundation

difference_closed_polygons_xld (Polygons, Sub : PolygonsDifference : :)

Berechnen der Differenzmenge von geschlossenen Polygonen.

Der Operator `difference_closed_polygons_xld` berechnet die Differenzmenge der von den geschlossenen Eingabepolygonen umgebenen Regionen. Der Rand dieser Differenzmenge wird in `PolygonsDifference` zurückgegeben.

Wenn die einzelnen Eingabepolygone nicht geschlossen sind, werden sie automatisch durch Verbinden von Anfangs- und Endpunkt geschlossen.

Intern werden zuerst die Regionen, die von den Polygonen `Polygons` bzw. `Sub` umschlossen sind, getrennt voneinander bestimmt (siehe unten). Danach wird die Differenzmenge der beiden sich ergebenden Regionen berechnet.

Im folgenden werden die beiden Eingabepolygone als *Menge von Rändern* der zu schneidenden Regionen betrachtet. Jede dieser Mengen besteht aus einer beliebigen Anzahl von Rändern (die einzelnen Polygone aus `Polygons` bzw. `Sub`). Jeder Rand kann konvex oder konkav sein und sich selbst schneiden. Löcher können durch Verschachtelung der Ränder gebildet werden.

Die von allen Rändern einer solchen Menge von Rändern umschlossene Region ist durch die sogenannte Even-Odd-Rule definiert. Sie besteht also aus der Menge aller Punkte, die folgende Eigenschaft besitzen: Die Strecke, die man durch Verbinden des Punktes mit einem außerhalb der Region liegenden Referenzpunkt erhält, schneidet eine ungerade Anzahl von Rändern.

Anschaulich bedeutet dies, dass ein Rand, der vollständig innerhalb eines anderen Randes liegt, ein Loch in der Region erzeugt, die durch den ersten Rand umschlossen ist. Analog dazu wird auch der Überlappungsbereich zweier

Regionen als "Loch"betrachtet, d.h., er gehört nicht zu der Region, die von den beiden Rändern umschlossen wird. Die Region, die von einem Rand umschlossen wird, ist dabei unabhängig von der Umlaufrichtung des Randes.

Ein sich selbst schneidender Rand kann entweder am Schnittpunkt aufgetrennt werden oder er wird so angeordnet, dass er sich selbst am Schnittpunkt berührt. Die sich ergebenden Ränder werden als unabhängige Ränder betrachtet.

Achtung

Die Ergebnispolygone `PolygonsDifference` enthalten keine Referenzen auf die möglicherweise von `Polygons` und `Sub` referenzierten XLD-Konturen. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen, z.B. `split_contours_xld`, zu Fehlermeldungen kommen.

Parameter

- ▷ **`Polygons`** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die Region umschließen, von der die zweite Region abgezogen wird.
- ▷ **`Sub`** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die Region umschließen, die von der ersten Region abgezogen wird.
- ▷ **`PolygonsDifference`** (output_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die Differenzmenge umschließen.

Ergebnis

`difference_closed_polygons_xld` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_polygons_xld`

Alternativen

`difference`

Siehe auch

`intersection_closed_polygons_xld`, `symm_difference_closed_polygons_xld`, `union2_closed_polygons_xld`, `difference_closed_contours_xld`

Modul

Foundation

`intersection_closed_contours_xld` `( Contours1, Contours2 : ContoursIntersection : : )`

Berechnen der Durchschnittsmenge von geschlossenen Konturen.

Der Operator `intersection_closed_contours_xld` berechnet die Durchschnittsmenge der von den geschlossenen Eingabekonturen umgebenen Regionen. Der Rand dieser Durchschnittsmenge wird in `ContoursIntersection` zurückgegeben.

Wenn die einzelnen Eingabekonturen nicht geschlossen sind, werden sie automatisch durch Verbinden von Anfangs- und Endpunkt geschlossen.

Intern werden zuerst die Regionen, die von den Konturen `Contours1` bzw. `Contours2` umschlossen sind, getrennt voneinander bestimmt (siehe unten). Danach wird die Durchschnittsmenge der beiden sich ergebenden Regionen berechnet.

Im folgenden werden die beiden Eingabekonturen als *Menge von Rändern* der zu schneidenden Regionen betrachtet. Jede dieser Mengen besteht aus einer beliebigen Anzahl von Rändern (die einzelnen Konturen aus `Contours1` bzw. `Contours2`). Jeder Rand kann konvex oder konkav sein und sich selbst schneiden. Löcher können durch Verschachtelung der Ränder gebildet werden.

Die von allen Rändern einer solchen Menge von Rändern umschlossene Region ist durch die sogenannte Even-Odd-Rule definiert. Sie besteht also aus der Menge aller Punkte, die folgende Eigenschaft besitzen: Die Strecke,

die man durch Verbinden des Punktes mit einem außerhalb der Region liegenden Referenzpunkt erhält, schneidet eine ungerade Anzahl von Rändern.

Anschaulich bedeutet dies, dass ein Rand, der vollständig innerhalb eines anderen Randes liegt, ein Loch in der Region erzeugt, die durch den ersten Rand umschlossen ist. Analog dazu wird auch der Überlappungsbereich zweier Regionen als "Loch"betrachtet, d.h., er gehört nicht zu der Region, die von den beiden Rändern umschlossen wird. Die Region, die von einem Rand umschlossen wird, ist dabei unabhängig von der Umlaufrichtung des Randes.

Ein sich selbst schneidender Rand kann entweder am Schnittpunkt aufgetrennt werden oder er wird so angeordnet, dass er sich selbst am Schnittpunkt berührt. Die sich ergebenden Ränder werden als unabhängige Ränder betrachtet.

Die resultierenden Konturen `ContoursIntersection` werden automatisch in Konturen, die Regionen umschließen, und Konturen, die Löcher umschließen, klassifiziert. Diese Information wird im globalen Attribut *'is_hole'* gespeichert. Für Konturen, die Löcher umschließen, wird das globale Attribut *'is_hole'* auf *1* gesetzt, ansonsten auf *0*.

Aufeinanderfolgende Punkte, die sich in Zeilen- und Spaltenrichtung um weniger als 1e-06 Pixel unterscheiden, werden als ein Punkt angesehen.

Parameter

- ▷ **Contours1** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die erste zu schneidende Region umschließen.
- ▷ **Contours2** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die zweite zu schneidende Region umschließen.
- ▷ **ContoursIntersection** (output_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die Durchschnittsmenge umschließen.

Ergebnis

`intersection_closed_contours_xld` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contour_region_xld`, `zero_crossing_sub_pix`, `threshold_sub_pix`

Alternativen

`intersection`

Siehe auch

`difference_closed_contours_xld`, `symm_difference_closed_contours_xld`, `union2_closed_contours_xld`, `intersection_closed_polygons_xld`

Modul

Foundation

intersection_closed_polygons_xld (Polygons1, Polygons2 : PolygonsIntersection : :)

Berechnen der Durchschnittsmenge von geschlossenen Polygonen.

Der Operator `intersection_closed_polygons_xld` berechnet die Durchschnittsmenge der von den geschlossenen Eingabepolygonen umgebenen Regionen. Der Rand dieser Durchschnittsmenge wird in `PolygonsIntersection` zurückgegeben.

Wenn die einzelnen Eingabepolygone nicht geschlossen sind, werden sie automatisch durch Verbinden von Anfangs- und Endpunkt geschlossen.

Intern werden zuerst die Regionen, die von den Polygonen `Polygons1` bzw. `Polygons2` umschlossen sind, getrennt voneinander bestimmt (siehe unten). Danach wird die Durchschnittsmenge der beiden sich ergebenden Regionen berechnet.

Im folgenden werden die beiden Eingabepolygone als *Menge von Rändern* der zu schneidenden Regionen betrachtet. Jede dieser Mengen besteht aus einer beliebigen Anzahl von Rändern (die einzelnen Polygonen aus `Polygons1` bzw. `Polygons2`). Jeder Rand kann konvex oder konkav sein und sich selbst schneiden. Löcher können durch Verschachtelung der Ränder gebildet werden.

Die von allen Rändern einer solchen Menge von Rändern umschlossene Region ist durch die sogenannte Even-Odd-Rule definiert. Sie besteht also aus der Menge aller Punkte, die folgende Eigenschaft besitzen: Die Strecke, die man durch Verbinden des Punktes mit einem außerhalb der Region liegenden Referenzpunkt erhält, schneidet eine ungerade Anzahl von Rändern.

Anschaulich bedeutet dies, dass ein Rand, der vollständig innerhalb eines anderen Randes liegt, ein Loch in der Region erzeugt, die durch den ersten Rand umschlossen ist. Analog dazu wird auch der Überlappungsbereich zweier Regionen als "Loch"betrachtet, d.h., er gehört nicht zu der Region, die von den beiden Rändern umschlossen wird. Die Region, die von einem Rand umschlossen wird, ist dabei unabhängig von der Umlaufrichtung des Randes.

Ein sich selbst schneidender Rand kann entweder am Schnittpunkt aufgetrennt werden oder er wird so angeordnet, dass er sich selbst am Schnittpunkt berührt. Die sich ergebenden Ränder werden als unabhängige Ränder betrachtet.

Achtung

Die Ergebnispolygone `PolygonsIntersection` enthalten keine Referenzen auf die möglicherweise von `Polygons1` und `Polygons2` referenzierten XLD-Konturen. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen, z.B. `split_contours_xld`, zu Fehlermeldungen kommen.

Parameter

- ▷ **`Polygons1`** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die erste zu schneidende Region umschließen.
- ▷ **`Polygons2`** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die zweite zu schneidende Region umschließen.
- ▷ **`PolygonsIntersection`** (output_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die Durchschnittsmenge umschließen.

Ergebnis

`intersection_closed_polygons_xld` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_polygons_xld`

Alternativen

`intersection`

Siehe auch

`difference_closed_polygons_xld`, `symm_difference_closed_polygons_xld`, `union2_closed_polygons_xld`, `intersection_closed_contours_xld`

Modul

Foundation

`symm_difference_closed_contours_xld` (Contours1, Contours2 : ContoursDifference : :)

Berechnen der symmetrischen Differenzmenge von geschlossenen Konturen.

Der Operator `symm_difference_closed_contours_xld` berechnet die symmetrische Differenzmenge der von den geschlossenen Eingabekonturen umgebenen Regionen. Der Rand dieser Differenzmenge wird in `ContoursDifference` zurückgegeben.

Wenn die einzelnen Eingabekonturen nicht geschlossen sind, werden sie automatisch durch Verbinden von Anfangs- und Endpunkt geschlossen.

Intern werden zuerst die Regionen, die von den Konturen `Contours1` bzw. `Contours2` umschlossen sind, getrennt voneinander bestimmt (siehe unten). Danach wird die symmetrische Differenzmenge der beiden sich ergebenden Regionen berechnet.

Im folgenden werden die beiden Eingabekonturen als *Menge von Rändern* der zu schneidenden Regionen betrachtet. Jede dieser Mengen besteht aus einer beliebigen Anzahl von Rändern (die einzelnen Konturen aus `Contours1` bzw. `Contours2`). Jeder Rand kann konvex oder konkav sein und sich selbst schneiden. Löcher können durch Verschachtelung der Ränder gebildet werden.

Die von allen Rändern einer solchen Menge von Rändern umschlossene Region ist durch die sogenannte Even-Odd-Rule definiert. Sie besteht also aus der Menge aller Punkte, die folgende Eigenschaft besitzen: Die Strecke, die man durch Verbinden des Punktes mit einem außerhalb der Region liegenden Referenzpunkt erhält, schneidet eine ungerade Anzahl von Rändern.

Anschaulich bedeutet dies, dass ein Rand, der vollständig innerhalb eines anderen Randes liegt, ein Loch in der Region erzeugt, die durch den ersten Rand umschlossen ist. Analog dazu wird auch der Überlappungsbereich zweier Regionen als "Loch"betrachtet, d.h., er gehört nicht zu der Region, die von den beiden Rändern umschlossen wird. Die Region, die von einem Rand umschlossen wird, ist dabei unabhängig von der Umlaufrichtung des Randes.

Ein sich selbst schneidender Rand kann entweder am Schnittpunkt aufgetrennt werden oder er wird so angeordnet, dass er sich selbst am Schnittpunkt berührt. Die sich ergebenden Ränder werden als unabhängige Ränder betrachtet.

Die resultierenden Konturen `ContoursDifference` werden automatisch in Konturen, die Regionen umschließen, und Konturen, die Löcher umschließen, klassifiziert. Diese Information wird im globalen Attribut *'is_hole'* gespeichert. Für Konturen, die Löcher umschließen, wird das globale Attribut *'is_hole'* auf *1* gesetzt, ansonsten auf *0*.

Aufeinanderfolgende Punkte, die sich in Zeilen- und Spaltenrichtung um weniger als 1e-06 Pixel unterscheiden, werden als ein Punkt angesehen.

Parameter

- ▷ **Contours1** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die erste Region umschließen.
- ▷ **Contours2** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die zweite Region umschließen.
- ▷ **ContoursDifference** (output_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die symmetrische Differenzmenge umschließen.

Ergebnis

`symm_difference_closed_contours_xld` liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contour_region_xld`, `zero_crossing_sub_pix`, `threshold_sub_pix`

Alternativen

`symm_difference`

Siehe auch

`intersection_closed_contours_xld`, `difference_closed_contours_xld`, `union2_closed_contours_xld`, `symm_difference_closed_polygons_xld`

Modul

Foundation

symm_difference_closed_polygons_xld (Polygons1, Polygons2 : PolygonsDifference : :)

Berechnen der symmetrischen Differenzmenge von geschlossenen Polygonen.

Der Operator symm_difference_closed_polygons_xld berechnet die symmetrische Differenzmenge der von den geschlossenen Eingabepolygonen umgebenen Regionen. Der Rand dieser symmetrische Differenzmenge wird in PolygonsDifference zurückgegeben.

Wenn die einzelnen Eingabepolygone nicht geschlossen sind, werden sie automatisch durch Verbinden von Anfangs- und Endpunkt geschlossen.

Intern werden zuerst die Regionen, die von den Polygonen Polygons1 bzw. Polygons2 umschlossen sind, getrennt voneinander bestimmt (siehe unten). Danach wird die symmetrische Differenzmenge der beiden sich ergebenden Regionen berechnet.

Im folgenden werden die beiden Eingabepolygone als *Menge von Rändern* der zu schneidenden Regionen betrachtet. Jede dieser Mengen besteht aus einer beliebigen Anzahl von Rändern (die einzelnen Polygone aus Polygons1 bzw. Polygons2). Jeder Rand kann konvex oder konkav sein und sich selbst schneiden. Löcher können durch Verschachtelung der Ränder gebildet werden.

Die von allen Rändern einer solchen Menge von Rändern umschlossene Region ist durch die sogenannte Even-Odd-Rule definiert. Sie besteht also aus der Menge aller Punkte, die folgende Eigenschaft besitzen: Die Strecke, die man durch Verbinden des Punktes mit einem außerhalb der Region liegenden Referenzpunkt erhält, schneidet eine ungerade Anzahl von Rändern.

Anschaulich bedeutet dies, dass ein Rand, der vollständig innerhalb eines anderen Randes liegt, ein Loch in der Region erzeugt, die durch den ersten Rand umschlossen ist. Analog dazu wird auch der Überlappungsbereich zweier Regionen als "Loch"betrachtet, d.h., er gehört nicht zu der Region, die von den beiden Rändern umschlossen wird. Die Region, die von einem Rand umschlossen wird, ist dabei unabhängig von der Umlaufrichtung des Randes.

Ein sich selbst schneidender Rand kann entweder am Schnittpunkt aufgetrennt werden oder er wird so angeordnet, dass er sich selbst am Schnittpunkt berührt. Die sich ergebenden Ränder werden als unabhängige Ränder betrachtet.

Achtung

Die Ergebnispolygone PolygonsDifference enthalten keine Referenzen auf die möglicherweise von Polygons1 und Polygons2 referenzierten XLD-Konturen. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen, z.B. split_contours_xld, zu Fehlermeldungen kommen.

Parameter

- ▷ **Polygons1** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die erste Region umschließen.
- ▷ **Polygons2** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die zweite Region umschließen.
- ▷ **PolygonsDifference** (output_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die symmetrische Differenzmenge umschließen.

Ergebnis

symm_difference_closed_polygons_xld liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_polygons_xld

Alternativen

symm_difference

Siehe auch

intersection_closed_polygons_xld, difference_closed_polygons_xld, union2_closed_polygons_xld, symm_difference_closed_contours_xld

Modul

Foundation

union2_closed_contours_xld (Contours1, Contours2 : ContoursUnion : :)

Berechnen der Vereinigungsmenge von geschlossenen Konturen.

Der Operator union2_closed_contours_xld berechnet die Vereinigungsmenge der von den geschlossenen Eingabekonturen umgebenen Regionen. Der Rand dieser Vereinigungsmenge wird in ContoursUnion zurückgegeben.

Wenn die einzelnen Eingabekonturen nicht geschlossen sind, werden sie automatisch durch Verbinden von Anfangs- und Endpunkt geschlossen.

Intern werden zuerst die Regionen, die von den Konturen Contours1 bzw. Contours2 umschlossen sind, getrennt voneinander bestimmt (siehe unten). Danach wird die Vereinigungsmenge der beiden sich ergebenden Regionen berechnet.

Im folgenden werden die beiden Eingabekonturen als *Menge von Rändern* der zu schneidenden Regionen betrachtet. Jede dieser Mengen besteht aus einer beliebigen Anzahl von Rändern (die einzelnen Konturen aus Contours1 bzw. Contours2). Jeder Rand kann konvex oder konkav sein und sich selbst schneiden. Löcher können durch Verschachtelung der Ränder gebildet werden.

Die von allen Rändern einer solchen Menge von Rändern umschlossene Region ist durch die sogenannte Even-Odd-Rule definiert. Sie besteht also aus der Menge aller Punkte, die folgende Eigenschaft besitzen: Die Strecke, die man durch Verbinden des Punktes mit einem außerhalb der Region liegenden Referenzpunkt erhält, schneidet eine ungerade Anzahl von Rändern.

Anschaulich bedeutet dies, dass ein Rand, der vollständig innerhalb eines anderen Randes liegt, ein Loch in der Region erzeugt, die durch den ersten Rand umschlossen ist. Analog dazu wird auch der Überlappungsbereich zweier Regionen als "Loch"betrachtet, d.h., er gehört nicht zu der Region, die von den beiden Rändern umschlossen wird. Die Region, die von einem Rand umschlossen wird, ist dabei unabhängig von der Umlaufrichtung des Randes.

Ein sich selbst schneidender Rand kann entweder am Schnittpunkt aufgetrennt werden oder er wird so angeordnet, dass er sich selbst am Schnittpunkt berührt. Die sich ergebenden Ränder werden als unabhängige Ränder betrachtet.

Die resultierenden Konturen ContoursUnion werden automatisch in Konturen, die Regionen umschließen, und Konturen, die Löcher umschließen, klassifiziert. Diese Information wird im globalen Attribut *'is_hole'* gespeichert. Für Konturen, die Löcher umschließen, wird das globale Attribut *'is_hole'* auf *1* gesetzt, ansonsten auf *0*.

Aufeinanderfolgende Punkte, die sich in Zeilen- und Spaltenrichtung um weniger als 1e-06 Pixel unterscheiden, werden als ein Punkt angesehen.

Parameter

- ▷ **Contours1** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die erste Region umschließen.
- ▷ **Contours2** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die zweite Region umschließen.
- ▷ **ContoursUnion** (output_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die die Vereinigungsmenge umschließen.

Ergebnis

union2_closed_contours_xld liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contour_region_xld, zero_crossing_sub_pix, threshold_sub_pix

Alternativen

union2

Siehe auch

difference_closed_contours_xld, symm_difference_closed_contours_xld, intersection_closed_contours_xld, union2_closed_polygons_xld

Modul

Foundation

union2_closed_polygons_xld (Polygons1, Polygons2 : PolygonsUnion : :)

Berechnen der Vereinigungsmenge von geschlossenen Polygonen.

Der Operator union2_closed_polygons_xld berechnet die Vereinigungsmenge der von den geschlossenen Eingabepolygonen umgebenen Regionen. Der Rand dieser Vereinigungsmenge wird in PolygonsUnion zurückgegeben.

Wenn die einzelnen Eingabepolygone nicht geschlossen sind, werden sie automatisch durch Verbinden von Anfangs- und Endpunkt geschlossen.

Intern werden zuerst die Regionen, die von den Polygonen Polygons1 bzw. Polygons2 umschlossen sind, getrennt voneinander bestimmt (siehe unten). Danach wird die Vereinigungsmenge der beiden sich ergebenden Regionen berechnet.

Im folgenden werden die beiden Eingabepolygone als *Menge von Rändern* der zu schneidenden Regionen betrachtet. Jede dieser Mengen besteht aus einer beliebigen Anzahl von Rändern (die einzelnen Polygonen aus Polygons1 bzw. Polygons2). Jeder Rand kann konvex oder konkav sein und sich selbst schneiden. Löcher können durch Verschachtelung der Ränder gebildet werden.

Die von allen Rändern einer solchen Menge von Rändern umschlossene Region ist durch die sogenannte Even-Odd-Rule definiert. Sie besteht also aus der Menge aller Punkte, die folgende Eigenschaft besitzen: Die Strecke, die man durch Verbinden des Punktes mit einem außerhalb der Region liegenden Referenzpunkt erhält, schneidet eine ungerade Anzahl von Rändern.

Anschaulich bedeutet dies, dass ein Rand, der vollständig innerhalb eines anderen Randes liegt, ein Loch in der Region erzeugt, die durch den ersten Rand umschlossen ist. Analog dazu wird auch der Überlappungsbereich zweier Regionen als "Loch"betrachtet, d.h., er gehört nicht zu der Region, die von den beiden Rändern umschlossen wird. Die Region, die von einem Rand umschlossen wird, ist dabei unabhängig von der Umlaufrichtung des Randes.

Ein sich selbst schneidender Rand kann entweder am Schnittpunkt aufgetrennt werden oder er wird so angeordnet, dass er sich selbst am Schnittpunkt berührt. Die sich ergebenden Ränder werden als unabhängige Ränder betrachtet.

Achtung

Die Ergebnispolygone PolygonsUnion enthalten keine Referenzen auf die möglicherweise von Polygons1 und Polygons2 referenzierten XLD-Konturen. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen, z.B. split_contours_xld, zu Fehlermeldungen kommen.

Parameter

- ▷ **Polygons1** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die erste Region umschließen.
- ▷ **Polygons2** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die zweite Region umschließen.
- ▷ **PolygonsUnion** (output_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, die die Vereinigungsmenge umschließen.

Ergebnis

union2_closed_polygons_xld liefert den Wert TRUE wenn alle Parameter korrekt sind. Andernfalls wird eine Fehlerbehandlung durchgeführt

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_polygons_xld

Alternativen

union2

Siehe auch

difference_closed_polygons_xld, symm_difference_closed_polygons_xld, intersection_closed_polygons_xld, union2_closed_contours_xld

Modul

Foundation

29.4 Merkmale

area_center_points_xld (XLD : : : Area, Row, Column)

Fläche und Schwerpunkt von Konturen oder Polygonen, die als Punktwolken betrachtet werden.

area_center_points_xld berechnet die Fläche und den Schwerpunkt der von den Eingabekonturen oder -polygonen XLD definierten Punktwolken (d.h. die Reihenfolge der Kontur- oder Polygonpunkte wird nicht berücksichtigt). Die Fläche ist im Falle der Punktwolke gegeben durch die Anzahl der Punkte. Der Schwerpunkt entspricht dem arithmetischen Mittel der Punkte. Bei geschlossenen Konturen oder Polygonen (Anfangspunkt = Endpunkt) wird der Endpunkt der Kontur oder des Polygons nicht berücksichtigt, da er sonst doppeltes Gewicht erhielte.

area_center_points_xld sollte dann verwendet werden, wenn die Kontur XLD entweder nicht kreuzungsfrei ist oder nicht kreuzungsfrei durch eine Line vom End- zum Anfangspunkt geschlossen werden kann. In diesem Fall liefert area_center_xld keine sinnvollen Ergebnisse. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der gleichen Reihenfolge wie die entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Achtung

Auch wenn sich die Eingabekontur oder das -polygon XLD nicht selbst überschneidet unterscheidet sich das Ergebnis von area_center_points_xld deutlich von dem Ergebnis, das durch area_center_xld erzielt werden würde, da das Ergebnis aus der Punktwolke der Kontur und nicht aus der umschlossenen Region berechnet wird.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Punktwolken als Konturen bzw. Polygone.
- ▷ **Area** (output_control) ..... real(-array) ⇝ *real*
  Fläche der Punktwolke.
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenindex des Schwerpunktes.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

area_center_points_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Alternativen

area_center_xld

Siehe auch

moments_points_xld, moments_any_points_xld, area_center, moments_region_2nd

Modul

Foundation

area_center_xld (XLD : : : Area, Row, Column, PointOrder)

Fläche und Schwerpunkt von Konturen oder Polygonen.

area_center_xld berechnet die Fläche und den Schwerpunkt der von den Eingabekonturen oder -polygonen XLD eingeschlossenen Flächen sowie den mathematischen Umlaufsinn der entsprechenden Stützpunkte. Die Fläche und der Schwerpunkt wird durch Anwendung des Gaußschen Integralsatzes nur unter Verwendung der Kontur- oder Polygonpunkte berechnet, d.h. es wird keine explizite Fläche zur Berechnung erzeugt. Sind die Punkte einer Kontur bzw. eines Polygons entgegen dem Uhrzeigersinn angeordnet (d.h. im mathematisch positiven Sinn), ist PointOrder *'positive'*. Es wird vorausgesetzt, dass die Kontur bzw. das Polygon geschlossen ist. Ist das nicht der Fall, wird sie/es künstlich geschlossen.

Es ist zu beachten, dass area_center_xld nur dann sinnvolle Ergebnisse liefert, falls die Kontur oder das Polygon eine Fläche in der Ebene einschließt. Insbesondere darf sich die Kontur oder das Polygon nicht selbst schneiden. Dies ist insbesondere dann zu beachten, wenn offene Konturen und Polygone übergeben werden, da diese automatisch geschlossen werden, was dazu führen kann, dass sich die Kontur oder das Polygon selbst schneidet. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden. Falls sich die Kontur oder das Polygon selbst schneidet, kann ein sinnvoller Wert für den Schwerpunkt mit area_center_points_xld berechnet werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Area** (output_control) ..... real(-array) ⇝ *real*
  Fläche, die von der Kontur bzw. dem Polygon eingeschlossen wird.
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenindex des Schwerpunktes.
- ▷ **PointOrder** (output_control) ..... string(-array) ⇝ *string*
  Mathematischer Umlaufsinn ('positive'/'negative') der Stützpunkte.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

area_center_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Alternativen

area_center_points_xld

Siehe auch

moments_xld, moments_any_xld, area_center, moments_region_2nd

Modul

Foundation

circularity_xld (XLD : : : Circularity)

Formfaktor für die Kreisförmigkeit von Konturen bzw. Polygonen.

circularity_xld berechnet für jede Eingabekontur bzw. jedes Eingabepolygon die Ähnlichkeit mit einem Kreis. Es muss darauf geachtet werden, dass sich die Eingabekonturen bzw. -polygone nicht selbst schneiden, da ansonsten der Ausgabeparameter keinen aussagekräftigen Wert enthält (Ob sich die Eingabekonturen selbst schneiden oder nicht, lässt sich mit test_self_intersection_xld überprüfen). Ist eine Eingabekontur oder ein Eingabepolygon nicht geschlossen, so wird sie/es vor der Berechnung automatisch geschlossen.

Berechnung: Sei F die eingeschlossene Fläche der Kontur bzw. des Polygons und max der maximale Abstand vom Schwerpunkt zu allen Kontur- bzw. Polygonpunkten, dann ist der Formfaktor Circularity definiert als:

$$\texttt{Circularity} = \frac{F}{(max^2 * \pi)}$$

Der Formfaktor Circularity ist bei einem Kreis gleich 1. Wenn die Kontur oder das Polygon langgestreckt ist, ist Circularity kleiner als 1. circularity_xld spricht besonders auf große Ausbuchtungen an.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden ihre Formfaktoren in einem Tupel in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen oder Polygone.

▷ **Circularity** (output_control) ..... real(-array) ⇝ *real*
Rundheit der Eingabekonturen bzw. -polygone.
Zusicherung: 0 <= Circularity && Circularity <= 1.0

Ergebnis

circularity_xld liefert den Wert TRUE, falls die Eingabekontur nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Alternativen

compactness_xld, convexity_xld, eccentricity_xld, rectangularity_xld

Siehe auch

area_center_xld, select_shape_xld

Modul

Foundation

compactness_xld (XLD : : : Compactness)

Formfaktor für die Kompaktheit von Konturen oder Polygonen.

compactness_xld berechnet die Kompaktheit für jede Eingabekontur bzw. jedes Eingabepolygon in XLD. Es muss darauf geachtet werden, dass sich die Eingabekonturen bzw. -polygone nicht selbst schneiden, da ansonsten der Ausgabeparameter keinen aussagekräftigen Wert enthält (Ob sich die Eingabekonturen bzw. -polygone selbst schneiden oder nicht, lässt sich mit test_self_intersection_xld überprüfen). Sind die Eingabekonturen bzw. -polygone nicht geschlossen, so werden sie vor der Berechnung automatisch geschlossen.

Berechnung: Sei L die Länge der Kontur/des Polygons und F die eingeschlossene Fläche, dann ist der Formfaktor Compactness definiert als:

$$\text{Compactness} = \frac{L^2}{4F\pi}$$

Der Formfaktor Compactness ist bei einem Kreis gleich 1. Ist die Kontur bzw. das Polygon langgestreckt, dann ist Compactness größer als 1. compactness_xld spricht auf den Verlauf der Kontur/des Polygons (Rauigkeit) an. Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden ihre Formfaktoren in einem Tupel in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Compactness** (output_control) ..... real(-array) ⇝ *real*
  Kompaktheit der Eingabekonturen bzw. Polygone.
  Zusicherung: Compactness >= 1.0 || Compactness == 0

Ergebnis

compactness_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Alternativen

circularity_xld, convexity_xld, eccentricity_xld, rectangularity_xld

Siehe auch

area_center_xld, select_shape_xld

Modul

Foundation

contour_point_num_xld (Contour : : : Length)

Ausgeben der Anzahl von Stützpunkten einer XLD-Kontur.

Mit contour_point_num_xld wird in Length die Länge der Kontur Contour, d.h. die Anzahl der Konturpunkte, ausgegeben.

Parameter

▷ **Contour** (input_object) ..... xld_cont(-array) ⇝ *object*
Eingabe-XLD-Kontur.

▷ **Length** (output_control) ..... integer(-array) ⇝ *integer*
Anzahl der Stützpunkte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Nachfolger

`get_contour_xld`, `get_contour_attrib_xld`

Alternativen

`get_regress_params_xld`

Siehe auch

`query_contour_attribs_xld`

Modul

Foundation

convexity_xld (XLD : : : Convexity)

Formfaktor für die Konvexität von Konturen oder Polygonen.

`convexity_xld` berechnet für jede Eingabekontur bzw. jedes Eingabepolygon die Konvexität. Es muss darauf geachtet werden, dass sich die Eingabekonturen bzw. Eingabepolygone nicht selbst schneiden, da ansonsten die Ausgabeparameter keinen aussagekräftigen Wert enthalten (Ob sich die Eingabekonturen bzw. Eingabepolygone selbst schneiden oder nicht, lässt sich mit `test_self_intersection_xld` überprüfen). Sind die Eingabekonturen bzw. Eingabepolygone nicht geschlossen, so werden sie vor der Berechnung automatisch geschlossen.

Berechnung: Sei F_c die Fläche der konvexen Hülle und F_o die eingeschlossene Originalfläche der Kontur bzw. des Polygons, dann ist der Formfaktor `Convexity` definiert als:

$$\texttt{Convexity} = \frac{F_o}{F_c}$$

Der Formfaktor `Convexity` ist gleich 1, wenn die Kontur bzw. das Polygon konvex ist (z.B. Rechteck, Kreis etc.). Sind Einbuchtungen vorhanden, dann ist `Convexity` kleiner als 1.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Zahlenwerte des Formfaktors in einem Tupel in der Reihenfolge der entsprechenden Konturen bzw. Polygone in `XLD` abgespeichert.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen oder Polygone.

▷ **Convexity** (output_control) ..... real(-array) ⇝ *real*
Konvexität der Eingabekonturen bzw. -polygone.
Zusicherung: Convexity <= 1

Ergebnis

`convexity_xld` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Siehe auch

area_center_xld, select_shape_xld, shape_trans_xld

Modul

Foundation

diameter_xld (XLD : : : Row1, Column1, Row2, Column2, Diameter)

Maximaler Abstand zweier Punkte von Konturen oder Polygonen.

diameter_xld berechnet den maximalen Abstand zweier Punkte für jede Eingabekontur bzw. für jedes Eingabepolygon. Es werden die Koordinaten der beiden Extrempunkte und der Abstand zwischen diesen Punkten zurückgegeben. Es muss darauf geachtet werden, dass sich die Eingabekontur bzw. das Eingabepolygon nicht selbst schneidet, da ansonsten die Ausgabeparameter keine aussagekräftigen Werte enthalten (Ob sich die Eingabekontur bzw. das Eingabepolygon selbst schneidet oder nicht, lässt sich mit test_self_intersection_xld überprüfen). Ist die Eingabekontur bzw. das Eingabepolygon nicht geschlossen, so wird sie vor der Berechnung automatisch geschlossen.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Achtung

Bei leerer Kontur bzw. leerem Polygon kann das Ergebnis von Row1, Column1, Row2 und Column2 (alle sind 0) zu Konfusionen führen.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Row1** (output_control) ..... line.begin.y(-array) ⇝ *real*
  Zeilenkoordinate des ersten Extrempunktes der Konturen bzw. Polygone.
- ▷ **Column1** (output_control) ..... line.begin.x(-array) ⇝ *real*
  Spaltenkoordinate des ersten Extrempunktes der Konturen bzw. Polygone.
- ▷ **Row2** (output_control) ..... line.end.y(-array) ⇝ *real*
  Zeilenkoordinate des zweiten Extrempunktes der Konturen bzw. Polygone.
- ▷ **Column2** (output_control) ..... line.end.x(-array) ⇝ *real*
  Spaltenkoordinate des zweiten Extrempunktes der Konturen bzw. Polygone.
- ▷ **Diameter** (output_control) ..... number(-array) ⇝ *real*
  Abstand der beiden Extrempunkte der Konturen oder Polygone.

Ergebnis

diameter_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Alternativen

smallest_rectangle2_xld

Siehe auch

area_center_xld

Modul

Foundation

dist_ellipse_contour_points_xld (Contour : : DistanceMode, ClippingEndPoints, Row, Column, Phi, Radius1, Radius2 : Distances)

Berechnen der Abstände von allen Punkten einer Kontur zu einer Ellipse.

Der operator dist_ellipse_contour_points_xld bestimmt die Abstände zwischen Punkten einer Kontur Contour und einer Ellipse, welche durch ihren Mittelpunkt (Row, Column), die Orientierung Phi der großen Achse und den Länge der großen und kleinen Achse Radius1 beziehungsweise Radius2 definiert ist.

Das Distanzmaß ist der geometrische Abstand, welcher manchmal auch orthogonaler Abstand genannt wird. Die Abstände Distances werden für alle Punkte auf der Kontur berechnet. Normalerweise ist der Abstand ein positiver Wert; dann ist die Option DistanceMode gleich *'unsigned'* zu wählen. Wird dagegen die Option *'signed'* gewählt, so weisen die Abständen positives oder negatives Vorzeichen auf, je nachdem ob sie außerhalb oder innerhalb der Ellipse liegen.

Je nach Vorverarbeitung sind die Start- und Endpunkte einer Kontur nicht exakt bekannt. Dann wird der Operator fit_ellipse_contour_xld typischerweise mit einem Wert größer null für den Parameter ClippingEndPoints aufgerufen, um eine bestimmte Anzahl an Punkten an beiden Enden der Kontur für die Berechnung auszuschließen. Damit der geometrische Abstand für dieselben Konturpunkte wie in dem Ellipsen-Fitting-Prozess berechnet werden kann, ist für den Parameter ClippingEndPoints auch derselbe Wert zu wählen.

Parameter

- ▷ **Contour** (input_object) . . . . . xld_cont ⇝ *object*
  Eingabekonturen.
- ▷ **DistanceMode** (input_control) . . . . . string ⇝ *string*
  Modus für Distanzwerte mit oder ohne Vorzeichen.
  Defaultwert: 'unsigned'
  Werteliste: DistanceMode ∈ {'unsigned', 'signed'}
- ▷ **ClippingEndPoints** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Abstandsberechnung ignoriert werden sollen.
  Defaultwert: 0
  Restriktion: ClippingEndPoints >= 0
- ▷ **Row** (input_control) . . . . . ellipse.center.y ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Column** (input_control) . . . . . ellipse.center.x ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Phi** (input_control) . . . . . ellipse.angle.rad ⇝ *real*
  Orientierung der Hauptachse im Bogenmaß.
  Restriktion: Phi >= 0 && Phi <= 6.283185307
- ▷ **Radius1** (input_control) . . . . . ellipse.radius1 ⇝ *real*
  Länge der großen Halbachse.
  Restriktion: Radius1 > 0
- ▷ **Radius2** (input_control) . . . . . ellipse.radius2 ⇝ *real*
  Länge der kleinen Halbachse.
  Restriktion: Radius2 >= 0 && Radius2 <= Radius1
- ▷ **Distances** (output_control) . . . . . real-array ⇝ *real*
  Abstände der Konturpunkte zur Ellipse.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dist_ellipse_contour_points_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fit_ellipse_contour_xld`

Alternativen

`dist_ellipse_contour_xld`

Modul

Foundation

dist_ellipse_contour_xld (Contours : : Mode, MaxNumPoints, ClippingEndPoints, Row, Column, Phi, Radius1, Radius2 : MinDist, MaxDist, AvgDist, SigmaDist)

Berechnen des Abstands von Konturen zu einer Ellipse.

`dist_ellipse_contour_xld` berechnet den Abstand der Konturen in `Contours` zu einer Ellipse, die durch ihren Mittelpunkt (`Row`, `Column`), die Orientierung ihrer Hauptachse `Phi` und die Länge der großen `Radius1` bzw. kleinen Halbachse `Radius2` beschrieben wird. Zur Bestimmung des Abstandes eines Konturpunktes $X = (x_i, y_i)$ zur Ellipse stehen dabei folgende Verfahren, selektierbar über `Mode` zur Verfügung:

'geometric' Als Abstandsmaß wird die geometrische Distanz der Konturpunkte zur Ellipse verwendet. Dieses Abstandsmaß wird auch orthogonaler oder euklidischer Abstand genannt.

'algebraic' Als Abstandsmaß wird die algebraische Distanz $ax_i^2 + bx_iy_i + cy_i^2 + dx_i + ey_i + f$ verwendet, wobei die Ellipsenparameter a - f so normiert werden, dass der Mittelpunkt der Ellipse als Abstandswert die Länge der kleineren Halbachse `Radius2` erhält. Das Maß ist mit einem *high curvature bias* behaftet: In der Nähe von Ellipsenpunkten mit hoher Krümmung (insbesondere die Pole auf der Hauptachse) ist der Abstand kleiner als in der Nähe von Ellipsenpunkten mit geringer Krümmung (etwa die Pole auf der Nebenachse).

'focpoints' Als Abstandsmaß wird die Abweichung $XF_1 + XF_2 - 2a$ verwendet, wobei F_1,F_2 die beiden Brennpunkte der Ellipse bezeichnen und a für die Länge der großen Halbachse `Radius1` steht. Das Maß ist mit einem *low curvature bias* behaftet: In der Nähe von Ellipsenpunkten mit hoher Krümmung (insbesondere die Pole auf der Hauptachse) ist der Abstand größer als in der Nähe von Ellipsenpunkten mit geringer Krümmung (etwa die Pole auf der Nebenachse).

'bisec' Als Abstandsmaß wird die Entfernung des Konturpunkts vom Schnittpunkt der Winkelhalbierenden der beiden Geraden durch X und die beiden Brennpunkte mit der Ellipse verwendet. Dies ist eine gute Näherung an den Orthogonalabstand des Konturpunktes von der Ellipse. Die Genauigkeit der Näherung hängt sowohl vom Seitenverhältnis der Ellipse als auch von der Entfernung ab.

Zurückgegeben werden dann folgende statistische Werte über den Absolutbeträgen der einzelnen Abstände: Der minimale `MinDist` und maximale Abstand `MaxDist`, der mittlere Abstand `AvgDist` und die Standardabweichung des Abstandes `SigmaDist`.

Zur Reduktion des Aufwands lässt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für `MaxNumPoints` eine Zahl ungleich -1 übergeben, werden nur maximal `MaxNumPoints` gleichmäßig über die Kontur verteilte Punkte verwendet. Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung nicht exakt bekannt sein können, besteht die Möglichkeit, `ClippingEndPoints` Punkte am Anfang und Ende der Kontur von der Abstandsberechnung auszuschließen.

Parameter

- ▷ **Contours** (input_object) . . . . . xld_cont(-array) ⇝ *object*
  Eingabekonturen.
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Methode zur Abstandsbestimmung.
  Defaultwert: 'geometric'
  Werteliste: `Mode` ∈ {'geometric', 'algebraic', 'focpoints', 'bisec'}
- ▷ **MaxNumPoints** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl Konturpunkte für Berechnung (-1 für alle Punkte).
  Defaultwert: -1
  Restriktion: MaxNumPoints >= 3
- ▷ **ClippingEndPoints** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Abstandsberechnung ignoriert werden sollen.
  Defaultwert: 0
  Restriktion: ClippingEndPoints >= 0
- ▷ **Row** (input_control) . . . . . ellipse.center.y ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Column** (input_control) . . . . . ellipse.center.x ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Phi** (input_control) . . . . . ellipse.angle.rad ⇝ *real*
  Orientierung der Hauptachse im Bogenmaß.
  Restriktion: Phi >= 0 && Phi <= 6.283185307
- ▷ **Radius1** (input_control) . . . . . ellipse.radius1 ⇝ *real*
  Länge der großen Halbachse.
  Restriktion: Radius1 > 0
- ▷ **Radius2** (input_control) . . . . . ellipse.radius2 ⇝ *real*
  Länge der kleinen Halbachse.
  Restriktion: Radius2 >= 0 && Radius2 <= Radius1
- ▷ **MinDist** (output_control) . . . . . real(-array) ⇝ *real*
  Minimaler Abstand.
- ▷ **MaxDist** (output_control) . . . . . real(-array) ⇝ *real*
  Maximaler Abstand.
- ▷ **AvgDist** (output_control) . . . . . real(-array) ⇝ *real*
  Mittlerer Abstand.
- ▷ **SigmaDist** (output_control) . . . . . real(-array) ⇝ *real*
  Standardabweichung des Abstands.

Beispiel

```
read_image (Image, 'caltab')
find_caltab (Image, CalPlate, 'caltab_big.descr', 3, 112, 5)
reduce_domain (Image, CalPlate, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 20, 40)
select_contours_xld (Edges, EdgesClosed, 'closed', 0, 2.0, 0, 0)
select_contours_xld (EdgesClosed, EdgesMarks, 'contour_length', 20, 100, \
                     0, 0)
fit_ellipse_contour_xld (EdgesMarks, 'fitzgibbon', -1, 2, 0, 200, 3, 2.0, \
                         Row, Column, Phi, Radius1, Radius2, StartPhi, \
                         EndPhi, PointOrder)
for i := 0 to |Row|-1 by 1
    select_obj (EdgesMarks, ObjectSelected, i+1)
    dist_ellipse_contour_xld (ObjectSelected, 'bisec', -1, 0, Row[i], \
                              Column[i], Phi[i], Radius1[i], Radius2[i], \
                              MinDist, MaxDist, AvgDist, SigmaDist)
endfor
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dist_ellipse_contour_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fit_ellipse_contour_xld`

Alternativen

`dist_ellipse_contour_points_xld`

Modul

Foundation

dist_rectangle2_contour_points_xld (Contour : : ClippingEndPoints, Row, Column, Phi, Length1, Length2 : Distances)

Berechnen der Abstände von allen Punkten einer Kontur zu einem Rechteck.

`dist_rectangle2_contour_points_xld` bestimmt die euklidischen Abstände zwischen den Punkten einer Kontur `Contour` und einem Rechteck beliebiger Orientierung, das durch den Mittelpunkt (`Row`, `Column`), die Orientierung `Phi` und halben Kantenlängen `Length1` und `Length2` definiert ist. Der Winkel `Phi` wird im Bogenmaß spezifiziert und gibt den Winkel der horizontalen Achse und der Kante mit der halben Länge `Length1` im mathematisch positiven Sinn (gegen den Uhrzeigersinn) an.

Je nach Vorverarbeitung lassen sich die Start- und Endpunkte einer Kontur nicht exakt bestimmen. Dann wird der Operator `fit_rectangle2_contour_xld` typischerweise mit einem Wert größer null für den Parameter `ClippingEndPoints` aufgerufen, um eine bestimmte Anzahl an Punkten an beiden Enden der Kontur für die Berechnung auszuschließen. Damit der Abstand für dieselben Konturpunkte wie in der Rechtecksanpassung berechnet werden kann, ist für den Parameter `ClippingEndPoints` derselbe Wert wie bei `fit_rectangle2_contour_xld` zu wählen.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
  Eingabekontur.
- ▷ **ClippingEndPoints** (input_control) ..... integer ⇝ *integer*
  Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Abstandsberechnung ignoriert werden sollen.
  Defaultwert: 0
  Wertevorschläge: `ClippingEndPoints` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Restriktion: ClippingEndPoints >= 0
- ▷ **Row** (input_control) ..... rectangle2.center.y ⇝ *real*
  Zeilenkoordinate des Mittelpunktes des Rechtecks.
- ▷ **Column** (input_control) ..... rectangle2.center.x ⇝ *real*
  Spaltenkoordinate des Mittelpunktes des Rechtecks.
- ▷ **Phi** (input_control) ..... rectangle2.angle.rad ⇝ *real*
  Orientierung der Hauptachse des Rechtecks in Bogenmaß.
- ▷ **Length1** (input_control) ..... rectangle2.hwidth ⇝ *real*
  Erster Halbmesser (halbe Länge) des Rechtecks.
  Restriktion: Length1 > 0
- ▷ **Length2** (input_control) ..... rectangle2.hheight ⇝ *real*
  Zweiter Halbmesser (halbe Breite) des Rechtecks.
  Restriktion: Length2 >= 0

▷ **Distances** (output_control) ..... real-array ⇝ *real*
Abstände der Konturpunkte zum Rechteck.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dist_rectangle2_contour_points_xld` den Wert TRUE. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`fit_rectangle2_contour_xld`

Modul

Foundation

eccentricity_points_xld (XLD : : : Anisometry)

Anisometrie von Konturen oder Polygonen, die wie Punktwolken behandelt werden.

`eccentricity_points_xld` berechnet das Formmerkmal `Anisometry`, das aus den geometrischen Momenten abgeleitet wird. Im Gegensatz zu `eccentricity_xld` werden nur die Eigenschaften der Konturpunkte, nicht die der durch die Kontur eingeschlossenen Fläche berechnet. Die Reihenfolge der Kontur- oder Polygonpunkte wird nicht berücksichtigt. Bei geschlossenen Konturen oder Polygonen (Anfangspunkt = Endpunkt) wird der Endpunkt der Kontur oder des Polygons nicht berücksichtigt, da er sonst doppeltes Gewicht erhielte.

Ist die Kontur `XLD` nicht kreuzungsfrei oder kann sie nicht kreuzungsfrei durch eine Line vom End- zum Anfangspunkt geschlossen werden, liefert `eccentricity_points_xld` immer noch sinnvolle Ergebnisse und ist in diesem Fall `eccentricity_xld` vorzuziehen. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit `test_self_intersection_xld` überprüft werden.

Berechnung: Seien die Ellipsenradien Ra, Rb (siehe `elliptic_axis_points_xld`) gegeben, dann gilt:

$$\texttt{Anisometry} = \frac{Ra}{Rb}$$

Die Anisometrie eines Kreises ist 1.0.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse für die Anisometrie in einem Tupel in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in `XLD` abgespeichert.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen bzw. Polygone.

▷ **Anisometry** (output_control) ..... real(-array) ⇝ *real*
Anisometrie der Konturen bzw. Polygone.
Zusicherung: Anisometry >= 1.0

Ergebnis

`eccentricity_points_xld` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).

- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Alternativen

eccentricity_xld

Siehe auch

elliptic_axis_points_xld, moments_region_2nd, select_shape_xld

Modul

Foundation

eccentricity_xld (XLD : : : Anisometry, Bulkiness, StructureFactor)

Aus den Ellipsenparametern abgeleitete Formmerkmale von Konturen bzw. Polygonen.

eccentricity_xld berechnet die drei Formmerkmale Anisometry, Bulkiness und StructureFactor, die aus den geometrischen Momenten hergeleitet sind, für jede Eingabekontur bzw. jedes Eingabepolyon. Ist die Eingabekontur oder das Eingabepolygon nicht geschlossen, so wird sie/es vor der Berechnung automatisch geschlossen.

Berechnung: Seien die Ellipsenradien Ra, Rb (siehe elliptic_axis_xld) und die eingeschlossene Fläche F der Kontur bzw. des Polygons gegeben, dann gilt:

$$\texttt{Anisometry} = \frac{Ra}{Rb}$$

$$\texttt{Bulkiness} = \frac{\pi \cdot Ra \cdot Rb}{F}$$

$$\texttt{StructureFactor} = \texttt{Anisometry} \cdot \texttt{Bulkiness} - 1$$

Die Anisometrie eines Kreises ist 1.0.

Es ist zu beachten, dass eccentricity_xld nur dann sinnvolle Ergebnisse liefert, falls die Kontur oder das Polygon eine Fläche in der Ebene einschließt. Insbesondere darf sich die Kontur oder das Polygon nicht selbst schneiden. Dies ist insbesondere dann zu beachten, wenn offene Konturen und Polygone übergeben werden, da diese automatisch geschlossen werden, was dazu führen kann, dass sich die Kontur oder das Polygon selbst schneidet. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden. Falls sich die Kontur oder das Polygon selbst schneidet, kann ein sinnvoller Wert für die Anisometrie mit eccentricity_points_xld berechnet werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Anisometry** (output_control) ..... real(-array) ⇝ *real*
  Anisometrie der Konturen bzw. Polygone.
  Zusicherung: Anisometry >= 1.0
- ▷ **Bulkiness** (output_control) ..... real(-array) ⇝ *real*
  Sperrigkeit der Konturen bzw. Polygone.
- ▷ **StructureFactor** (output_control) ..... real(-array) ⇝ *real*
  Strukturfaktor der Konturen bzw. Polygone.

Ergebnis

eccentricity_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Alternativen

eccentricity_points_xld

Siehe auch

elliptic_axis_xld, moments_region_2nd, select_shape_xld, area_center_xld

Modul

Foundation

elliptic_axis_points_xld (XLD : : : Ra, Rb, Phi)

Parameter der äquivalenten Ellipse von Konturen oder Polygonen, die wie Punktwolken behandelt werden.

elliptic_axis_points_xld berechnet die Radien (Ra, Rb) und die Orientierungen (Phi, im Bogenmaß) der Ellipsen, die die gleiche Orientierung und das gleiche Seitenverhältnis wie die von den Eingabekonturen oder -polygonen definierten Punktwolken haben. (d.h. die Reihenfolge der Kontur- oder Polygonpunkte wird nicht berücksichtigt). Bei geschlossenen Konturen oder Polygonen (Anfangspunkt = Endpunkt) wird der Endpunkt der Kontur oder des Polygons nicht berücksichtigt, da er sonst doppeltes Gewicht erhielte.

Berechnung: Seien die Momente M_{20}, M_{02} und M_{11} normiert auf die Fläche gegeben (siehe moments_points_xld),dann berechnen sich die Länge der Hauptachse Ra und die Länge der Nebenachse Rb zu:

$$\mathtt{Ra} = \frac{\sqrt{8(M_{20} + M_{02} + \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}}{2}$$

$$\mathtt{Rb} = \frac{\sqrt{8(M_{20} + M_{02} - \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}}{2}$$

Die Orientierung Phi, also der Winkel zwischen der Hauptachse und der x- bzw. Spaltenachse, ist definiert durch:

$$\mathtt{Phi} = -0.5 \operatorname{atan2}(2M_{11}, M_{02} - M_{20})$$

elliptic_axis_points_xld sollte dann verwendet werden, wenn die Kontur XLD entweder nicht kreuzungsfrei ist oder nicht kreuzungsfrei durch eine Line vom End- zum Anfangspunkt geschlossen werden kann. In diesem Fall liefert elliptic_axis_xld keine sinnvollen Ergebnisse. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der gleichen Reihenfolge wie die entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Ra** (output_control) ..... real(-array) ⇝ *real*
  Hauptradius.
  Zusicherung: Ra >= 0.0
- ▷ **Rb** (output_control) ..... real(-array) ⇝ *real*
  Nebenradius.
  Zusicherung: Rb >= 0.0 && Rb <= Ra
- ▷ **Phi** (output_control) ..... angle.rad(-array) ⇝ *real*
  Winkel zwischen Hauptachse und Spalten-Achse (Bogenmaß).
  Zusicherung: - pi / 2 < Phi && Phi <= pi / 2

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

`elliptic_axis_points_xld` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`gen_contours_skeleton_xld`, `edges_sub_pix`, `threshold_sub_pix`, `gen_contour_polygon_xld`, `test_self_intersection_xld`

Nachfolger

`area_center_points_xld`, `gen_ellipse_contour_xld`

Alternativen

`elliptic_axis_xld`, `smallest_rectangle2`

Siehe auch

`moments_xld`, `smallest_circle_xld`, `smallest_rectangle1_xld`, `smallest_rectangle2_xld`, `shape_trans_xld`

Literatur

R. Haralick, L. Shapiro „Computer and Robot Vision“ Addison-Wesley, 1992, pp. 73-75

Modul

Foundation

elliptic_axis_xld (XLD : : : Ra, Rb, Phi)

Parameter der äquivalenten Ellipse von Konturen oder Polygonen.

`elliptic_axis_xld` berechnet die Radien und die Orientierungen der Ellipsen, die die gleiche Orientierung und das gleiche Seitenverhältnis wie die jeweiligen Eingabekonturen bzw. Eingabepolygone haben. Es wird die Länge der Hauptachse `Ra` und der Nebenachse `Rb` sowie die Orientierung der Hauptachse bezüglich der Horizontalen (`Phi`) bestimmt. Der Winkel wird dabei im Bogenmaß angegeben. Es wird vorausgesetzt, dass die Konturen bzw. die Polygone geschlossen sind. Ist das nicht der Fall, werden sie künstlich geschlossen.

Berechnung: Seien die Momente M_{20}, M_{02} und M_{11} normiert auf die Fläche gegeben (siehe `moments_xld`), dann berechnen sich die Radien `Ra` und `Rb` zu:

$$\texttt{Ra} = \frac{\sqrt{8(M_{20} + M_{02} + \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}}{2}$$

$$\texttt{Rb} = \frac{\sqrt{8(M_{20} + M_{02} - \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}}{2}$$

Die Orientierung `Phi` ist definiert durch:

$$\texttt{Phi} = -0.5\text{atan2}(2M_{11}, M_{02} - M_{20})$$

Es ist zu beachten, dass `elliptic_axis_xld` nur dann sinnvolle Ergebnisse liefert, falls die Kontur oder das Polygon eine Fläche in der Ebene einschließt. Insbesondere darf sich die Kontur oder das Polygon nicht selbst schneiden. Dies ist insbesondere dann zu beachten, wenn offene Konturen und Polygone übergeben werden, da diese automatisch geschlossen werden, was dazu führen kann, dass sich die Kontur oder das Polygon selbst schneidet. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit `test_self_intersection_xld`

überprüft werden. Falls sich die Kontur oder das Polygon selbst schneidet, können sinnvolle Werte für die Ellipsenparameter mit `elliptic_axis_points_xld` berechnet werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in `XLD` abgespeichert.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen oder Polygone.

▷ **Ra** (output_control) ..... real(-array) ⇝ *real*
Hauptradius.
Zusicherung: Ra >= 0.0

▷ **Rb** (output_control) ..... real(-array) ⇝ *real*
Nebenradius.
Zusicherung: Rb >= 0.0 && Rb <= Ra

▷ **Phi** (output_control) ..... angle.rad(-array) ⇝ *real*
Winkel zwischen Hauptachse und x-Achse (Bogenmaß).
Zusicherung: - pi / 2 < Phi && Phi <= pi / 2

Komplexität

Sei N die Anzahl der Punkte in der Kontur bzw. im Polygon, dann beträgt die Laufzeitkomplexität $O(N)$.

Ergebnis

`elliptic_axis_xld` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`gen_contours_skeleton_xld`, `edges_sub_pix`, `threshold_sub_pix`, `gen_contour_polygon_xld`, `test_self_intersection_xld`

Nachfolger

`area_center_xld`, `gen_ellipse_contour_xld`

Alternativen

`elliptic_axis_points_xld`, `smallest_rectangle2`

Siehe auch

`moments_xld`, `smallest_circle_xld`, `smallest_rectangle1_xld`, `smallest_rectangle2_xld`, `shape_trans_xld`

Literatur

R. Haralick, L. Shapiro „Computer and Robot Vision“ Addison-Wesley, 1992, pp. 73-75

Modul

Foundation

```
fit_circle_contour_xld ( Contours : : Algorithm, MaxNumPoints,
    MaxClosureDist, ClippingEndPoints, Iterations,
    ClippingFactor : Row, Column, Radius, StartPhi, EndPhi,
    PointOrder )
```

Approximieren von bogenförmigen XLD-Konturen durch Kreise.

`fit_circle_contour_xld` approximiert die XLD-Konturen `Contours` durch Kreise. Eine Segmentierung der Eingabekonturen erfolgt dabei nicht. Es muss also sichergestellt sein, dass jede Kontur in `Contours` genau einem Kreis entspricht. Zurückgegeben wird der Mittelpunkt (`Row`, `Column`) und der Radius `Radius`.

Das gewünschte Approximationsverfahren wird über den Parameter `Algorithm` ausgewählt:

'algebraic' Minimiert wird der algebraische Abstand zwischen den Konturpunkten und dem gesuchten Kreis.

'ahuber' Wie *'algebraic'*. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden (siehe unten).

'atukey' Wie *'algebraic'*. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden (siehe unten).

'geometric' Minimiert wird der geometrische Anstand von Konturpunkten und dem gesuchten Kreis. Dieses Fehlermaß ist in einem statistischen Sinne optimal, braucht aber auch eine längere Rechenzeit. Besonders bei merklich verrauschten Konturpunkten ist die Wahl dieser Option zu empfehlen.

'geohuber' Wie *'geometric'*. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden (siehe unten).

'geotukey' Wie *'geometric'*. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden (siehe unten).

In den Modi '*huber' und '*tukey' wird eine robuste Fehlerstatistik verwendet, um die Standardabweichung der Abstände der Konturpunkte (ohne Ausreißer) von dem approximierenden Kreis zu ermitteln. Der Parameter `ClippingFactor` (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto mehr Punkte werden als Ausreißer behandelt. Jeder Konturpunkt fließt individuell gewichtet in die Berechnung der Kreisparameter mit ein. Ausreißer werden nach dem Ansatz von Tukey komplett eliminiert, während bei dem Verfahren nach Huber diese noch gering, i.e., linear, gewichtet werden. Ohne robuste Gewichtung geht der Abstand quadratisch in die Optimierung mit ein. In der Praxis ist der Ansatz nach Tukey zu empfehlen.

Der Parameter `Iterations` enthält die Anzahl durchzuführender Iterationen für die Algorithmen *'algebraic'*, *'ahuber'* und *'atukey'*. Für die Algorithmen *'geometric'*, *'geohuber'* und *'geotukey'* wird dieser Parameter ignoriert. Wird `Iterations` auf null gesetzt, so versucht der Algorithmus nicht, den gefitteteten Kreis iterativ zu verbessern. Er überprüft lediglich, ob der erste Fit bereits ausreichend approximiert, abhängig von der gewählten Behandlung der Ausreißer.

Zur Reduktion des Aufwands lässt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für `MaxNumPoints` eine Zahl ungleich *-1* übergeben, werden nur maximal `MaxNumPoints` gleichmäßig über die Kontur verteilte Punkte verwendet.

Für Kreisbögen, also offene Konturen, werden die Anfangs- und Endpunkte bestimmt, indem zu den Anfangs- und Endpunkten der Eingabekonturen die nächstgelegenen Kreispunkte ermittelt werden. Zurückgegeben werden dann in `StartPhi` und `EndPhi` die korrespondierenden Winkel dieser Punkte bezogen auf die Horizontale, vgl. `gen_ellipse_contour_xld`. Konturen, deren Anfangs- und Endpunkte weniger als `MaxClosureDist` voneinander entfernt sind, werden als geschlossen betrachtet und entsprechend durch einen Kreis (anstelle eines Kreisbogens) approximiert. Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung nicht exakt bekannt sein können, besteht die Möglichkeit, `ClippingEndPoints` Punkte am Anfang und Ende der Kontur von der Kreisanpassung auszuschließen. Sie werden jedoch weiterhin für die Bestimmung der Anfangs- und Endwinkel verwendet.

Die notwendige, minimale Anzahl an Konturpunkten zur Anpassung eines Kreises ist drei. Es wird daher vorausgesetzt, dass die Konturen mindestens $3 + 2 \cdot$ `ClippingEndPoints` Punkte enthalten.

Parameter

▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *object*
Eingabekonturen.

▷ **`Algorithm`** (input_control) ..... string ⇝ *string*
Algorithmus zur Kreisanpassung.
Defaultwert: 'algebraic'
Werteliste: `Algorithm` ∈ {'algebraic', 'ahuber', 'atukey', 'geometric', 'geohuber', 'geotukey'}

▷ **`MaxNumPoints`** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl Konturpunkte zur Kreisanpassung (-1 für alle Punkte).
Defaultwert: -1
Restriktion: MaxNumPoints >= 3

▷ **`MaxClosureDist`** (input_control) ..... real ⇝ *real*
Maximaler Abstand zweier Konturendpunkte, so dass die Kontur noch als geschlossene Kontur akzeptiert wird.
Defaultwert: 0.0
Restriktion: MaxClosureDist >= 0.0

▷ **ClippingEndPoints** (input_control) ..... integer ⇝ *integer*
Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Kreisanpassung ignoriert werden sollen.
Defaultwert: 0
Restriktion: ClippingEndPoints >= 0

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl von Iterationen bei der gewichteten Optimierung.
Defaultwert: 3
Restriktion: Iterations >= 0

▷ **ClippingFactor** (input_control) ..... real ⇝ *real*
Clipping Faktor für die Ausreißerdämpfung (typisch: 1.0 bei Huber und 2.0 bei Tukey).
Defaultwert: 2.0
Wertevorschläge: `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
Restriktion: ClippingFactor > 0

▷ **Row** (output_control) ..... circle.center.y(-array) ⇝ *real*
Zeilenkoordinate des Mittelpunktes des Kreises.

▷ **Column** (output_control) ..... circle.center.x(-array) ⇝ *real*
Spaltenkoordinate des Mittelpunktes des Kreises.

▷ **Radius** (output_control) ..... circle.radius(-array) ⇝ *real*
Radius des Kreises.

▷ **StartPhi** (output_control) ..... angle.rad(-array) ⇝ *real*
Winkel des Startpunktes [rad].

▷ **EndPhi** (output_control) ..... angle.rad(-array) ⇝ *real*
Winkel des Endpunktes [rad].

▷ **PointOrder** (output_control) ..... string(-array) ⇝ *string*
Mathematischer Umlaufsinn.
Werteliste: `PointOrder` ∈ {'positive', 'negative'}

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Kreisanpassung durchgeführt werden, liefert `fit_circle_contour_xld` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter `ClippingFactor` so klein gewählt worden ist, dass alle Punkte als Ausreißer klassifiziert werden, wird die Fehlermeldung 3264 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `smooth_contours_xld`

Nachfolger

`gen_ellipse_contour_xld`, `disp_circle`, `get_points_ellipse`

Siehe auch

`fit_ellipse_contour_xld`, `fit_line_contour_xld`, `fit_rectangle2_contour_xld`

Modul

Foundation

fit_ellipse_contour_xld (Contours : : Algorithm, MaxNumPoints, MaxClosureDist, ClippingEndPoints, VossTabSize, Iterations, ClippingFactor : Row, Column, Phi, Radius1, Radius2, StartPhi, EndPhi, PointOrder)

Approximieren von bogenförmigen XLD-Konturen durch Ellipsen.

fit_ellipse_contour_xld approximiert die XLD-Konturen Contours durch Ellipsenbögen bzw. geschlossene Ellipsen. Eine Segmentierung der Eingabekonturen erfolgt dabei nicht. Es muss also sichergestellt sein, dass jede Kontur in Contours genau einem Ellipsenbogen entspricht. Zurückgegeben wird der Mittelpunkt (Row, Column), die Orientierung der Hauptachse Phi und die Länge der großen Radius1 bzw. der kleinen Halbachse Radius2 jeder Ellipse. Ellipsenbögen werden zusätzlich durch den Anfangs- und Endwinkel StartPhi und EndPhi des Start- bzw. Endpunktes, sowie den zugehörigen Umlaufsinn PointOrder charakterisiert. Bei geschlossenen Ellipsen werden die entsprechenden Ausgabeparameter mit 0, 2π und 'positive' besetzt.

Das gewünschte Approximationsverfahren wird über den Parameter Algorithm ausgewählt:

'fitzgibbon' Minimiert wird der algebraische Abstand
$ax_i^2 + bx_iy_i + cy_i^2 + dx_i + ey_i + f$ zwischen den Konturpunkten (x_i, y_i) und der gesuchten Ellipse. Durch die Nebenbedingung $4ac - b^2 = 1$ wird dabei sichergestellt, dass eine Ellipse (und nicht etwa eine Hyperbel oder Parabel) angepasst wird.

'fhuber' Wie *'fitzgibbon'*. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden (siehe unten).

'ftukey' Wie *'fitzgibbon'*. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden (siehe unten).

'geometric' Minimiert wird der geometrische Anstand von Konturpunkten und der gesuchten Ellipse. Dieses Fehlermaß ist in einem statistischen Sinne optimal, braucht aber auch eine längere Rechenzeit, da die Berechnung iterative erfolgt. Besonders bei merklich verrauschten Konturpunkten ist die Wahl dieser Option zu empfehlen.

'geohuber' Wie *'geometric'*. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden (siehe unten).

'geotukey' Wie *'geometric'*. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden (siehe unten).

'voss' Die Eingabekontur wird in eine affine Standardlage transformiert. Dann wird vermöge der Momente der transformierten Kontur (bzw. der davon eingeschlossenen Bildregion) dasjenige Einheitskreissegment bestimmt, dessen Standardlage am besten mit der der transformierten Kontur übereinstimmt. Aus der zugehörigen Ellipse in Standardlage und der affinen Transformation, die die Kontur in Standardlage überführt hat, wird dann die entsprechende Ellipse ermittelt. Es werden VossTabSize Einheitskreissegment unterschieden (und deren Momente etc. beim ersten Aufruf des Operators tabelliert).

'focpoints' Für jeden Punkt P auf einer Ellipse gilt, dass die Summe seiner Abstände zu den Brennpunkten F_1,F_2 der Ellipse gleich dem doppelten Hauptradius a ist. Bei diesem Ansatz wird die Abweichung PF_1+PF_2-2a für alle Konturpunkte durch eine *least squares* Ausgleichsrechnung minimiert.

'fphuber' Wie *'focpoints'*. Es wird jedoch eine *gewichtete* Least-Squares-Ausgleichsrechnung durchgeführt, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden (siehe unten).

'fptukey' Wie *'focpoints'*. Es wird jedoch eine *gewichtete* Least-Squares-Ausgleichsrechnung durchgeführt, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden (siehe unten).

In den Modi '*huber' und '*tukey' wird eine robuste Fehlerstatistik verwendet, um die Standardabweichung der Abstände der Konturpunkte (ohne Ausreißer) von der approximierenden Ellipse zu ermitteln. Der Parameter ClippingFactor (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto mehr Punkte werden als Ausreißer behandelt. Jeder Konturpunkt fließt individuell gewichtet in die Berechnung der Ellipsenparameter mit ein. Dieser Prozess kann iteriert werden, und der Parameter Iterations enthält die Anzahl durchzuführender Iterationen. Iterations ist relevant für alle Verfahren bis auf die beiden auf dem geometrischen Abstand basierenden Verfahren *'geohuber'* und *'geotukey'*, bei denen dieser Parameter ignoriert wird. Falls der Algorithmus innerhalb dieser Anzahl von Iterationen keine passende Ellipse finden kann, so wird eine Linie gefittet; für Radius2 wird null zurückgegeben. Ausreißer werden nach dem Ansatz von Tukey komplett eliminiert, während bei dem Verfahren nach Huber diese noch gering, i.e., linear, gewichtet werden. Ohne robuste Gewichtung geht der Abstand quadratisch in die Optimierung mit ein. In der Praxis ist der Ansatz nach Tukey zu empfehlen.

Zur Reduktion des Aufwandes lässt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für MaxNumPoints eine Zahl ungleich *-1* übergeben, werden nur maximal MaxNumPoints gleichmäßig über die Kontur verteilte Punkte verwendet.

Für Ellipsenbögen, also offene Konturen, werden die Anfangs- und Endpunkte bestimmt, indem zu den Anfangs- und Endpunkten der Eingabekonturen die nächstgelegenen Ellipsenendpunkte ermittelt werden. Zurückgegeben

werden dann in StartPhi und EndPhi die korrespondierenden Winkel dieser Punkte bezogen auf die Hauptachse der Ellipse, vgl. gen_ellipse_contour_xld. Konturen, deren Anfangs- und Endpunkte weniger als MaxClosureDist voneinander entfernt sind, werden als geschlossen betrachtet und entsprechend durch eine Ellipse approximiert. Für geschlossene Konturen werden die Startwerte der *'focpoints'* Algorithmen auf schnellere und weniger genaue Weise geschätzt. Daher können für manche geschlossene Konturen mehr Iterations notwendig sein. Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung nicht exakt bekannt sein können, besteht die Möglichkeit, ClippingEndPoints Punkte am Anfang und Ende der Kontur von der Ellipsenanpassung auszuschließen. Sie werden jedoch weiterhin für die Bestimmung der Anfangs- und Endwinkel verwendet.

Die notwendige, minimale Anzahl an Konturpunkten zur Anpassung einer Ellipse ist fünf. Es wird daher vorausgesetzt, dass die Konturen mindestens $5 + 2 \cdot$ ClippingEndPoints Punkte enthalten.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabekonturen.
- ▷ **Algorithm** (input_control) ..... string ⇝ *string*
  Algorithmus zur Ellipsenanpassung.
  Defaultwert: 'fitzgibbon'
  Werteliste: Algorithm ∈ {'fitzgibbon', 'fhuber', 'ftukey', 'geometric', 'geohuber', 'geotukey', 'voss', 'focpoints', 'fphuber', 'fptukey'}
- ▷ **MaxNumPoints** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl Konturpunkte zur Ellipsenanpassung (-1 für alle Punkte).
  Defaultwert: -1
  Restriktion: MaxNumPoints >= 5
- ▷ **MaxClosureDist** (input_control) ..... real ⇝ *real*
  Maximaler Abstand zweier Konturendpunkte, so dass die Kontur noch als geschlossene Kontur akzeptiert wird.
  Defaultwert: 0.0
  Restriktion: MaxClosureDist >= 0.0
- ▷ **ClippingEndPoints** (input_control) ..... integer ⇝ *integer*
  Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Ellipsenanpassung ignoriert werden sollen.
  Defaultwert: 0
  Restriktion: ClippingEndPoints >= 0
- ▷ **VossTabSize** (input_control) ..... integer ⇝ *integer*
  Anzahl Einheitskreissegmente für das Verfahren nach Voss.
  Defaultwert: 200
  Restriktion: VossTabSize >= 25 && VossTabSize <= 5000
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von Iterationen bei der gewichteten Optimierung.
  Defaultwert: 3
  Restriktion: Iterations >= 0
- ▷ **ClippingFactor** (input_control) ..... real ⇝ *real*
  Clipping Faktor für die Ausreißerdämpfung (typisch: 1.0 bei '*huber' und 2.0 bei '*tukey').
  Defaultwert: 2.0
  Wertevorschläge: ClippingFactor ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
  Restriktion: ClippingFactor > 0
- ▷ **Row** (output_control) ..... ellipse.center.y(-array) ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Column** (output_control) ..... ellipse.center.x(-array) ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Phi** (output_control) ..... ellipse.angle.rad(-array) ⇝ *real*
  Orientierung der Hauptachse in Bogenmaß.
- ▷ **Radius1** (output_control) ..... ellipse.radius1(-array) ⇝ *real*
  Länge der großen Halbachse.
- ▷ **Radius2** (output_control) ..... ellipse.radius2(-array) ⇝ *real*
  Länge der kleinen Halbachse.
- ▷ **StartPhi** (output_control) ..... angle.rad(-array) ⇝ *real*
  Winkel des Startpunktes [rad].

▷ **EndPhi** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Winkel des Endpunktes [rad].

▷ **PointOrder** (output_control) . . . . . string(-array) ⇝ *string*
mathematischer Umlaufsinn.
Werteliste: PointOrder ∈ {'positive', 'negative'}

Beispiel

```
read_image (Image, 'caltab')
find_caltab (Image, CalPlate, 'caltab_800mm.descr', 3, 112, 5)
reduce_domain (Image, CalPlate, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 20, 40)
select_contours_xld (Edges, EdgesClosed, 'closed', 0, 2.0, 0, 0)
select_contours_xld (EdgesClosed, EdgesMarks, 'length', 20, 80, 0, 0)
fit_ellipse_contour_xld (EdgesMarks, 'fitzgibbon', -1, 2, 0, 200, 3, 2.0, \
                         Row, Column, Phi, Radius1, Radius2, StartPhi, \
                         EndPhi, PointOrder)
gen_ellipse_contour_xld (EllMarks, Row, Column, Phi, Radius1, Radius2, \
                         StartPhi, EndPhi, PointOrder, 1.5)
length_xld(EllMarks,Length)
```

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Ellipsenanpassung durchgeführt werden, liefert fit_ellipse_contour_xld den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter ClippingFactor so klein gewählt worden ist, dass alle Punkte als Ausreißer klassifiziert werden, wird die Fehlermeldung 3264 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix, smooth_contours_xld

Nachfolger

gen_ellipse_contour_xld, disp_ellipse, get_points_ellipse

Siehe auch

fit_line_contour_xld, fit_circle_contour_xld, fit_rectangle2_contour_xld

Modul

Foundation

fit_line_contour_xld (Contours : : Algorithm, MaxNumPoints, ClippingEndPoints, Iterations, ClippingFactor : RowBegin, ColBegin, RowEnd, ColEnd, Nr, Nc, Dist)

Approximieren von XLD-Konturen durch Liniensegmente.

fit_line_contour_xld approximiert die XLD-Konturen Contours durch Liniensegmente. Eine Segmentierung der Eingabekonturen erfolgt dabei nicht. Es muss also sichergestellt sein, dass jede Kontur in Contours genau einem Liniensegment entspricht. Zurückgegeben wird der Anfangs- (RowBegin, ColBegin) bzw. Endpunkt (RowEnd, ColEnd), sowie die Geradengleichung jedes Liniensegments, codiert durch den Normalenvektor (Nr, Nc) der Gerade und dem Abstand Dist der Gerade vom Ursprung, d.h. die Geradengleichung ist gegeben durch $r \cdot \mathtt{Nr} + c \cdot \mathtt{Nc} - \mathtt{Dist} = 0$.

Das gewünschte Approximationsverfahren wird über den Parameter Algorithm ausgewählt:

'regression': Standard-Least-Squares-Geradenanpassung.

'huber': Gewichtete *Least-Squares*-Geradenanpassung, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden (siehe unten).

'tukey': Gewichtete *Least-Squares*-Geradenanpassung, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden (siehe unten).

'drop': Gewichtete *Least-Squares*-Geradenanpassung, bei der Ausreißer ignoriert werden, d.h. alle Konturpunkte, die weiter von der Kontur entfernt liegen als die mittlere Entfernung von der Regressionsgeraden multipliziert mit dem `ClippingFactor` (s.u.).

'gauss': Gewichtete *Least-Squares*-Geradenanpassung, bei der Ausreißer gestützt auf Mittelwert und Standardabweichung der Abweichungen aller Konturpunkte von der Geraden gedämpft werden.

In den Modi *'huber'*, *'tukey'* und *'drop'* wird eine robuste Fehlerstatistik verwendet, um die Standardabweichung der Abstände der Konturpunkte (ohne Ausreißer) von der approximierenden Geraden zu ermitteln. Der Parameter `ClippingFactor` (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto mehr Punkte werden als Ausreißer behandelt. Die Ausreißerdetektion wird iteriert. Der Parameter `Iterations` enthält die Anzahl durchzuführender Iterationen. Er wird im Modus *'regression'* ignoriert. Bei dem Ansatz nach Tukey (*'tukey'*) werden Ausreißer noch vor der Kreisanpassung eliminiert (und die übrigen Punkte werden gewichtet), während die Ausreißer bei dem Ansatz nach Huber (*'huber'*) noch in geringem Maße in die Optimierung eingehen. Speziell gehen bei Letzterem die Ausreißer linear und die übrigen Punkte quadratisch in die Optimierung ein. In der Praxis ist der Ansatz nach Tukey zu empfehlen.

Zur Reduktion des Aufwandes lässt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für `MaxNumPoints` eine Zahl ungleich -1 übergeben, werden nur maximal `MaxNumPoints` gleichmäßig über die Kontur verteilte Punkte verwendet.

Die Anfangs- und Endpunkte werden bestimmt, indem zu den Anfangs- und Endpunkten der Eingabekonturen die nächstgelegenen Punkte auf den zugrundeliegenden Regressionsgeraden ermittelt werden. Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung nicht exakt bekannt sein können, besteht die Möglichkeit, `ClippingEndPoints` Punkte am Anfang und Ende der Kontur von der Geradenanpassung auszuschließen. Sie werden jedoch weiterhin für die Bestimmung der Anfangs- und Endpunkte verwendet.

Die notwendige, minimale Anzahl an Konturpunkten zur Anpassung einer Linie ist zwei. Es wird daher vorausgesetzt, dass die Konturen mindestens $2 + 2 \cdot$ `ClippingEndPoints` Punkte enthalten.

Parameter

▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *object*
Eingabekonturen.

▷ **`Algorithm`** (input_control) ..... string ⇝ *string*
Algorithmus zur Geradenanpassung.
Defaultwert: 'tukey'
Werteliste: `Algorithm` ∈ {'regression', 'huber', 'tukey', 'gauss', 'drop'}

▷ **`MaxNumPoints`** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl Konturpunkte zur Geradenanpassung (-1 für alle Punkte).
Defaultwert: -1
Restriktion: MaxNumPoints >= 2

▷ **`ClippingEndPoints`** (input_control) ..... integer ⇝ *integer*
Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Geradenanpassung ignoriert werden sollen.
Defaultwert: 0
Restriktion: ClippingEndPoints >= 0

▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl von Iterationen (unbenutzt bei *'regression'*).
Defaultwert: 5
Restriktion: Iterations >= 0

▷ **`ClippingFactor`** (input_control) ..... real ⇝ *real*
Clipping Faktor für die Ausreißerdämpfung (typische Werte: *1.0* bei *'huber'* und *'drop'* sowie *2.0* bei *'tukey'*).
Defaultwert: 2.0
Wertevorschläge: `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
Restriktion: ClippingFactor > 0

▷ **RowBegin** (output_control) ..... line.begin.y(-array) ⇝ *real*
Zeilenkoordinate der Anfangspunkte der Liniensegmente.

▷ **ColBegin** (output_control) ..... line.begin.x(-array) ⇝ *real*
Spaltenkoordinate der Anfangspunkte der Liniensegmente.

▷ **RowEnd** (output_control) ..... line.end.y(-array) ⇝ *real*
Zeilenkoordinate der Endpunkte der Liniensegmente.

▷ **ColEnd** (output_control) ..... line.end.x(-array) ⇝ *real*
Spaltenkoordinate der Endpunkte der Liniensegmente.

▷ **Nr** (output_control) ..... number(-array) ⇝ *real*
Geradenparameter: Zeilenkoordinate von Normalenvektor

▷ **Nc** (output_control) ..... number(-array) ⇝ *real*
Geradenparameter: Spaltenkoordinate von Normalenvektor

▷ **Dist** (output_control) ..... number(-array) ⇝ *real*
Geradenparameter: Abstand der Geraden vom Ursprung

Beispiel

```
read_image (Image, 'mreut')
edges_sub_pix (Image, Edges, 'lanser2', 0.5, 20, 40)
gen_polygons_xld (Edges, Polygons, 'ramer', 2)
split_contours_xld (Polygons, Contours, 'polygon', 1, 5)
fit_line_contour_xld (Contours, 'regression', -1, 0, 5, 2, RowBegin, \
                      ColBegin, RowEnd, ColEnd, Nr, Nc, Dist)
```

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Geradenanpassung durchgeführt werden, liefert `fit_line_contour_xld` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter `ClippingFactor` so klein gewählt worden ist, dass alle Punkte als Ausreißer klassifiziert werden, wird die Fehlermeldung 3264 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `smooth_contours_xld`

Nachfolger

`disp_line`, `line_orientation`

Siehe auch

`regress_contours_xld`, `get_regress_params_xld`

Modul

Foundation

fit_rectangle2_contour_xld (Contours : : Algorithm, MaxNumPoints, MaxClosureDist, ClippingEndPoints, Iterations, ClippingFactor : Row, Column, Phi, Length1, Length2, PointOrder)

Anpassen von Rechtecken an XLD-Konturen.

`fit_rectangle2_contour_xld` passt Rechtecke an die rechteckigen XLD-Konturen `Contours` an und gibt die Parameter der Rechtecke in `Row`, `Column` (Mittelpunkt), `Phi` (Orientierung) sowie `Length1` und `Length2` (halben Kantenlängen) zurück. Der Winkel `Phi` wird im Bogenmaß ausgegeben und gibt den Winkel

der horizontalen Achse und der Kante mit der halben Länge `Length1` im mathematisch positiven Sinn (gegen den Uhrzeigersinn) an. Zusätzlich wird der Umlaufsinn der Kontur in `PointOrder` zurückgegeben. `PointOrder` = *'positive'* bedeutet hierbei, dass die Kontur im mathematisch positiven Sinn (gegen den Uhrzeigersinn) durchlaufen wird.

Das gewünschte Approximationsverfahren wird über den Parameter `Algorithm` ausgewählt:

'regression' Standard *Least-Squares*-Geradenanpassung.

'huber' Gewichtete *Least-Squares*-Geradenanpassung, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden (siehe unten).

'tukey' Gewichtete *Least-Squares*-Geradenanpassung, bei der Ausreißer nach dem Ansatz von Tukey ignoriert werden (siehe unten).

In den Modi *'huber'* und *'tukey'* wird eine robuste Fehlerstatistik verwendet, um die Standardabweichung der Abstände der Konturpunkte (ohne Ausreißer) von den approximierenden Geraden des Rechtecks zu ermitteln. Die Standardabweichung wird pro Seite des Rechtecks separat berechnet, um auch Konturen, deren Seiten nicht genau senkrecht aufeinander stehen, verarbeiten zu können. Der Parameter `ClippingFactor` (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto mehr Punkte werden als Ausreißer behandelt. Die Ausreißerdetektion wird iteriert. Der Parameter `Iterations` enthält die Anzahl durchzuführender Iterationen. Die letzten beiden Parameter werden für `Algorithm` = *'regression'* ignoriert. Bei dem Ansatz nach Tukey ('*tukey') werden Ausreißer noch vor der Anpassung eliminiert (und die übrigen Punkte werden gewichtet), während die Ausreißer bei dem Ansatz nach Huber ('*huber') noch in geringem Maße in die Optimierung eingehen. Speziell gehen bei Letzterem die Ausreißer linear und die übrigen Punkte quadratisch in die Optimierung ein. Bei dem algebraischen Abstand gehen alle Punkte quadratisch in die Optimierung ein. In der Praxis ist der Ansatz nach Tukey zu empfehlen.

Zur Reduktion des Aufwandes lässt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für `MaxNumPoints` eine Zahl ungleich *-1* übergeben, werden nur maximal `MaxNumPoints` gleichmäßig über die Kontur verteilte Punkte verwendet.

Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung möglicherweise fehlerbehaftet sein können, besteht die Möglichkeit, `ClippingEndPoints` Punkte am Anfang und Ende der Kontur von der Rechteckanpassung auszuschließen.

Konturen, deren Anfangs- und Endpunkte weniger als `MaxClosureDist` voneinander entfernt sind, werden als geschlossen betrachtet. Bei geschlossenen Konturen wird der Endpunkt der Kontur nicht zur Anpassung herangezogen, da er sonst doppelt gewichtet würde.

Die Anpassung des Rechtecks an die Kontur basiert darauf, dass die Konturpunkte intern den vier Seiten des Rechtecks zugeordnet werden. Dabei muss es für jede Seite des Rechtecks mindestens einen Punkt geben, der im Inneren, d.h. nicht an den Enden, des Liniensegments der Seite liegt. Deshalb sind also mindestens acht Punkte zur Anpassung des Rechtecks notwendig. Ein Punkt der Kontur wird der Seite des Rechtecks zugeordnet, zu der er den kürzesten Abstand hat. Dabei werden die aktuell intern optimierten Rechtecksparameter verwendet. Falls bei der Zuordnung von Punkten zu Seiten des Rechtecks mindestens eine Seite ohne zugeordneten Punkt gefunden wird, kann das Rechteck nicht eindeutig bestimmt werden. In diesem Fall wird der Fehler 3266 zurückgeliefert. Aus diesem Grund muss bei der Verwendung von `fit_rectangle2_contour_xld` sichergestellt werden, dass die Eingabekonturen einem Rechteck hinreichend ähnlich sind. Insbesondere sollte keiner der Winkel der Kontur, falls sie durch vier Geradenstücke approximiert würde, weniger als 45 Grad oder mehr als 135 Grad betragen. Aufgrund der Zuordnung der Konturpunkte zur nächstgelegenen Rechteckseite würde dies dazu führen, dass mindestens einer Seite keine Konturpunkte zugeordnet würden. Weiterhin ist zu beachten, dass `ClippingFactor` nicht zu klein gewählt wird, um zu vermeiden, dass durch die Ausreißerunterdrückung Rechtecksseiten ohne Punkte entstehen. Dies kann nur für `Algorithm` = *'tukey'* passieren. Falls die obigen Bedingungen beachtet werden, liefert `fit_rectangle2_contour_xld` hochgenaue Rechtecksparameter zurück. In Verbindung mit der Ausreißerunterdrückung nach Tukey kann `fit_rectangle2_contour_xld` z.B. sehr gut dazu verwendet werden, Rechtecke robust an rechteckige Konturen mit abgerundeten Ecken anzupassen.

Parameter

▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *object*
Eingabekonturen.

▷ **`Algorithm`** (input_control) ..... string ⇝ *string*
Algorithmus zur Rechtecksanpassung.
Defaultwert: 'regression'
Werteliste: `Algorithm` ∈ {'regression', 'huber', 'tukey'}

▷ **MaxNumPoints** (input_control) . . . . . integer ⇝ *integer*
Maximale Anzahl Konturpunkte zur Rechtecksanpassung (-1 für alle Punkte).
Defaultwert: -1
Restriktion: MaxNumPoints == -1 || MaxNumPoints >= 8

▷ **MaxClosureDist** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand zweier Konturendpunkte, so dass die Kontur noch als geschlossene Kontur akzeptiert wird.
Defaultwert: 0.0
Restriktion: MaxClosureDist >= 0.0

▷ **ClippingEndPoints** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Rechtecksanpassung ignoriert werden sollen.
Defaultwert: 0
Wertevorschläge: `ClippingEndPoints` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Restriktion: ClippingEndPoints >= 0

▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
Maximale Anzahl von Iterationen (unbenutzt bei *'regression'*).
Defaultwert: 3
Restriktion: Iterations >= 0

▷ **ClippingFactor** (input_control) . . . . . real ⇝ *real*
Clipping Faktor für die Ausreißerdämpfung (typische Werte: *1.0* bei *'huber'* und *2.0* bei *'tukey'*).
Defaultwert: 2.0
Wertevorschläge: `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
Restriktion: ClippingFactor > 0

▷ **Row** (output_control) . . . . . rectangle2.center.y(-array) ⇝ *real*
Zeilenkoordinate des Mittelpunktes des Rechtecks.

▷ **Column** (output_control) . . . . . rectangle2.center.x(-array) ⇝ *real*
Spaltenkoordinate des Mittelpunktes des Rechtecks.

▷ **Phi** (output_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real*
Orientierung der Hauptachse des Rechtecks in Bogenmaß.

▷ **Length1** (output_control) . . . . . rectangle2.hwidth(-array) ⇝ *real*
Erster Halbmesser (halbe Länge) des Rechtecks.

▷ **Length2** (output_control) . . . . . rectangle2.hheight(-array) ⇝ *real*
Zweiter Halbmesser (halbe Breite) des Rechtecks.

▷ **PointOrder** (output_control) . . . . . string(-array) ⇝ *string*
Mathematischer Umlaufsinn der Kontur.
Werteliste: `PointOrder` ∈ {'positive', 'negative'}

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Rechteckanpassung durchgeführt werden, liefert `fit_rectangle2_contour_xld` den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt. Wenn der Parameter `ClippingFactor` so klein gewählt worden ist, dass alle Punkte als Ausreißer klassifiziert werden, wird der Fehler 3266 zurückgeliefert. Falls für mindestens eine Seite des Rechtecks keine Punkte, die auf dieser Seite liegen, gefunden werden konnten, wird auch der Fehler 3266 zurückgeliefert.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `smooth_contours_xld`, `union_collinear_contours_xld`, `union_collinear_contours_ext_xld`, `union_adjacent_contours_xld`

Nachfolger

dist_rectangle2_contour_points_xld, gen_rectangle2_contour_xld

Alternativen

smallest_rectangle2_xld

Siehe auch

fit_line_contour_xld, fit_circle_contour_xld, fit_ellipse_contour_xld

Modul

Foundation

get_contour_angle_xld (Contour : : AngleMode, CalcMode, Lookaround : Angles)

Punktweises Berechnen der Tangentenrichtung einer Kontur.

Der Operator get_contour_angle_xld berechnet für jeden Punkt der Kontur Contour die Richtung der Tangente. Dabei kann zwischen zwei Winkelausgaben AngleMode gewählt werden: Mit *'abs'* werden Winkel bezüglich der Horizontalen zwischen 0 und 2π (entgegen dem Uhrzeigersinn) ausgegeben, mit *'rel'* wird die Richtungsänderung gegenüber dem vorhergehenden Konturpunkt ausgegeben. Die Werte liegen in diesem Fall zwischen $-\pi$ und π, wobei negative Werte einer rechtsgekrümmten und positive Werte einer linksgekrümmten Kontur entsprechen.

Drei mögliche Berechnungsarten CalcMode berechnen die Tangentenrichtung im Konturpunkt i mit Hilfe der Konturpunkte im Bereich $i-$Lookaround bis $i+$Lookaround. Mit *'range'* wird der Winkel der Geraden durch die beiden Randpunkte dieses Bereichs ermittelt. Bei der Berechnungsart *'mean'* werden hingegen die Winkel aller Einzelschritte zwischen den Konturpunkten des Bereichs arithmetisch gemittelt. Mit *'regress'* wird die Steigung der Regressionsgeraden zwischen allen (gleichberechtigten) Punkten dieses Bereichs ermittelt. Lookaround ist ein Maß für die Glättung der Kontur. Die Winkel werden im Bogenmaß in dem Tupel Angles ausgegeben.

Aufgrund des durch Lookaround definierten Intervalls muss die Eingabe-Kontur eine Mindestanzahl an Punkten besitzen. Diese beträgt $2*$Lookaround$+2$, damit der Operator für die unterschiedlichen Parametereinstellungen stets sinnvolle Ausgaben liefert.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
  Eingabe-Kontur.
- ▷ **AngleMode** (input_control) ..... string ⇝ *string*
  Ausgabeart der Winkel.
  Defaultwert: 'abs'
  Werteliste: AngleMode ∈ {'abs', 'rel'}
- ▷ **CalcMode** (input_control) ..... string ⇝ *string*
  Berechnungsart der Winkel.
  Defaultwert: 'range'
  Werteliste: CalcMode ∈ {'range', 'mean', 'regress'}
- ▷ **Lookaround** (input_control) ..... integer ⇝ *integer*
  Größe der zu betrachtenden Nachbarschaft.
  Defaultwert: 3
  Restriktion: Lookaround > 0
- ▷ **Angles** (output_control) ..... real-array ⇝ *real*
  Richtung der Konturpunkte.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Siehe auch

get_contour_xld, get_contour_attrib_xld

Modul

Foundation

get_contour_attrib_xld (Contour : : Name : Attrib)

Auslesen von Punktattributen einer XLD-Kontur.

get_contour_attrib_xld liefert für die XLD-Kontur Contour den Wert des für jeden Punkt der Kontur definierten Attributs mit Namen Name im Parameter Attrib zurück. Punktattribute sind für jeden Konturpunkt definierte, zusätzliche Werte und beschreiben lokale Eigenschaften. Welche Punktattribute für eine Kontur definiert sind, kann mit query_contour_attribs_xld abgefragt werden.

Die folgende Aufzählung enthält Informationen über die verschiedenen Punktattribute und die Operatoren, mit welchen sie einer Kontur hinzufügen werden (für Ausnahmen siehe die entsprechende Operatorreferenz):

'angle'

Die Richtung der Normalenvektoren einer Kontur wird durch *'angle'* [rad] beschrieben (siehe Bild unten). Dabei sind die Vektoren so orientiert, dass sie auf die rechte Seite der Kontur zeigen (Blickrichtung entlang der Kontur von Start- nach Endpunkt). Die Winkel sind bezüglich der Zeilenachse gegen den Uhrzeigersinn gegeben.

Das Attribut *'angle'* wird durch folgende Operatoren hinzugefügt:

edges_color_sub_pix, edges_sub_pix, lines_color, lines_facet, lines_gauss

'response'

'response' gibt die Magnitude des Gradienten einer Kante an (siehe Bild unten).

Das Attribut *'response'* wird durch folgende Operatoren hinzugefügt:

edges_color_sub_pix, edges_sub_pix, lines_color, lines_facet, lines_gauss

'width_right'

Die Linienbreite rechts der Kontur (Blickrichtung entlang der Kontur von Start- nach Endpunkt) wird durch *'width_right'* [px] beschrieben (siehe Bild unten).

Das Attribut *'width_right'* wird durch folgende Operatoren hinzugefügt:

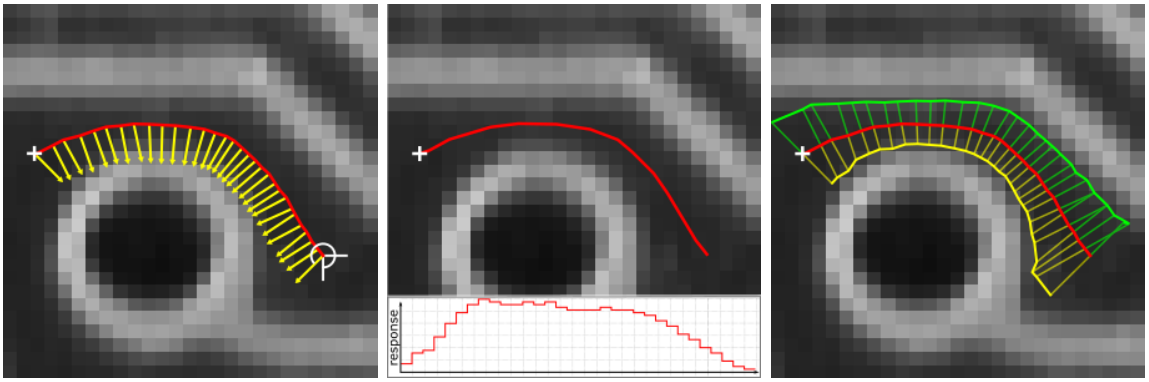
lines_color, lines_gauss

'width_left'

Die Linienbreite links der Kontur (Blickrichtung entlang der Kontur von Start- nach Endpunkt) wird durch *'width_left'* [px] beschrieben (siehe Bild unten).

Das Attribut *'width_left'* wird durch folgende Operatoren hinzugefügt:

lines_color, lines_gauss



(1) (2) (3)

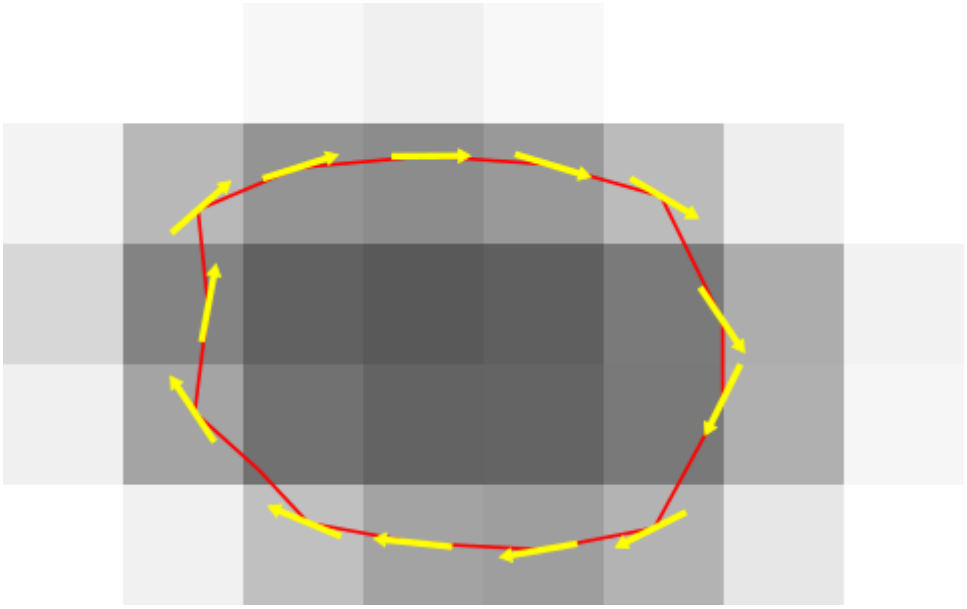
Darstellung verschiedener Attribute einer Kontur (rot). Der Startpunkt der Kontur ist mit einem weißen Kreuz markiert. (1) Die Normalenvektoren (gelb) einer Kontur werden im Winkel *'angle'* (bezüglich der Zeilenachse) dargestellt, (2) Verhalten des Attributs *'response'* entlang einer Kontur, und (3) Visualisierung der Linienbreiten *'width_right'* (gelb) und *'width_left'* (grün).

'edge_direction'

Gibt die Richtung der Kante (nicht der Kontur), berechnet aus den Grauwertgradienten in Spalten- und Zeilenrichtung, an. Die Winkel sind bezüglich der Zeilenachse gegen den Uhrzeigersinn gegeben.

Das Attribut *'edge_direction'* [rad] wird durch folgende Operatoren hinzugefügt:

`edges_color_sub_pix`, `edges_sub_pix`



Das Attribut *'edge_direction'* beschreibt die Vektoren (gelb) in Kantenrichtung für jeden Punkt der Kontur (rot).

'asymmetry'

Das Attribut *'asymmetry'* beschreibt das Verhältnis der Grauwertgradienten zu beiden Seiten der Kontur. Befindet sich der asymmetrische Bereich, d.h. der Bereich mit dem schwächeren Gradienten zur rechten Seite der Linie (Blickrichtung von Start- zu Zielpunkt), so hat *'asymmetry'* einen positiven Wert. Befindet sich der schwächere Gradient links der Linie, ist der Wert negativ (siehe Bild unten).

Das Attribut *'asymmetry'* wird durch folgenden Operator hinzugefügt:

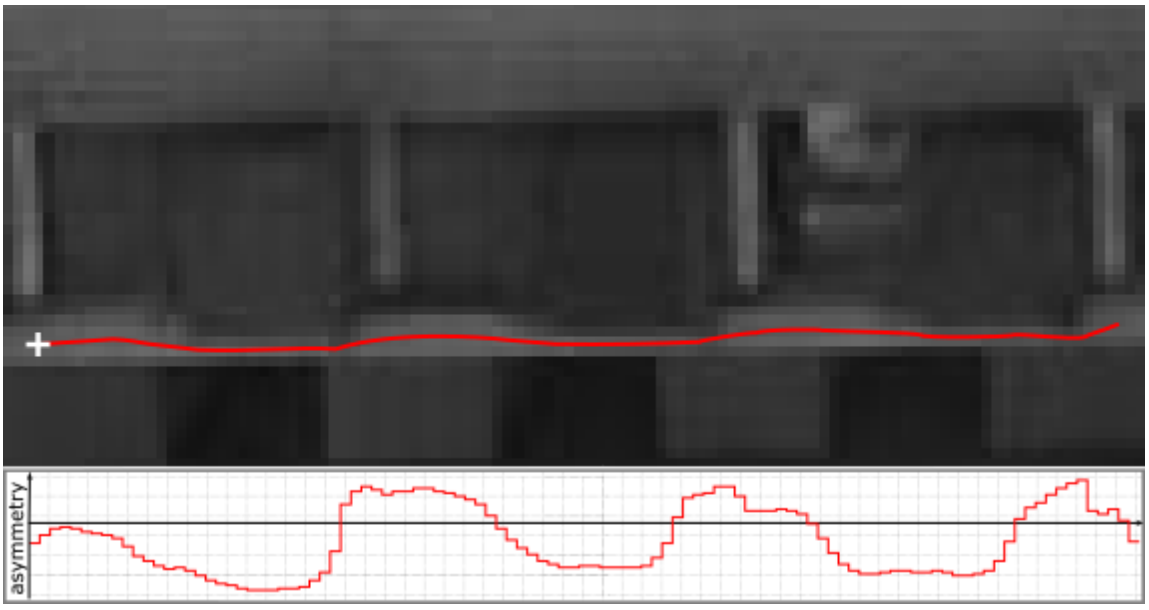
`lines_gauss`

'contrast'

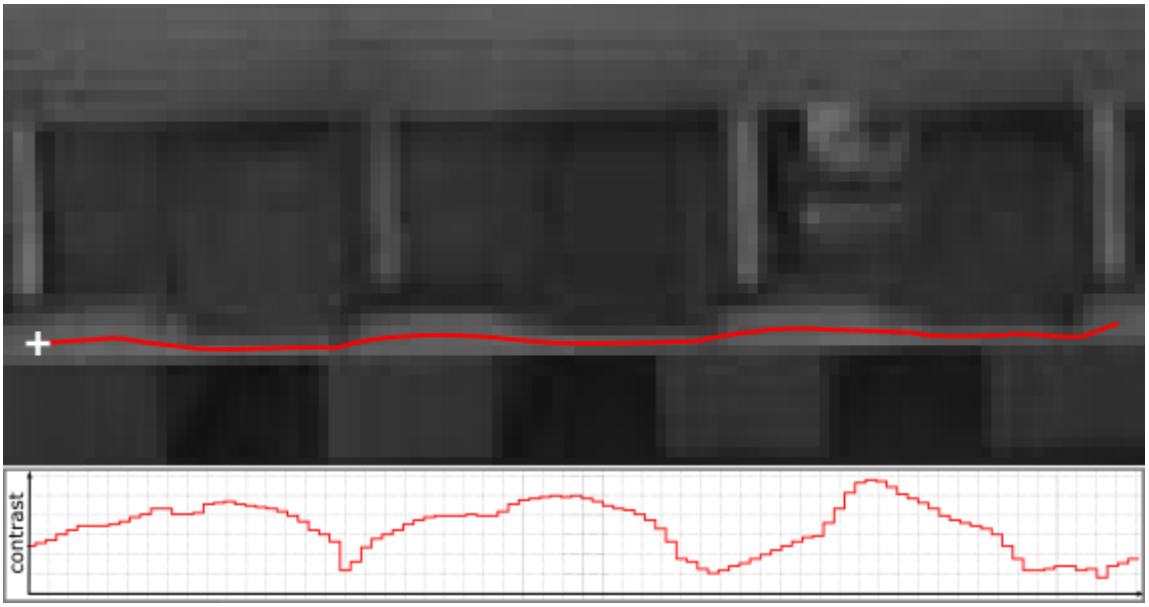
Der Kontrast einer Kontur beschreibt die Grauwertunterschiede zwischen der extrahierten Linie und dem lokalen Hintergrund. Das Attribut *'contrast'* ist positiv wenn die Linie heller als der Hintergrund ist, für dunklere Linien auf hellerem Grund ist der Wert negativ (siehe Bild unten).

Das Attribut *'contrast'* wird durch folgenden Operator hinzugefügt:

`lines_gauss`



(1)



(2)

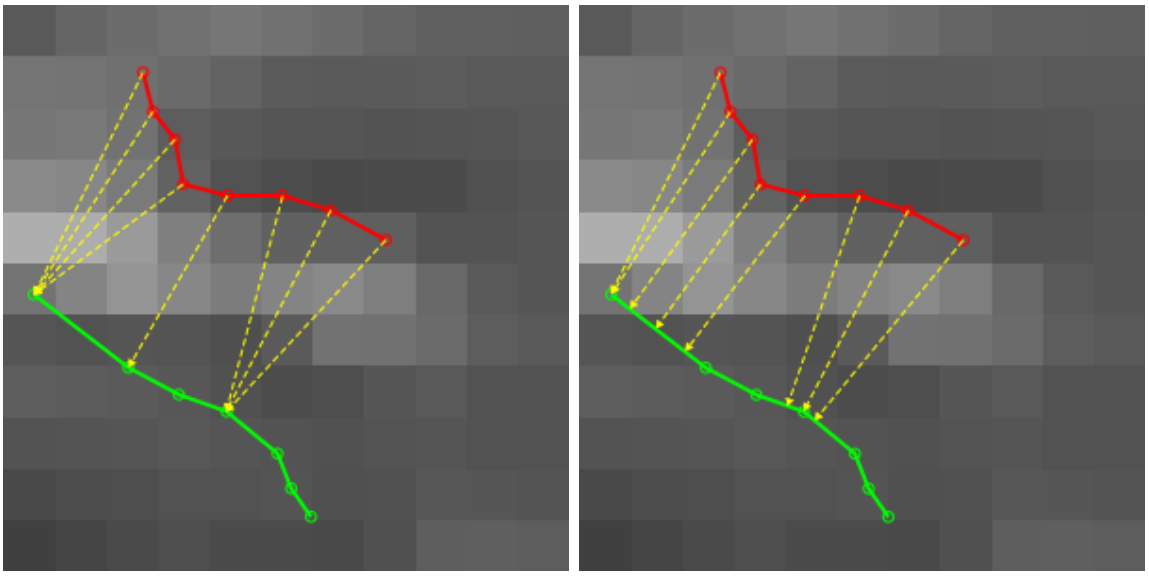
Verhalten der Attribute (1) *'asymmetry'* und (2) *'contrast'* einer Kontur entlang einer Bildstruktur (der Startpunkt der Kontur ist mit einem weißen Kreuz markiert).

'distance'

'distance' [px] gibt den minimalen Abstand zu den Stützpunkten beziehungsweise den Segmenten (je nach Berechnungsmodus) einer Referenzkontur an.

Das Attribut *'distance'* wird durch folgende Operatoren hinzugefügt:

`apply_distance_transform_xld`, `distance_contours_xld`



(1) (2)

(1) Abstand (*'distance'*) einer Kontur (rot) zu den Stützpunkten einer Referenzkontur (grün) und (2) Abstand *'distance'* zu den Segmenten einer Referenzkontur.

Für Informationen über globale Konturattribute siehe `get_contour_global_attrib_xld`.

Parameter

▷ **`Contour`** (input_object) ..... xld_cont ⇝ *object*
Eingabe-Kontur.

▷ **`Name`** (input_control) ..... string ⇝ *string*
Name des Attributs.
Defaultwert: 'angle'
Wertevorschläge: `Name` ∈ {'angle', 'edge_direction', 'width_right', 'width_left', 'response', 'contrast', 'asymmetry', 'distance'}

▷ **`Attrib`** (output_control) ..... real-array ⇝ *real*
Zurückgeliefertes Attribut.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`lines_gauss`, `lines_facet`, `edges_sub_pix`, `distance_contours_xld`, `apply_distance_transform_xld`

Siehe auch

`query_contour_attribs_xld`, `get_contour_global_attrib_xld`, `query_contour_global_attribs_xld`

Modul

Foundation

`get_contour_global_attrib_xld` (Contour : : Name : Attrib)

Auslesen von globalen Attributen einer XLD-Kontur.

`get_contour_global_attrib_xld` liefert für die XLD-Kontur `Contour` den Wert des global definierten Attributs mit Namen `Name` im Parameter `Attrib` zurück. Globale Attribute sind für eine gesamte Kontur definierte, zusätzliche Werte. Welche globalen Attribute für eine Kontur definiert sind, kann mit `query_contour_global_attribs_xld` abgefragt werden.

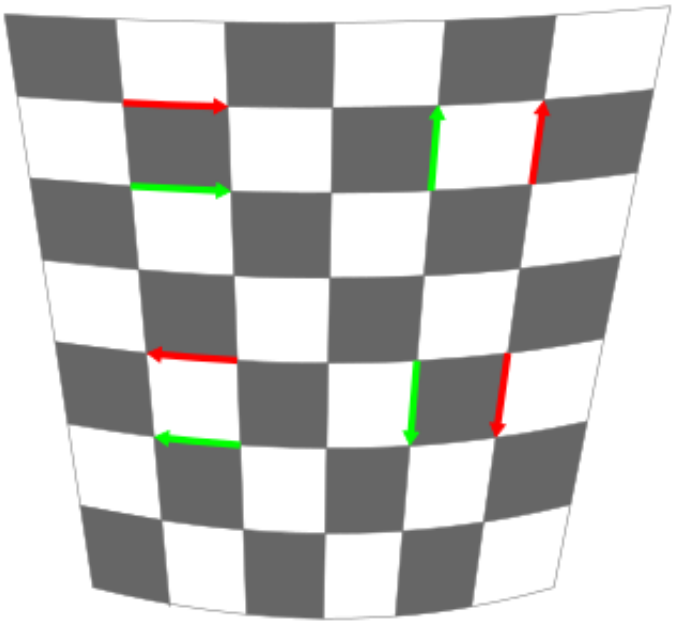
Die folgende Aufzählung enthält Informationen über die verschiedenen globalen Attribute und die Operatoren, mit welchen sie einer Kontur hinzugefügt werden:

'bright_dark'

Der Typ des Hell-Dunkel-Übergangs wird für jede Ausgabe-Kontur im Attribut *'bright_dark'* gespeichert. Das Attribut bekommt den Wert *1.0*, wenn an der Verbindungslinie ein Übergang von hell nach dunkel stattfindet (von links nach rechts, in Blickrichtung vom Anfangs- zum Endpunkt), umgekehrt entspricht der Wert *0.0*.

Das Attribut *'bright_dark'* wird durch folgenden Operator hinzugefügt:

`connect_grid_points`



Beispielkonturen (rot und grün) entlang von Kanten eines Entzerrungsgitters (Richtung von Start- zu Endpunkt durch Pfeil angedeutet). Der Typ des Hell-Dunkel-Übergangs, der im Attribut *'bright_dark'* gespeichert ist, ist *1.0* für rote und *0.0* für grüne Konturen.

'cont_approx'

Die Unterscheidung, ob eine Ausgabekontur bestenfalls durch ein Liniensegment oder einen Kreis- oder Ellipsenbogen approximiert wird, erfolgt über das globale Konturattribut *'cont_approx'*. Falls *'cont_approx'=-1*, ist ein Liniensegment , falls *'cont_approx'=0* ein Ellipsenbogen und falls *'cont_approx'=1* ein Kreisbogen am besten dazu geeignet.

Das Attribut *'cont_approx'* wird durch folgenden Operator hinzugefügt:

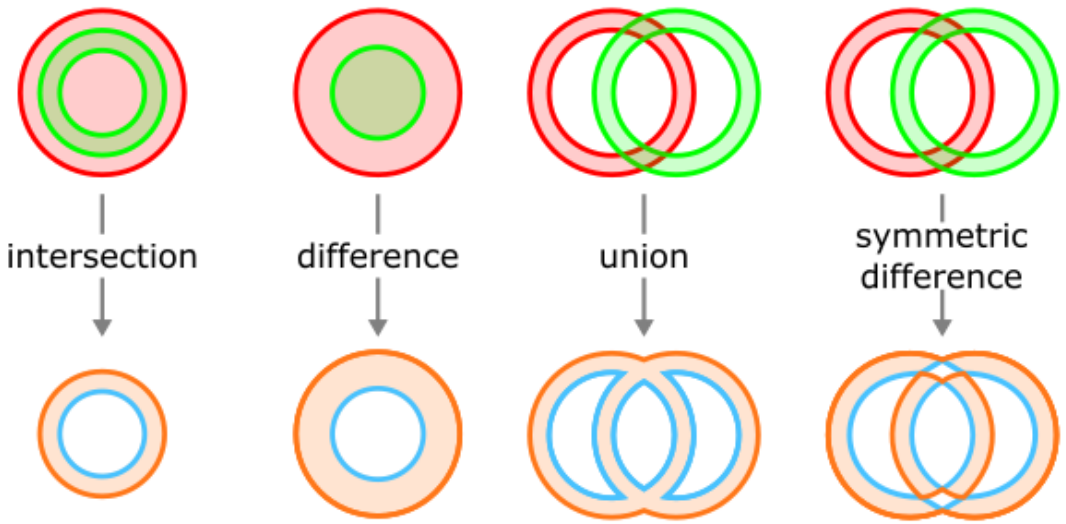
`segment_contours_xld`

'is_hole'

Für Konturen, die Löcher umschließen, wird das globale Attribut *'is_hole'* auf *1* gesetzt, ansonsten auf *0*.

Das Attribut *'is_hole'* wird durch folgende Operatoren hinzugefügt:

`symm_difference_closed_contours_xld`, `difference_closed_contours_xld`, `intersection_closed_contours_xld`, `union2_closed_contours_xld`



Durch die Anwendung verschiedener Mengenoperatoren auf Konturen wird das globale Konturattribut *'is_hole'* für die Ausgabe-Kontur(en) definiert. Im obigen Beispiel werden für zwei Kontur-Sets (rot und grün) Mengenoperationen durchgeführt. Schließen die resultierenden Konturen ein Loch ein (blau) ist *'is_hole' 1.0*, andernfalls ist *'is_hole' 0.0* (orange).

'regr_dev_dist'

'regr_dev_dist' [px] gibt die Standardabweichung des euklidischen Abstands zwischen den Stützpunkten der Kontur und der Regressionsgeraden an (siehe Bild unten).

Das Attribut *'regr_dev_dist'* wird durch den folgenden Operator hinzugefügt:

`regress_contours_xld`

'regr_dist'

'regr_dist' [px] gibt den minimalen euklidischen Abstand der Regressionsgerade vom Ursprung des Bildkoordinatensystems an (siehe Bild unten).

Das Attribut *'regr_dist'* wird durch den folgenden Operator hinzugefügt:

`regress_contours_xld`

'regr_mean_dist'

'regr_mean_dist' [px] gibt den Mittelwert der euklidischen Abstände zwischen den Stützpunkten der Kontur und der Regressionsgeraden an (siehe Bild unten).

Das Attribut *'regr_mean_dist'* wird durch den folgenden Operator hinzugefügt:

`regress_contours_xld`

'regr_norm_col'

'regr_norm_col' [px] entspricht der Spaltenkoordinate des normierten Normalenvektors der Regressionsgeraden, wobei der Normalenvektor vom Ursprung des Bildkoordinatensystems Richtung Regressionsgerade zeigt (siehe Bild unten).

Das Attribut *'regr_norm_col'* wird durch den folgenden Operator hinzugefügt:

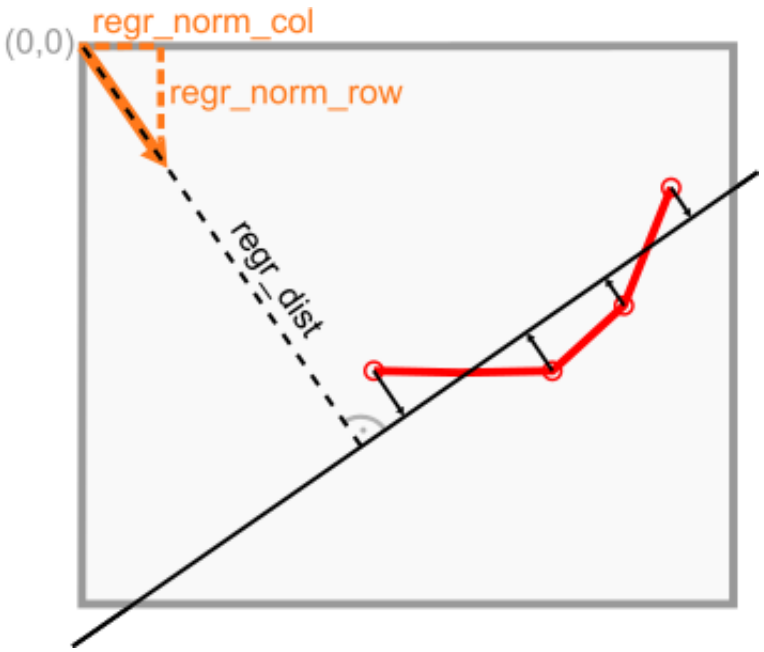
`regress_contours_xld`

'regr_norm_row'

'regr_norm_row' [px] entspricht der Zeilenkoordinate des normierten Normalenvektors der Regressionsgeraden, wobei der Normalenvektor vom Ursprung des Bildkoordinatensystems Richtung Regressionsgerade zeigt (siehe Bild unten).

Das Attribut *'regr_norm_row'* wird durch den folgenden Operator hinzugefügt:

`regress_contours_xld`



Skizze der Attribute *'regr_dist'*, *'regr_norm_col'* und *'regr_norm_row'* für die lineare Regression einer XLD Kontur (rot). *'regr_mean_dist'* und *'regr_dev_dist'* werden aus den Distanzen (schwarze Pfeile) zwischen den Stützpunkten der Kontur und der Regressionsgeraden (durchgezogene schwarze Linie) berechnet.

Parameter

▷ **Contour** (input_object) ..... xld_cont ⇝ *object*
Eingabe-Kontur.

▷ **Name** (input_control) ..... string(-array) ⇝ *string*
Name des Attributs.
Defaultwert: 'regr_norm_row'
Wertevorschläge: `Name` ∈ {'regr_norm_row', 'regr_norm_col', 'regr_mean_dist', 'regr_dev_dist', 'cont_approx', 'bright_dark', 'is_hole'}

▷ **Attrib** (output_control) ..... real-array ⇝ *real*
Zurückgeliefertes Attribut.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`lines_gauss`, `lines_facet`, `edges_sub_pix`, `segment_contours_xld`

Nachfolger

`fit_circle_contour_xld`, `fit_ellipse_contour_xld`, `fit_line_contour_xld`, `fit_rectangle2_contour_xld`

Siehe auch

`query_contour_global_attribs_xld`, `get_contour_attrib_xld`, `query_contour_attribs_xld`

Modul

Foundation

get_regress_params_xld (Contours : : : Length, Nx, Ny, Dist, Fpx, Fpy, Lpx, Lpy, Mean, Deviation)

Ausgeben von Konturparametern.

Diese Routine gibt folgende Konturparameter für alle Konturen des Tupels `Contours` als Tupel aus:

- die Anzahl der Konturpunkte `Length`,
- die Koordinaten `Nx` und `Ny` des Normalenvektors der Regressionsgeraden
- den Abstand `Dist` der Regressionsgeraden vom Ursprung
- die subpixelgenauen Koordinaten `Fpx` und `Fpy` des Fußpunktes des Lotes das ersten Endpunktes auf die optimale Regressionsgerade,
- die subpixelgenauen Koordinaten `Lpx` und `Lpy` des Fußpunktes des Lotes das zweiten Endpunktes auf die optimale Regressionsgerade,
- den Mittelwert des absoluten euklidischen Abstandes der Konturpunkte zur optimalen Regressionsgeraden,
- die Standardabweichung dieser Abstände.

Achtung

Bevor Konturinformationen mit `get_regress_params_xld` ausgegeben werden können, müssen die Regressionsgeradenparameter mit `regress_contours_xld` berechnet werden.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont-array ⇝ *object*
  Eingabe-Konturen.
- ▷ **`Length`** (output_control) ..... integer-array ⇝ *integer*
  Länge der Kontur.
- ▷ **`Nx`** (output_control) ..... point.x-array ⇝ *real*
  X-Koordinate des Normalenvektors.
- ▷ **`Ny`** (output_control) ..... point.y-array ⇝ *real*
  Y-Koordinate des Normalenvektors.
- ▷ **`Dist`** (output_control) ..... number-array ⇝ *real*
  Abstand der Regressionsgerade vom Ursprung.
- ▷ **`Fpx`** (output_control) ..... point.x-array ⇝ *real*
  X-Koordinate des ersten Endpunkts.
- ▷ **`Fpy`** (output_control) ..... point.y-array ⇝ *real*
  Y-Koordinate des ersten Endpunkts.
- ▷ **`Lpx`** (output_control) ..... point.x-array ⇝ *real*
  X-Koordinate des zweiten Endpunkts.
- ▷ **`Lpy`** (output_control) ..... point.y-array ⇝ *real*
  Y-Koordinate des zweiten Endpunkts.
- ▷ **`Mean`** (output_control) ..... real-array ⇝ *real*
  Mittlerer Abstand der Konturpunkte von der Regressionsgerade.
- ▷ **`Deviation`** (output_control) ..... real-array ⇝ *real*
  Standardabweichung des Abstands der Punkte von der Geraden.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`regress_contours_xld`

Nachfolger

`disp_line`, `line_orientation`

Siehe auch

`fit_line_contour_xld`, `get_contour_global_attrib_xld`, `query_contour_global_attribs_xld`, `get_contour_xld`, `get_contour_attrib_xld`, `gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Modul

Foundation

height_width_ratio_xld (XLD : : : Height, Width, Ratio)

Ermitteln der Breite, Höhe und des Seitenverhältnisses des umschließenden Rechtecks parallel zu den Koordinatenachsen von Konturen oder Polygonen.

height_width_ratio_xld berechnet das umschließende Rechteck (parallel zu den Koordinatenachsen) für jede Eingabekontur bzw. jedes Eingabepolygon. Das umschließende Rechteck wird durch die Koordinaten der Eckpunkte (Row1,Column1,Row2,Column2) beschrieben. Basierend auf diesen Werten werden die Breite, Höhe und das Seitenverhältnis wie folgt berechnet:

$$\begin{aligned} \texttt{Width} &= \texttt{Column2} - \texttt{Column1} \\ \texttt{Height} &= \texttt{Row2} - \texttt{Row1} \\ \texttt{Ratio} &= \texttt{Height}/\texttt{Width} \end{aligned}$$

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert. Bei leerer Kontur haben alle Parameter den Wert 0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Achtung

Falls Width = 0, wird Ratio auf DBL_MAX (1.797e308) gesetzt.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen oder Polygone.
- ▷ **Height** (output_control) ..... extent.y(-array) ⇝ *real*
  Höhe des umschließenden Rechtecks.
- ▷ **Width** (output_control) ..... extent.x(-array) ⇝ *real*
  Breite des umschließenden Rechtecks.
- ▷ **Ratio** (output_control) ..... real(-array) ⇝ *real*
  Seitenverhältnis des umschließenden Rechtecks.

Komplexität

Sei N die Anzahl der Punkte in der Kontur, dann beträgt die Laufzeitkomplexität $O(N)$.

Ergebnis

height_width_ratio_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld

Nachfolger

gen_polygons_xld

Alternativen

smallest_rectangle1_xld, smallest_rectangle2_xld, shape_trans_xld

Siehe auch

shape_trans_xld, smallest_circle_xld, elliptic_axis_xld, area_center_xld

Modul

Foundation

info_parallels_xld (Parallels, Image : : : QualityMin, QualityMax, GrayMin, GrayMax, StandardMin, StandardMax)

Abrufen von Informationen zur Fläche, die von zwei parallelen Linien umschlossen ist.

info_parallels_xld charakterisiert die von je zwei parallelen Linien, die unter Parallels abgespeichert sind, eingeschlossene Fläche. Dabei wird das Bild Image herangezogen, um die Grauwertbeschaffenheit dieser Fläche zu untersuchen. Der Algorithmus dieser Routine entspricht im wesentlichen dem von mod_parallels_xld. Es werden die Werte Qualitäts-Faktor (QualityMin und QualityMax, siehe gen_parallels_xld für die entsprechende Berechnung), mittlerer Grauwert (GrayMin und GrayMax) und mittlere Standardabweichung (StandardMin und StandardMax) bzgl. des zugehörigen mittleren Grauwerts zurückgeliefert.

Dieser Operator dient zur Bestimmung von geeigneten Schwellenwerten für mod_parallels_xld.

Parameter

- ▷ **Parallels** (input_object) ..... xld_para-array ⇝ *object*
  Parallele Polygonteilstücke.
- ▷ **Image** (input_object) ..... singlechannelimage ⇝ *object : byte*
  Zu untersuchendes Eingabebild.
- ▷ **QualityMin** (output_control) ..... real ⇝ *real*
  Minimaler Qualitäts-Faktor.
- ▷ **QualityMax** (output_control) ..... real ⇝ *real*
  Maximaler Qualitäts-Faktor.
- ▷ **GrayMin** (output_control) ..... integer ⇝ *integer*
  Minimaler mittlerer Grauwert.
- ▷ **GrayMax** (output_control) ..... integer ⇝ *integer*
  Maximaler mittlerer Grauwert.
- ▷ **StandardMin** (output_control) ..... real ⇝ *real*
  Minimale Standardabweichung.
- ▷ **StandardMax** (output_control) ..... real ⇝ *real*
  Maximale Standardabweichung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_parallels_xld

Nachfolger

mod_parallels_xld

Siehe auch

intensity, min_max_gray

Modul

Foundation

length_xld (XLD : : : Length)

Länge von Konturen oder Polygonen.

length_xld berechnet die Gesamtlänge der Konturen oder der Polygone, die in XLD übergeben werden. Die Länge ergibt sich als Summe der Euklidschen Abstände der einzelnen Kurvenpunkte. Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

▷ **XLD** (input_object) . . . . . xld(-array) ⇝ *object*
Zu untersuchende Konturen bzw. Polygone.

▷ **Length** (output_control) . . . . . real(-array) ⇝ *real*
Länge der Kontur bzw. des Polygons.
Zusicherung: Length >= 0

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

`length_xld` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`gen_contours_skeleton_xld`, `smooth_contours_xld`, `gen_polygons_xld`

Siehe auch

`area_center_xld`, `moments_any_xld`, `moments_xld`, `contlength`

Modul

Foundation

local_max_contours_xld (Contours,
Image : LocalMaxContours : MinPercent, MinDiff, Distance :)

Selektieren von Konturen, deren Grauwerte ein lokales Maximum sind.

Die Funktion `local_max_contours_xld` wählt aus den Konturen, die in `Contours` übergeben werden, diejenigen Konturen aus, deren Grauwerte ein lokales Maximum im Eingabebild `Image` darstellen. Damit eine Kontur als lokales Maximum gilt, müssen mindestens `MinPercent` der Konturpunkte ein lokales Maximum senkrecht zur Kontur-Richtung sein. Die Kontur-Richtung wird berechnet, indem eine Regressionsgerade durch fünf umgebende Punkte gelegt wird. Zur Entscheidung, ob ein lokales Maximum vorliegt, wird ein Grauwertprofil in beide Normalenrichtungen bis zum Abstand `Distance` vom aktuellen Konturpunkt betrachtet. Der Grauwert an der Konturposition in beiden Richtungen um mindestens `MinDiff` größer sein, als der Grauwert an einer der Positionen des Profils, um als Maximum zu gelten. Die selektierten Konturen werden in `LocalMaxContours` zurückgegeben.

Parameter

▷ **Contours** (input_object) . . . . . xld_cont-array ⇝ *object*
Zu untersuchende Konturen.

▷ **Image** (input_object) . . . . . singlechannelimage ⇝ *object : byte*
Zu untersuchendes Eingabebild.

▷ **LocalMaxContours** (output_object) . . . . . xld_cont-array ⇝ *object*
Selektierte Konturen.

▷ **MinPercent** (input_control) . . . . . number ⇝ *integer* / real
Mindestanteil von Maximumspunkten.
Defaultwert: 70
Wertevorschläge: `MinPercent` ∈ {60, 70, 75, 80, 85, 90, 95}
Restriktion: 0.0 <= MinPercent && MinPercent <= 100.0

▷ **MinDiff** (input_control) . . . . . integer ⇝ *integer*
Minimale Differenz, um die das Maximum größer sein muss als die Werte am Rand.
Defaultwert: 15
Wertevorschläge: `MinDiff` ∈ {5, 8, 10, 12, 15, 20}
Restriktion: 0 <= MinDiff && MinDiff <= 255

▷ **Distance** (input_control) . . . . . integer ⇝ *integer*
Maximalabstand der Punkte, die zur Bestimmung, ob ein Punkt ein lokales Maximum ist, verwendet werden, zum Punkt auf der Kontur.
Defaultwert: 4
Wertevorschläge: `Distance` ∈ {2, 3, 4, 5, 6}
Restriktion: Distance >= 1

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Nachfolger

`gen_polygons_xld`

Siehe auch

`smooth_contours_xld`

Modul

Foundation

`max_parallels_xld` (ExtParallels : MaxPolygons : :)

Berechnen paralleler Polygone mit maximaler Ausdehnung.

Mit `max_parallels_xld` werden alle modifizierten parallelen Polygonteilstücke, die unter `ExtParallels` abgespeichert sind, zu einem Polygonstück zusammengefasst. Die gefundenen Polygone werden mit ihrem kleinsten Anfangsindex und größtem Endindex abgespeichert. Es werden somit Polygone, die an mehreren Stellen Parallelität aufweisen, als ein langes Polygonstück (von der ersten Linie bis zur letzten parallelen Linie) in `MaxPolygons` ausgegeben.

Parameter

▷ **ExtParallels** (input_object) . . . . . xld_ext_para-array ⇝ *object*
Erweiterte Parallelen.

▷ **MaxPolygons** (output_object) . . . . . xld_poly-array ⇝ *object*
Maximal erweiterte Polygonstücke.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`mod_parallels_xld`

Nachfolger

`get_polygon_xld`, `get_lines_xld`

Modul

Foundation

moments_any_points_xld (XLD : : Mode, Area, CenterRow, CenterCol, P, Q : M)

Geometrische Momente beliebiger Ordnung von Konturen oder Polygonen, die wie Punktwolken behandelt werden.

moments_any_points_xld berechnet Momente beliebiger Ordnung der durch Konturen oder Polygonen XLD gegebenen Punktwolken (d.h. die Reihenfolge der Kontur- oder Polygonpunkte wird nicht berücksichtigt).

Abhängig vom Parameter Mode werden die berechneten Momente wie folgt normiert:

'unnormalized': Keine Normalisierung. Das berechnete Moment ist:

$$M_{p,q} = \sum_{i=0}^{n-1} r_i^p c_i^q$$

'unnormalized_central': Verschiebung der Konturpunkte um ihren Schwerpunkt $[\bar{r}, \bar{c}]$ = [CenterRow, CenterCol]:

$$M_{p,q} = \sum_{i=0}^{n-1} (r_i - \bar{r})^p (c_i - \bar{c})^q$$

'normalized': Normalisierung durch die Fläche A = Area von XLD:

$$M_{p,q} = \frac{1}{A} \sum_{i=0}^{n-1} r_i^p c_i^q$$

'central': Normalisierung durch die Fläche A = Area und zusätzlich Verschiebung der Bildregion um ihren Schwerpunkt $[\bar{r}, \bar{c}]$ = [CenterRow, CenterCol]:

$$M_{p,q} = \frac{1}{A} \sum_{i=0}^{n-1} (r_i - \bar{r})^p (c_i - \bar{c})^q$$

Für die Normierung der Momente werden ggf. drei spezielle Werte benötigt: Die Fläche Area sowie die Koordinaten CenterRow,CenterRow des Schwerpunktes (diese können mittels area_center_points_xld berechnet werden).

Bei geschlossenen Konturen oder Polygonen (Anfangspunkt = Endpunkt) wird der Endpunkt der Kontur oder des Polygons nicht berücksichtigt, da er sonst doppeltes Gewicht erhielte.

moments_any_points_xld sollte dann verwendet werden, wenn die Kontur XLD entweder nicht kreuzungsfrei ist oder nicht kreuzungsfrei durch eine Line vom End- zum Anfangspunkt geschlossen werden kann. In diesem Fall liefert moments_xld keine sinnvollen Ergebnisse. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Momente für die einzelnen Konturen bzw. Polygone hintereinander in M abgelegt.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus.
  Defaultwert: 'unnormalized'
  Wertevorschläge: Mode ∈ {'unnormalized', 'unnormalized_central', 'normalized', 'central'}
- ▷ **Area** (input_control) ..... real(-array) ⇝ *real*
  Fläche der Punktwolke.
- ▷ **CenterRow** (input_control) ..... point.y(-array) ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **CenterCol** (input_control) ..... point.x(-array) ⇝ *real*
  Spaltenindex des Schwerpunktes.

▷ **P** (input_control) ..... point.x(-array) ⇝ *integer*
Erster Index der zu berechnenden Momente $M_{p,q}$.
Defaultwert: 1
Wertevorschläge: P ∈ {0, 1, 2, 3, 4}

▷ **Q** (input_control) ..... point.x(-array) ⇝ *integer*
Zweiter Index der zu berechnenden Momente $M_{p,q}$.
Defaultwert: 1
Wertevorschläge: Q ∈ {0, 1, 2, 3, 4}

▷ **M** (output_control) ..... real(-array) ⇝ *real*
Die berechneten Momente.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

moments_any_points_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

area_center_points_xld, gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Alternativen

moments_points_xld, moments_any_xld

Siehe auch

moments_points_xld, area_center_points_xld, moments_region_2nd, area_center

Modul

Foundation

moments_any_xld (XLD : : Mode, PointOrder, Area, CenterRow, CenterCol, P, Q : M)

Geometrische Momente beliebiger Ordnung von Konturen oder Polygonen.

moments_any_xld berechnet Momente beliebiger Ordnung der von Konturen oder Polygonen XLD umschlossenen Bildregionen. Die Momente werden durch Anwendung des Gaußschen Integralsatzes nur unter Verwendung der Kontur- oder Polygonpunkte berechnet, d.h. es wird keine explizite Fläche zur Berechnung erzeugt. Es wird vorausgesetzt, dass die Konturen bzw. Polygone geschlossen sind. Ist das nicht der Fall, werden sie künstlich geschlossen.

Es ist zu beachten, dass moments_any_xld nur dann sinnvolle Ergebnisse liefert, falls die Kontur oder das Polygon eine Fläche in der Ebene einschließt. Insbesondere darf sich die Kontur oder das Polygon nicht selbst schneiden. Dies ist insbesondere dann zu beachten, wenn offene Konturen und Polygone übergeben werden, da diese automatisch geschlossen werden, was dazu führen kann, dass sich die Kontur oder das Polygon selbst schneidet. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden. Falls sich die Kontur oder das Polygon selbst schneidet, kann ein sinnvoller Wert für die Momente mit moments_any_points_xld berechnet werden.

Abhängig vom Parameter Mode werden die berechneten Momente wie folgt normiert: Mode:

'unnormalized': Keine Normalisierung. Sei R die eingeschlossene Bildregion. So entspricht das berechnete Moment $M_{p,q}$

$$M_{p,q} = \int_R r^p c^q \, dr \, dc$$

'unnormalized_central': Verschiebung der Bildregion um ihren Schwerpunkt $[\bar{r}, \bar{c}]$ = [CenterRow, CenterCol]:

$$M_{p,q} = \int_R (r - \bar{r})^p (c - \bar{c})^q \, dr \, dc$$

'normalized': Normalisierung durch die Fläche $A =$ Area der eingeschlossenen Bildregion:

$$M_{p,q} = \frac{1}{A} \int_R r^p c^q \, dr \, dc$$

'central': Normalisierung durch die Fläche $A =$ Area der eingeschlossenen Bildregion und zusätzlich Verschiebung der Bildregion um ihren Schwerpunkt $[\bar{r}, \bar{c}] =$ [CenterRow, CenterCol]:

$$M_{p,q} = \frac{1}{A} \int_R (r - \bar{r})^p (c - \bar{c})^q \, dr \, dc$$

Für die Normierung der Momente werden ggf. drei spezielle Momente benötigt: Die Fläche Area der eingeschlossenen Bildregion sowie die Koordinaten CenterRow,CenterCol des Schwerpunktes, vgl. area_center_xld. Außerdem erwartet moments_any_xld den Umlaufsinn der Eingabekonturen bzw. -polygone in PointOrder (diese Werte können mit area_center_xld berechnet werden). Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Momente für die einzelnen Konturen bzw. Polygone hintereinander in M abgelegt.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus.
  Defaultwert: 'unnormalized'
  Wertevorschläge: Mode ∈ {'unnormalized', 'unnormalized_central', 'normalized', 'central'}
- ▷ **PointOrder** (input_control) ..... string(-array) ⇝ *string*
  Mathematischer Umlaufsinn der Stützpunkte.
  Defaultwert: 'positive'
  Wertevorschläge: PointOrder ∈ {'positive', 'negative'}
- ▷ **Area** (input_control) ..... real(-array) ⇝ *real*
  Fläche, die von der Kontur eingeschlossen wird.
- ▷ **CenterRow** (input_control) ..... point.y(-array) ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **CenterCol** (input_control) ..... point.x(-array) ⇝ *real*
  Spaltenindex des Schwerpunktes.
- ▷ **P** (input_control) ..... point.x(-array) ⇝ *integer*
  Erster Index der zu berechnenden Momente $M_{p,q}$.
  Defaultwert: 1
  Wertevorschläge: P ∈ {0, 1, 2, 3, 4}
- ▷ **Q** (input_control) ..... point.x(-array) ⇝ *integer*
  Zweiter Index der zu berechnenden Momente $M_{p,q}$.
  Defaultwert: 1
  Wertevorschläge: Q ∈ {0, 1, 2, 3, 4}
- ▷ **M** (output_control) ..... real(-array) ⇝ *real*
  Die berechneten Momente.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

moments_any_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

area_center_xld, gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Alternativen

moments_xld, moments_any_points_xld

Siehe auch

moments_xld, area_center_xld, moments_region_2nd, area_center

Modul

Foundation

moments_points_xld (XLD : : : M11, M20, M02)

Geometrische Momente M_{20}, M_{02} und M_{11} von Konturen oder Polygonen die als Punktwolke betrachtet werden.

moments_points_xld berechnet die Momente (M20, M02) und das Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen (M11) der von den Eingabekonturen bzw. -polygonen definierten Punktwolken. Die Momente ergeben sich damit durch

$$M_{p,q} = \sum_{i=0}^{n-1} (r_i - \bar{r})^p (c_i - \bar{c})^q.$$

$(\bar{r}, \bar{c})$ bezeichnet dabei den Schwerpunkt und n die Anzahl der Punkte.

Bei geschlossenen Konturen oder Polygonen (Anfangspunkt = Endpunkt) wird der Endpunkt der Kontur oder des Polygons nicht berücksichtigt, da er sonst doppeltes Gewicht erhielte.

moments_points_xld sollte dann verwendet werden, wenn die Kontur XLD entweder nicht kreuzungsfrei ist oder nicht kreuzungsfrei durch eine Line vom End- zum Anfangspunkt geschlossen werden kann. In diesem Fall liefert moments_xld keine sinnvollen Ergebnisse. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der gleichen Reihenfolge wie die entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **M11** (output_control) ..... real(-array) ⇝ *real*
  Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen.
- ▷ **M20** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung in Zeilenrichtung.
- ▷ **M02** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung in Spaltenrichtung.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

moments_points_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).

- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Alternativen

moments_xld, moments_any_xld, moments_any_points_xld

Siehe auch

moments_any_points_xld, area_center_points_xld, moments_region_2nd, area_center

Modul

Foundation

moments_xld (XLD : : : M11, M20, M02)

Geometrische Momente M_{20}, M_{02} und M_{11} von Konturen oder Polygonen.

moments_xld berechnet die Momente (M20, M02) und das Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen (M11). Zur Definition dieser Größen siehe moments_region_2nd. Die Momente werden durch Anwendung des Gaußschen Integralsatzes nur unter Verwendung der Kontur- oder Polygonpunkte berechnet, d.h. es wird keine explizite Fläche zur Berechnung erzeugt. Es wird vorausgesetzt, dass die Konturen bzw. Polygone geschlossen sind. Ist das nicht der Fall, werden sie künstlich geschlossen.

Es ist zu beachten, dass moments_xld nur dann sinnvolle Ergebnisse liefert, falls die Kontur oder das Polygon eine Fläche in der Ebene einschließt. Insbesondere darf sich die Kontur oder das Polygon nicht selbst schneiden. Dies ist insbesondere dann zu beachten, wenn offene Konturen und Polygone übergeben werden, da diese automatisch geschlossen werden, was dazu führen kann, dass sich die Kontur oder das Polygon selbst schneidet. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden. Falls sich die Kontur oder das Polygon selbst schneidet, können sinnvolle Werte für die Momente mit moments_points_xld berechnet werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen bzw. Polygone.

▷ **M11** (output_control) ..... real(-array) ⇝ *real*
Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen.

▷ **M20** (output_control) ..... real(-array) ⇝ *real*
Moment 2. Ordnung in Zeilenrichtung.

▷ **M02** (output_control) ..... real(-array) ⇝ *real*
Moment 2. Ordnung in Spaltenrichtung.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

moments_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Alternativen

moments_any_xld, moments_points_xld, moments_any_points_xld

Siehe auch

moments_any_xld, area_center_xld, moments_region_2nd, area_center

Modul

Foundation

orientation_points_xld (XLD : : : Phi)

Orientierung von Konturen oder Polygonen, die wie Punktwolken behandelt werden.

orientation_points_xld berechnet die Orientierung Phi der von den Eingabekonturen oder -polygonen XLD definierten Punktwolken (d.h. die Reihenfolge der Kontur- oder Polygonpunkte wird nicht berücksichtigt). Der Winkel Phi wird analog zu elliptic_axis_points_xld bestimmt. Es wird jedoch zusätzlich der Punkt p_m mit maximalem Abstand bestimmt. Bei geschlossenen Konturen oder Polygonen (Anfangspunkt = Endpunkt) wird der Endpunkt der Kontur oder des Polygons nicht berücksichtigt, da er sonst doppeltes Gewicht erhielte.

Falls der Winkel zwischen dem Vektor vom Mittelpunkt zum Punkt $\overrightarrow{p_m c}$ und dem durch Phi bestimmten Vector größer als π ist, wird π auf den den Winkel Phi addiert. Für den Fall, dass XLD nur aus zwei Punkten besteht, ergibt sich Phi aus der Richtung vom ersten zum zweiten Punkt.

orientation_points_xld sollte dann verwendet werden, wenn die Kontur XLD entweder nicht kreuzungsfrei ist oder nicht kreuzungsfrei durch eine Line vom End- zum Anfangspunkt geschlossen werden kann. In diesem Fall liefert orientation_xld keine sinnvollen Ergebnisse. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse für die Orientierungen in einem Tupel in der gleichen Reihenfolge wie die entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen bzw. Polygone.

▷ **Phi** (output_control) ..... angle.rad(-array) ⇝ *real*
Orientierung der Konturen bzw. Polygone (Bogenmaß).
Zusicherung: - pi < Phi && Phi <= pi

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld

Alternativen

orientation_xld, elliptic_axis_points_xld, smallest_rectangle2_xld

Siehe auch

moments_region_2nd

Modul

Foundation

orientation_xld (XLD : : : Phi)

Orientierung von Konturen oder Polygonen.

orientation_xld berechnet die Orientierung (Phi) für jede Eingabekontur bzw. jedes Eingabepolygon. Das Verfahren basiert auf dem Winkel, der mit elliptic_axis_xld bestimmt wird. Es wird jedoch zusätzlich der Punkt auf der Kontur bzw. dem Polygon mit maximalem Abstand bestimmt. Ist dessen Spaltenkoordinate kleiner als die Spaltenkoordinate des Schwerpunkts, so wird π zu dem Winkel hinzugezählt. Es wird vorausgesetzt, dass die Kontur geschlossen ist. Ist das nicht der Fall, wird sie künstlich geschlossen.

Es ist zu beachten, dass orientation_xld nur dann sinnvolle Ergebnisse liefert, falls die Kontur oder das Polygon eine Fläche in der Ebene einschließt. Insbesondere darf sich die Kontur oder das Polygon nicht selbst schneiden. Dies ist insbesondere dann zu beachten, wenn offene Konturen und Polygone übergeben werden, da diese automatisch geschlossen werden, was dazu führen kann, dass sich die Kontur oder das Polygon selbst schneidet. Ob sich die Konturen oder Polygone selbst schneiden, kann leicht mit test_self_intersection_xld überprüft werden. Falls sich die Kontur oder das Polygon selbst schneidet, kann ein sinnvoller Wert für die Orientierung mit orientation_points_xld berechnet werden.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse für die Orientierungen in einem Tupel in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen bzw. Polygone.

▷ **Phi** (output_control) ..... angle.rad(-array) ⇝ *real*
Orientierung der Konturen bzw. Polygone (Bogenmaß).
Zusicherung: - pi < Phi && Phi <= pi

Ergebnis

orientation_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Alternativen

elliptic_axis_xld, smallest_rectangle2_xld

Siehe auch

moments_region_2nd

Modul

Foundation

query_contour_attribs_xld (Contour : : : Attribs)

Auslesen der Namen von definierten Attributen einer XLD-Kontur.

query_contour_attribs_xld liefert für die XLD-Kontur Contour die Namen aller für jeden Punkt der Kontur definierten Attribute in Attribs zurück. Attribute sind für jeden Konturpunkt definierte, zusätzliche Werte, wie z.B. die Richtung senkrecht zur Kontur (*'angle'*). Operatoren, die solche Attribute definieren, enthalten eine Beschreibung der Namen und der Semantik der von ihnen definierten Werte. Die Werte des Attributs können mit get_contour_attrib_xld ausgelesen werden.

Einen Überblick über die verschiedenen Konturattribute und die Operatoren, welche sie den Konturpunkten hinzufügen, gibt die Operator-Referenz von get_contour_attrib_xld.

Parameter

▷ **Contour** (input_object) . . . . . xld_cont ⇝ *object*
Eingabe-Kontur.

▷ **Attribs** (output_control) . . . . . string-array ⇝ *string*
Liste der definierten Kontur-Attribute.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

lines_gauss, lines_facet, edges_sub_pix

Siehe auch

get_contour_attrib_xld, get_contour_global_attrib_xld, query_contour_global_attribs_xld

Modul

Foundation

query_contour_global_attribs_xld (Contour : : : Attribs)

Auslesen der Namen von definierten globalen Attributen einer XLD-Kontur.

query_contour_global_attribs_xld liefert für die XLD-Kontur Contour die Namen aller für die Kontur definierten globalen Attribute in Attribs zurück. Globale Attribute sind für eine gesamte Kontur definierte, zusätzliche Werte, wie z.B. der Normalenvektor der Regressionsgerade einer Kontur (*'regr_norm_row'* und *'regr_norm_col'*). Operatoren, die globale Konturattribute definieren, enthalten eine Beschreibung der Namen und der Semantik der von ihnen definierten Werte. Der Wert des Attributs kann mit get_contour_global_attrib_xld ausgelesen werden.

Einen Überblick über die verschiedenen globalen Konturattribute und die Operatoren, welche sie den Konturpunkten hinzufügen, gibt die Operator-Referenz von get_contour_global_attrib_xld.

Parameter

▷ **Contour** (input_object) . . . . . xld_cont ⇝ *object*
Eingabe-Kontur.

▷ **Attribs** (output_control) . . . . . string-array ⇝ *string*
Liste der definierten globalen Kontur-Attribute.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

lines_gauss, lines_facet, edges_sub_pix

Siehe auch

get_contour_global_attrib_xld, get_contour_attrib_xld, query_contour_attribs_xld

Modul

Foundation

rectangularity_xld (XLD : : : Rectangularity)

Formfaktor für die Rechteckigkeit von Konturen oder Polygonen.

rectangularity_xld berechnet die Rechteckigkeit für jede Eingabekontur bzw. jedes Eingabepolygon in XLD. Es muss darauf geachtet werden, dass sich die Eingabekonturen bzw. -polygone nicht selbst schneiden, da ansonsten der Ausgabeparameter keinen aussagekräftigen Wert enthält (Ob sich die Eingabekonturen bzw. -polygone selbst schneiden oder nicht, lässt sich mit test_self_intersection_xld überprüfen). Sind die Eingabekonturen bzw. -polygone nicht geschlossen, so werden sie vor der Berechnung automatisch geschlossen.

Zur Bestimmung der Rechteckigkeit wird zunächst ein Rechteck ermittelt, das die gleichen ersten und zweiten Momente besitzt wie die Eingabekontur bzw. das Eingabepolygon. Die Berechnung des Rechteckigkeitsmaßes basiert schließlich auf der Fläche der Differenz zwischen dem ermittelten Rechteck und der Eingabekontur bzw. des Eingabepolygons normiert auf die Fläche des Rechtecks.

Für Rechtecke liefert rectangularity_xld den Wert 1 zurück. Je weniger die Eingabekonturen bzw. -polygone einem Rechteck gleichen, desto geringer ist Rectangularity. Wird mehr als eine Kontur bzw. ein Polygon übergeben,dann werden die Zahlenwerte für die Rechteckigkeit in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Kontur bzw. des Polygons im Eingabetupel XLD entspricht.

Achtung

Bei Eingabekonturen bzw. -polygone, deren Orientierung über Momente zweiter Ordnung nicht definiert ist (wie es z.B. beim Quadrat der Fall ist), wird für Rectangularity je nach Orientierung der Eingabekonturen bzw. -polygone ein bis zu 10% zu niedriger Wert zurückgeliefert.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Rectangularity** (output_control) ..... real(-array) ⇝ *real*
  Rechteckigkeit der Eingabekonturen bzw. -polygone.
  Zusicherung: 0 <= Rectangularity && Rectangularity <= 1.0

Ergebnis

rectangularity_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabekonturen bzw. -polygone vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Alternativen

circularity_xld, convexity_xld, compactness_xld, eccentricity_xld

Siehe auch

area_center_xld, select_shape_xld

Literatur

P. L. Rosin: „Measuring rectangularity“; Machine Vision and Applications; vol. 11; pp. 191-196; Springer-Verlag, 1999.

Modul

Foundation

select_contours_xld (Contours : SelectedContours : Feature, Min1, Max1, Min2, Max2 :)

Filtern von XLD-Konturen.

select_contours_xld filtert die Konturen Contours. Dabei kann zwischen folgenden Merkmalen Feature ausgewählt werden:

'contour_length': Alle Konturen, deren Länge kleiner als Min1 oder größer als Max1 ist, werden gelöscht, (Min2 und Max2 sind hier ohne Bedeutung).

'maximum_extent': Alle Konturen, deren maximale Ausdehnung (gemessen zwischen den acht Extrempunkten in Zeilen- und Spaltenrichtung, wie sie durch Haralick und Shapiro Computer and Robot Vision, Addison-Wesley 1992, Kapitel 3.2 beschrieben werden) kleiner als Min1 oder größer als Max1 ist, werden gelöscht, (Min2 und Max2 sind hier ohne Bedeutung).

'direction': Nur alle Konturen, deren Richtungen der optimalen Regressionsgeraden zwischen Min1 und Max1 (im Bogenmaß, mathematischer Drehsinn) liegen, werden behalten. Die Parameter Min1 und Max1 werden auf den Bereich [0,2*PI[abgebildet. (Min2 und Max2 sind hier ohne Bedeutung).

'curvature': Nur alle Konturen, deren mittlerer Abstand der Konturpunkte von der optimalen Regressionsgeraden zwischen Min1 und Max1 liegt und der Standardabweichung dieses Abstands zwischen Min2 und Max2 liegt, werden behalten.

'closed': Nur Konturen, deren Anfangs- und Endpunkte einen Abstand von maximal Max1 Pixeln voneinander haben, werden behalten. (Min1, Min2 und Max2 sind hier ohne Bedeutung).

'open': Nur Konturen, deren Anfangs- und Endpunkte einen Abstand von mindestens Min1 Pixeln voneinander haben, werden behalten. (Max1, Min2 und Max2 sind hier ohne Bedeutung).

Wenn beim Filtern nach der Krümmung Min1 = Max1 = 0 oder Min2 = Max2 = 0 gewählt wird, spielt das zugeordnete Merkmal keine einschränkende Rolle für die Selektion.

Achtung

Bevor Konturen mit select_contours_xld gemäß Merkmal *'direction'* oder *'curvature'* gefiltert werden können, müssen die Regressionsgeradenparameter mit regress_contours_xld berechnet werden. Falls dies nicht geschehen ist, ruft select_contours_xld intern regress_contours_xld mit den Parametern Mode = *'no'* und Iterations = *1* auf. Falls ein anderer Modus gewünscht wird, muss regress_contours_xld explizit aufgerufen werden.

Parameter

▷ **Contours** (input_object) ..... xld_cont-array ⇝ *object*
Eingabe-Konturen.

▷ **SelectedContours** (output_object) ..... xld_cont-array ⇝ *object*
Ausgabe-Konturen.

▷ **Feature** (input_control) ..... string ⇝ *string*
Selektionsmerkmal.
Defaultwert: 'contour_length'
Werteliste: Feature ∈ {'contour_length', 'maximum_extent', 'direction', 'curvature', 'closed', 'open'}

▷ **Min1** (input_control) ..... real ⇝ *real*
Untere Schranke.
Defaultwert: 0.5

▷ **Max1** (input_control) ..... real ⇝ *real*
Obere Schranke.
Defaultwert: 200.0

▷ **Min2** (input_control) ..... real ⇝ *real*
Untere Schranke.
Defaultwert: -0.5

▷ **Max2** (input_control) ..... real ⇝ *real*
Obere Schranke.
Defaultwert: 0.5

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

regress_contours_xld

Siehe auch

get_contour_xld, get_contour_attrib_xld, gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix, get_regress_params_xld, get_contour_global_attrib_xld, query_contour_global_attribs_xld

Literatur

R. Haralick, L. Shapiro: „Computer and Robot Vision“ Vol. 1; Kapitel 3.2, Addison-Wesley 1992

Modul

Foundation

select_shape_xld (XLD : SelectedXLD : Features, Operation, Min, Max :)

Auswahl von Konturen oder Polygonen mit Hilfe von Formmerkmalen.

select_shape_xld wählt Konturen bzw. Polygone anhand ihrer Form aus. Für jede Eingabekontur bzw. jedes Eingabepolygon aus XLD werden die angegebenen Merkmale (Features) berechnet. Wenn jedes (Operation = *'and'*) oder mindestens eines (Operation = *'or'*) der so berechneten Merkmale in den vorgegebenen Grenzen (Min,Max) liegt, wird die Kontur bzw. das Polygon in die Ausgabe übernommen (dupliziert). Um diese Grenzen nach unten oder oben offen zu lassen, kann statt eines Wertes auch *'min'* beziehungsweise *'max'* für Min und Max gesetzt werden.

Bedingung: $\mathtt{Min}_i \leq \mathtt{Features}_i(Object) \leq \mathtt{Max}_i$

Mögliche Werte für Features:

(alle Features könnten auch für Polygone verwendet werden)

'area': Fläche des Objektes (vgl. area_center_xld)

'area_points': Fläche der Konturpunkte (vgl. area_center_points_xld)

'row': Zeilenindex des Schwerpunkts (vgl. area_center_xld)

'column': Spaltenindex des Schwerpunkts (vgl. area_center_xld)

'column_points': Spaltenindex des Schwerpunkts der Konturpunkte (vgl. area_center_points_xld)

'row_points': Zeilenindex des Schwerpunkts der Konturpunkte (vgl. area_center_points_xld)

'width': Breite der Kontur (siehe height_width_ratio_xld)

'height': Höhe der Kontur (siehe height_width_ratio_xld)

'ratio': Verhältnis der Höhe zur Breite der Kontur (siehe height_width_ratio_xld)

'row1': Zeilenindex der linken oberen Ecke (vgl. Operator smallest_rectangle1_xld)

'column1': Spaltenindex der linken oberen Ecke (vgl. Operator smallest_rectangle1_xld)

'row2': Zeilenindex der rechten unteren Ecke (vgl. Operator smallest_rectangle1_xld)

'column2': Spaltenindex der rechten unteren Ecke (vgl. Operator smallest_rectangle1_xld)

'circularity': Kreisförmigkeit (vgl. circularity_xld)

'compactness': Kompaktheit (vgl. compactness_xld)

'rectangularity': Rechteckigkeit (vgl. rectangularity_xld)

'contlength': Gesamtlänge der Kontur (vgl. length_xld)

'convexity': Konvexität (vgl. convexity_xld)

'ra': Hauptradius der äquivalenten Ellipse (vgl. elliptic_axis_xld)

'rb': Nebenradius der äquivalenten Ellipse (vgl. elliptic_axis_xld)

'phi': Orientierung der äquivalenten Ellipse (vgl. elliptic_axis_xld)

'ra_points': Hauptradius der äquivalenten Ellipse der Konturpunkte (vgl. elliptic_axis_points_xld)

'rb_points': Nebenradius der äquivalenten Ellipse Konturpunkte (vgl. elliptic_axis_points_xld)

'phi_points': Orientierung der äquivalenten Ellipse Konturpunkte (vgl. `elliptic_axis_points_xld`)

'anisometry' Anisometrie (vgl. `eccentricity_xld`)

'anisometry_points' Anisometrie der Konturpunkte (vgl. `eccentricity_points_xld`)

'bulkiness' Bulkiness (vgl. `eccentricity_xld`)

'struct_factor' Strukturfaktor (vgl. `eccentricity_xld`)

'outer_radius': Radius des kleinsten umschließenden Kreises (vgl. `smallest_circle_xld`)

'max_diameter': Maximale Ausdehnung der Region (vgl. `diameter_xld`)

'orientation': Orientierung der Region (vgl. `orientation_xld`)

'orientation_points': Orientierung der Konturpunkte (vgl. `orientation_points_xld`)

'rect2_phi': Orientierung des kleinsten umschließenden Rechtecks (vgl. `smallest_rectangle2_xld`)

'rect2_len1': Halbe Länge des kleinsten umschließenden Rechtecks (vgl. `smallest_rectangle2_xld`)

'rect2_len2': Halbe Breite des kleinsten umschließenden Rechtecks (vgl. `smallest_rectangle2_xld`)

'moments_m11','moments_m20','moments_m02': Geometrische Konturenmomente (vgl. `moments_xld`)

'moments_m11_points','moments_m20_points','moments_m02_points': Geometrische Momente der Konturpunkte (vgl. `moments_points_xld`)

Wird nur ein Merkmal (`Features`) verwendet, dann ist der Wert von `Operation` bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie eingegeben werden. Bei der Berechnung einiger Merkmale muss darauf geachtet werden, dass sich die Eingabekonturen bzw. -polygone nicht selbst schneiden, da ansonsten keine aussagekräftigen Ergebnisse zurückgeliefert werden können (Ob sich die Eingabekonturen bzw. -polygone selbst schneiden oder nicht, lässt sich mit `test_self_intersection_xld` überprüfen).

Parameter

▷ **`XLD`** (input_object) ..... xld-array ⇝ *object*
Konturen bzw. Polygone, die untersucht werden sollen.

▷ **`SelectedXLD`** (output_object) ..... xld-array ⇝ *object*
Konturen bzw. Polygone, die die Bedingung(en) erfüllen.

▷ **`Features`** (input_control) ..... string(-array) ⇝ *string*
Zu testende Formmerkmale.
Defaultwert: 'area'
Werteliste: `Features` ∈ {'area', 'area_points', 'row', 'row_points', 'column', 'column_points', 'width', 'height', 'ratio', 'row1', 'column1', 'row2', 'column2', 'circularity', 'compactness', 'contlength', 'convexity', 'rectangularity', 'ra', 'ra_points', 'rb', 'rb_points', 'phi', 'phi_points', 'anisometry', 'anisometry_points', 'bulkiness', 'struct_factor', 'outer_radius', 'max_diameter', 'orientation', 'orientation_points', 'rect2_phi', 'rect2_len1', 'rect2_len2', 'moments_m11', 'moments_m11_points', 'moments_m20', 'moments_m20_points', 'moments_m02', 'moments_m02_points'}

▷ **`Operation`** (input_control) ..... string ⇝ *string*
Verknüpfungsart der einzelnen Merkmale.
Defaultwert: 'and'
Werteliste: `Operation` ∈ {'and', 'or'}

▷ **`Min`** (input_control) ..... real(-array) ⇝ *real* / integer / string
Untere Grenzen der Merkmale oder 'min'.
Defaultwert: 150.0
Typischer Wertebereich: 0.0 ≤ `Min` ≤ 99999.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 1.0

▷ **`Max`** (input_control) ..... real(-array) ⇝ *real* / integer / string
Obere Grenzen der Merkmale oder 'max'.
Defaultwert: 99999.0
Typischer Wertebereich: 0.0 ≤ `Max` ≤ 99999.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 1.0
Restriktion: Max >= Min

Ergebnis

select_shape_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Nachfolger

shape_trans_xld, count_obj

Siehe auch

area_center_xld, area_center_points_xld, circularity_xld, compactness_xld, convexity_xld, elliptic_axis_xld, elliptic_axis_points_xld, eccentricity_xld, eccentricity_points_xld, rectangularity_xld, smallest_circle_xld, smallest_rectangle1_xld, smallest_rectangle2_xld, diameter_xld, orientation_xld, orientation_points_xld, moments_xld, moments_points_xld, select_obj

Modul

Foundation

select_xld_point (XLDs : DestXLDs : Row, Column :)

Auswahl aller Konturen oder Polygone, die einen übergebenen Punkt enthalten.

select_xld_point selektiert alle Konturen oder Polygone aus XLDs, die den Testpunkt (Row,Column) enthalten. Es kann vorkommen, dass mehr als eine Kontur oder Polygon den Punkt enthält, falls sich die Konturen oder Polygone überlappen. In diesem Fall werden all diese Konturen oder Polygone ausgegeben. Falls keine Konturen oder Polygone den angegebenen Punkt enthalten, wird ein leeres Objekt ausgegeben. Es wird vorausgesetzt, dass die Konturen oder Polygone geschlossen sind. Ist das nicht der Fall, werden sie künstlich geschlossen. Dennoch sollte darauf geachtet werden, dass nur sinnvolle, geschlossene Konturen und Polygone in XLDs übergeben werden, indem vorab select_contours_xld mit dem Parameter *'closed'* aufgerufen wird.

Achtung

Liegt der Testpunkt direkt auf einer Kontur bzw. einem Polygon, ist das Ergebnis von select_xld_point unvorhersagbar, d.h. die Kontur bzw. das Polygon wird in Abhängigkeit von Faktoren wie der Orientierung der Kontur bzw. des Polygons zu DestXLDs hinzugefügt oder nicht. Weiterhin ist zu beachten, dass bei großer Nähe des Punktes zu den Kontur- bzw. Polygonseiten eine verlässliche Klassifikation durch numerische Ungenauigkeiten praktisch ausgeschlossen ist.

Parameter

▷ **XLDs** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen oder Polygone.

▷ **DestXLDs** (output_object) ..... xld(-array) ⇝ *object*
Alle Konturen oder Polygone, die den Testpunkt enthalten.

▷ **Row** (input_control) ..... point.y ⇝ *real* / integer
Zeilenkoordinate des Testpunktes.
Defaultwert: 100.0

▷ **Column** (input_control) ..... point.x ⇝ *real* / integer
Spaltenkoordinate des Testpunktes.
Defaultwert: 100.0

Ergebnis

select_xld_point liefert den Wert TRUE, falls die Parameter korrekt sind. Das Verhalten bei

leerer Eingabe (keine Eingabekonturen oder -polygone vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

select_contours_xld, close_contours_xld, threshold_sub_pix

Alternativen

test_xld_point

Modul

Foundation

smallest_circle_xld (XLD : : : Row, Column, Radius)

Kleinster umschließender Kreis von Konturen oder Polygonen.

smallest_circle_xld bestimmt den kleinsten umschließenden Kreis von Konturen oder Polygonen, also den Kreis mit dem kleinsten Flächeninhalt unter allen Kreisen, die die Kontur enthalten. Für diesen Kreis werden der Mittelpunkt (Row,Column) und der Radius (Radius) berechnet.

Werden mehrere Konturen oder Polygone in XLD übergeben, so werden entsprechende Tupel als Ausgabeparameter zurückgegeben. Bei leerer Kontur haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu untersuchende Konturen oder Polygone.

▷ **Row** (output_control) ..... circle.center.y(-array) ⇝ *real*
Zeilenkoordinate des Mittelpunktes des umschließenden Kreises.

▷ **Column** (output_control) ..... circle.center.x(-array) ⇝ *real*
Spaltenkoordinate des Mittelpunktes des umschließenden Kreises.

▷ **Radius** (output_control) ..... circle.radius(-array) ⇝ *real*
Radius des umschließenden Kreises.
Zusicherung: Radius >= 0

Komplexität

Seien N die Anzahl der Punkte in der Kontur dann beträgt die Laufzeitkomplexität $O(N * ln(N))$.

Ergebnis

smallest_circle_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld

Nachfolger

gen_ellipse_contour_xld

Alternativen

smallest_rectangle1_xld, smallest_rectangle2_xld

Siehe auch

elliptic_axis_xld, smallest_rectangle1_xld, smallest_rectangle2_xld, shape_trans_xld

Modul

Foundation

smallest_rectangle1_xld (XLD : : : Row1, Column1, Row2, Column2)

Umschließendes Rechteck parallel zu den Koordinatenachsen von Konturen oder Polygonen.

smallest_rectangle1_xld berechnet das umschließende Rechteck (parallel zu den Koordinatenachsen) für jede Eingabekontur bzw. jedes Eingabepolygon. Das umschließende Rechteck wird durch die Koordinaten der Eckpunkte (Row1,Column1,Row2,Column2) beschrieben.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert. Bei leerer Kontur haben alle Parameter den Wert 0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Achtung

Bei leeren Konturen kann das Ergebnis von Row1,Column1, Row2 und Column2 (alle sind 0) zu Konfusionen führen.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen oder Polygone.
- ▷ **Row1** (output_control) ..... rectangle.origin.y(-array) ⇝ *real*
  Zeilenkoordinate des linken oberen Eckpunkts des umschließenden Rechtecks.
- ▷ **Column1** (output_control) ..... rectangle.origin.x(-array) ⇝ *real*
  Spaltenkoordinate des linken oberen Eckpunkts des umschließenden Rechtecks.
- ▷ **Row2** (output_control) ..... rectangle.corner.y(-array) ⇝ *real*
  Zeilenkoordinate des rechten unteren Eckpunkts des umschließenden Rechtecks.
- ▷ **Column2** (output_control) ..... rectangle.corner.x(-array) ⇝ *real*
  Spaltenkoordinate des rechten unteren Eckpunkts des umschließenden Rechtecks.

Komplexität

Sei N die Anzahl der Punkte in der Kontur, dann beträgt die Laufzeitkomplexität $O(N)$.

Ergebnis

smallest_rectangle1_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld

Nachfolger

gen_polygons_xld

Alternativen

height_width_ratio_xld, smallest_rectangle2_xld, shape_trans_xld

Siehe auch

shape_trans_xld, smallest_rectangle2_xld, smallest_circle_xld, elliptic_axis_xld, area_center_xld

Modul

Foundation

smallest_rectangle2_xld (XLD : : : Row, Column, Phi, Length1, Length2)

Kleinstes umschließendes Rechteck mit beliebiger Orientierung von Konturen oder Polygonen.

smallest_rectangle2_xld bestimmt das kleinste umschließende Rechteck für jede Eingabekontur bzw. jedes Eingabepolygon, also das Rechteck mit dem kleinsten Flächeninhalt unter allen Rechtecken, die die Kontur enthalten. Für dieses Rechteck werden der Mittelpunkt, der Neigungswinkel und die beiden Halbmesser berechnet.

Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln in der Reihenfolge ihrer entsprechenden Konturen bzw. Polygone in XLD abgespeichert. Bei leerer Kontur haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu untersuchende Konturen oder Polygone.
- ▷ **Row** (output_control) ..... rectangle2.center.y(-array) ⇝ *real*
  Zeilenkoordinate des Mittelpunktes des umschließenden Rechtecks.
- ▷ **Column** (output_control) ..... rectangle2.center.x(-array) ⇝ *real*
  Spaltenkoordinate des Mittelpunktes des umschließenden Rechtecks.
- ▷ **Phi** (output_control) ..... rectangle2.angle.rad(-array) ⇝ *real*
  Orientierung des umschließenden Rechtecks (Bogenmaß)
  Zusicherung: - pi / 2 < Phi && Phi <= pi / 2
- ▷ **Length1** (output_control) ..... rectangle2.hwidth(-array) ⇝ *real*
  Erster Halbmesser (halbe Länge) des umschließenden Rechtecks.
  Zusicherung: Length1 >= 0.0
- ▷ **Length2** (output_control) ..... rectangle2.hheight(-array) ⇝ *real*
  Zweiter Halbmesser (halbe Breite) des umschließenden Rechtecks.
  Zusicherung: Length2 >= 0.0 && Length2 <= Length1

Komplexität

Seien N die Anzahl der Punkte in der Kontur und C die Anzahl der Punkte der entsprechenden konvexen Hülle, dann beträgt die Laufzeitkomplexität $O(N * ln(N) + C^2)$.

Ergebnis

smallest_rectangle2_xld liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld

Nachfolger

gen_polygons_xld

Alternativen

smallest_rectangle1, shape_trans_xld

Siehe auch

smallest_rectangle1, smallest_circle, elliptic_axis_xld

Modul

Foundation

test_closed_xld (XLD : : : IsClosed)

Prüfen, ob Konturen oder Polygone geschlossen sind.

test_closed_xld prüft für alle Konturen und Polygone aus XLD ob sie geschlossen sind, und liefert das Ergebnis in IsClosed zurück. Ist eine Kontur oder ein Polygon geschlossen, so ist der entsprechende Rückgabewert 1, sonst 0. Eine Kontur oder ein Polygon ist genau dann geschlossen, wenn der letzte Punkt mit dem ersten übereinstimmt.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Zu testende Konturen und Polygone.
- ▷ **IsClosed** (output_control) ..... integer(-array) ⇝ *integer*
  Tupel mit booleschen Werten.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contour_region_xld, edges_sub_pix

Modul

Foundation

test_self_intersection_xld (XLD : : CloseXLD : DoesIntersect)

Test auf Selbstüberschneidungen von XLD-Konturen oder -Polygonen.

test_self_intersection_xld überprüft, ob sich die Eingabekonturen oder -polygone in XLD selbst schneiden. Das Ausgabearray DoesIntersect enthält eine *1*, falls die zugehörige Kontur oder das zugehörige Polygon die eine Selbstüberschneidung besitzt und eine *0* sonst.

Hat der Eingabeparameter CloseXLD den Wert *'true'*, so werden offene Konturen oder Polygone zunächst geschlossen, indem der erste und der letzte Kontur- oder Polygonpunkt durch ein Liniensegment verbunden werden.

Wenn sich eine Kontur oder ein Polygon lediglich selbst berührt, so wird dies nicht als Selbstüberschneidung gewertet.

Achtung

Eine Kontur oder ein Polygon berührt sich dann selbst, wenn ein Kontur- oder Polygonpunkt exakt auf einem Kontur- oder Polygonstück liegt. Ob ein Punkt aber genau auf einem Kontur- oder Polygonstück liegt oder nicht, lässt sich aus numerischen Gründen praktisch nicht entscheiden.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
  Eingabekonturen bzw. -polygone.
- ▷ **CloseXLD** (input_control) ..... string ⇝ *string*
  Sollen die Eingabekonturen oder -polygone zuerst geschlossen werden?
  Defaultwert: 'true'
  Werteliste: CloseXLD ∈ {'true', 'false'}

▷ **DoesIntersect** (output_control) ..... number(-array) ⇝ *integer*
1 für Konturen oder Polygone mit Selbstüberschneidung und 0 sonst.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Modul

Foundation

test_xld_point (XLD : : Row, Column : IsInside)

Test, ob ein(e) oder mehrere Konturen oder Polygone einen oder mehrere gegebene Punkte umschließen.

`test_xld_point` prüft, ob ein Testpunkt (`Row`, `Column`) von einer Kontur oder einem Polygon `XLD` eingeschlossen wird und liefert das Ergebnis in `IsInside` zurück. Ist der Testpunkt von der Kontur bzw. dem Polygon eingeschlossen, so ist der Rückgabewert *1*, sonst *0*. Werden mehr als ein Testpunkt und/oder mehr als eine Kontur/Polygon übergeben, dann werden die Ergebnisse in Tupeln abgespeichert. Im Falle von einem Testpunkt und mehreren Konturen/Polygonen zeigt das (i-1)-te Element des Ergebnistupels `IsInside`, ob der Punkt von der i-ten Kontur oder dem i-ten Polygon eingeschlossen ist. Im Falle von mehreren Testpunkten und einer Kontur/Polygon zeigt das i-te Element des Ergebnistupels `IsInside`, ob der i-te Punkt von der Kontur/Polygon eingeschlossen ist. Im Falle von mehreren Testpunkten und mehreren Konturen/Polygonen zeigt das (i-1)-te Element des Ergebnistupels `IsInside`, ob der (i-1)-te Punkt von der i-ten Kontur oder dem i-ten Polygon eingeschlossen ist (Die Indizes unterscheiden sich, da die Indizierung des Ergebnistupels im Gegensatz zur Indizierung von Objekttupeln bei 0 beginnt). Die Anzahl der Testpunkte muss dabei gleich der Anzahl der Konturen/Polygone sein. Es wird vorausgesetzt, dass die Konturen oder Polygone geschlossen sind. Ist das nicht der Fall, werden sie künstlich geschlossen. Dennoch sollte darauf geachtet werden, dass nur sinnvolle, geschlossene Konturen und Polygone in `XLD` übergeben werden.

Achtung

Liegt ein Testpunkt direkt auf der Kontur bzw. dem Polygon, ist das Ergebnis von `test_xld_point` unvorhersagbar, d.h. *0* oder *1* wird in Abhängigkeit von Faktoren wie der Orientierung der Kontur bzw. des Polygons zurückgegeben. Weiterhin ist zu beachten, dass bei großer Nähe des Punktes zu den Kontur- bzw. Polygonseiten eine verlässliche Klassifikation durch numerische Ungenauigkeiten praktisch ausgeschlossen ist.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu testende Konturen bzw. Polygone.

▷ **Row** (input_control) ..... point.y(-array) ⇝ *real*
Zeilenkoordinaten der zu testenden Punkte.

▷ **Column** (input_control) ..... point.x(-array) ⇝ *real*
Spaltenkoordinaten der zu testenden Punkte.
Parameteranzahl: Column == Row

▷ **IsInside** (output_control) ..... integer(-array) ⇝ *integer*
Tupel mit booleschen Werten.

Ergebnis

`test_xld_point` liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe lässt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

select_contours_xld, close_contours_xld, threshold_sub_pix

Modul

Foundation

29.5 Transformationen

add_noise_white_contour_xld (
Contours : NoisyContours : NumRegrPoints, Amp :)

Verrauschen von XLD-Konturen.

add_noise_white_contour_xld erzeugt aus den Konturen, die in Contours übergeben werden, neue Konturen NoisyContours mit verrauschten Konturpunkten. Dazu wird für jeden Konturpunkt der Eingabekonturen die lokale Regressionsgeraden aus den NumRegrPoints benachbarten Punkten bestimmt. Dann wird der Punkt senkrecht zur Regressionsgeraden verschoben. Der Betrag dieser Verschiebung entspricht gleichverteiltem, mittelwertfreiem, weißem Rauschen mit maximaler Amplitude Amp, das mittels der C-Funktion „drand48“ mit zeitabhängigem Seed generiert wird.

Parameter

▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
Zu verrauschende Konturen.

▷ **NoisyContours** (output_object) ..... xld_cont(-array) ⇝ *object*
Verrauschte Konturen.

▷ **NumRegrPoints** (input_control) ..... integer ⇝ *integer*
Anzahl der für die Regressionsgerade verwendeten Punkte.
Defaultwert: 5
Wertevorschläge: NumRegrPoints ∈ {3, 5, 7, 9}
Restriktion: NumRegrPoints >= 3 && odd(NumRegrPoints)

▷ **Amp** (input_control) ..... real ⇝ *real*
Maximale Amplitude des additiven Rauschens (in [-Amp,Amp] gleichverteilt).
Defaultwert: 1.0
Wertevorschläge: Amp ∈ {0.25, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 10.0}
Restriktion: Amp > 0

Beispiel

```
draw_ellipse(WindowHandle,Row,Column,Phi,Radius1,Radius2)
gen_ellipse_contour_xld(Ellipse,Row,Column,Phi,Radius1,Radius2,0,6.28319, \
                        'positive', 1.5)
add_noise_white_contour_xld(Ellipse,NoisyEllipse,5,0.5)
fit_ellipse_contour_xld (NoisyEllipse, 'fitzgibbon', -1, 2, 0, 200, 3, 2.0, \
                         ERow, EColumn, EPhi, ERadius1, ERadius2, \
                         EStartPhi, EEndPhi, EPointOrder)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert add_noise_white_contour_xld den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix, gen_ellipse_contour_xld

Nachfolger

smooth_contours_xld

Siehe auch

smooth_contours_xld, add_noise_white

Modul

Foundation

clip_contours_xld (Contours : ClippedContours : Row1, Column1, Row2, Column2 :)

Beschneiden von XLD-Konturen.

clip_contours_xld beschneidet alle Konturen, die in Contours übergeben werden, d.h., es werden nur solche Konturpunkte eingetragen, die innerhalb des Rechtecks, das durch Row1, Column1, Row2 und Column2 gegeben ist, liegen. Dabei ist zu beachten, dass sich das Rechteck wie eine Region verhält, d.h., das Rechteck umschließt die enthaltenen Pixel vollständig. Gegebenenfalls werden Konturen aufgebrochen und als mehrere Konturen eingetragen. Die resultierenden Konturen werden in ClippedContours ausgegeben.

Parameter

▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
Zu beschneidende Konturen.

▷ **ClippedContours** (output_object) ..... xld_cont(-array) ⇝ *object*
Beschnittene Konturen.

▷ **Row1** (input_control) ..... rectangle.origin.y ⇝ *integer*
Zeile der linken oberen Ecke des Clip-Rechtecks.
Defaultwert: 0
Wertevorschläge: Row1 ∈ {0, 500, 1000, 1500, 2000}

▷ **Column1** (input_control) ..... rectangle.origin.x ⇝ *integer*
Spalte der linken oberen Ecke des Clip-Rechtecks.
Defaultwert: 0
Wertevorschläge: Column1 ∈ {0, 500, 1000, 1500, 2000}

▷ **Row2** (input_control) ..... rectangle.corner.y ⇝ *integer*
Zeile der rechten unteren Ecke des Clip-Rechtecks.
Defaultwert: 512
Wertevorschläge: Row2 ∈ {512, 1024, 1536, 2048}

▷ **Column2** (input_control) ..... rectangle.corner.x ⇝ *integer*
Spalte der rechten unteren Ecke des Clip-Rechtecks.
Defaultwert: 512
Wertevorschläge: Column2 ∈ {512, 1024, 1536, 2048}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Nachfolger

gen_polygons_xld

Alternativen

crop_contours_xld

Siehe auch

clip_region, crop_part, crop_contours_xld

Modul

Foundation

clip_end_points_contours_xld (Contours : ClippedContours : Mode, Length :)

Abschneiden der Endpunkte einer XLD-Kontur.

clip_end_points_contours_xld schneidet von einer XLD-Kontur die Endpunkte ab. Dabei kann über den Parameter Mode festgelegt werden, mit welchem Maß abgeschnitten wird.

Falls Mode = *'length'*, wird in Length die euklidische Länge des abzuschneidenden Konturstückes angegeben. Falls Mode = *'num_points'*, wird in Length die Anzahl an abzuschneidenden Punkte übergeben.

Die beschnittene Kontur wird in ClippedContours zurückgegeben. Für den Fall, dass alle Punkte einer Kontur entfernt würden, wird keine Kontur zurückgegeben. Deshalb ist die Anzahl an Ausgabekonturen kleiner oder gleich der Anzahl an Eingabekonturen.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabekontur.
- ▷ **ClippedContours** (output_object) ..... xld_cont(-array) ⇝ *object*
  Verkürzte Kontur
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Clipping-Modus.
  Defaultwert: 'num_points'
  Werteliste: Mode ∈ {'num_points', 'length'}
- ▷ **Length** (input_control) ..... number ⇝ *real* / integer
  Clipping-Länge in der Einheit Pixel (Mode = 'length') oder Zahl (Mode = 'num_points')
  Defaultwert: 3
  Wertevorschläge: Length ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Ergebnis

Sind die Parameterwerte korrekt, liefert clip_end_points_contours_xld den Wert TRUE. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

segment_contours_xld

Siehe auch

clip_contours_xld, crop_contours_xld

Modul

Foundation

close_contours_xld (Contours : ClosedContours : :)

Schließen von XLD-Konturen.

close_contours_xld schließt alle Konturen, die in Contours übergeben werden. Die resultierenden Konturen werden in ClosedContours ausgegeben. Um sinnvolle Ergebnisse zu erhalten, sollte man zunächst mit select_contours_xld diejenigen Konturen auswählen, deren Endpunkte nicht weiter als ein bestimmter Maximalabstand auseinander liegen.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *object*
  Zu schließende Konturen.
- ▷ **ClosedContours** (output_object) ..... xld_cont(-array) ⇝ *object*
  Geschlossene Konturen.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

select_contours_xld

Nachfolger

area_center_xld, circularity_xld, compactness_xld, convexity_xld, eccentricity_xld, elliptic_axis_xld, moments_xld, moments_any_xld, orientation_xld, select_shape_xld, test_xld_point

Modul

Foundation

combine_roads_xld (EdgePolygons, ModParallels, ExtParallels, CenterLines : RoadSides : MaxAngleParallel, MaxAngleColinear, MaxDistanceParallel, MaxDistanceColinear :)

Kombinieren von Straßenhypothesen aus zwei Auflösungsstufen.

Mit combine_roads_xld werden die Straßenhypothesen aus zwei Auflösungsstufen kombiniert. Es werden nur Straßenhypothesen selektiert, die in beiden Auflösungsstufen unterstützende Hypothesen besitzen. Dabei stellen die Eingabeobjekte EdgePolygons, ModParallels und ExtParallels die Ergebnisse einer Straßenextraktion auf der höchsten Auflösungsstufe dar. Der Parameter CenterLines ist das Ergebnis der Straßenextraktion auf einer niedrigen Auflösungsstufe. Es werden von den Polygonen EdgePolygons diejenigen in RoadSides zurückgegeben, für die Evidenz in beiden Stufen gefunden wurde. Die Parameter MaxAngleParallel und MaxAngleColinear geben die Winkel an, die zwei parallele bzw. kollineare Linien einschließen dürfen. Die Parameter MaxDistanceParallel und MaxDistanceColinear geben den Maximalabstand von parallelen bzw. kollinearen Linien an. Die Kombination erfolgt intern durch eine Anzahl von Regeln.

Parameter

- ▷ **EdgePolygons** (input_object) ..... xld_poly-array ⇝ *object*
  Polygone, die untersucht werden sollen.
- ▷ **ModParallels** (input_object) ..... xld_mod_para-array ⇝ *object*
  Modifizierte Parallelen für EdgePolygons.
- ▷ **ExtParallels** (input_object) ..... xld_ext_para-array ⇝ *object*
  Erweiterte Parallelen für EdgePolygons.
- ▷ **CenterLines** (input_object) ..... xld_poly-array ⇝ *object*
  Mittelachsen-Polygone, die untersucht werden sollen.
- ▷ **RoadSides** (output_object) ..... xld_poly-array ⇝ *object*
  Gefundene Straßenränder (als Polygone).

▷ **MaxAngleParallel** (input_control) . . . . . angle.rad ⇝ *real* / integer
Maximaler Winkel zwischen zwei parallelen Linien.
Defaultwert: 0.523598775598
Wertevorschläge: MaxAngleParallel ∈ {0.349065850399, 0.523598775598, 0.6981317008}
Restriktion: 0 <= MaxAngleParallel <= pi / 2

▷ **MaxAngleColinear** (input_control) . . . . . angle.rad ⇝ *real* / integer
Maximaler Winkel zwischen zwei kollinearen Linien.
Defaultwert: 0.261799387799
Wertevorschläge: MaxAngleColinear ∈ {0.174532925199, 0.261799387799, 0.349065850399}
Restriktion: 0 <= MaxAngleColinear <= pi / 2

▷ **MaxDistanceParallel** (input_control) . . . . . real ⇝ *real* / integer
Maximaler Abstand zwischen zwei parallelen Linien.
Defaultwert: 40
Wertevorschläge: MaxDistanceParallel ∈ {20, 30, 40, 50, 60}
Restriktion: MaxDistanceParallel > 0

▷ **MaxDistanceColinear** (input_control) . . . . . real ⇝ *real* / integer
Maximaler Abstand zwischen zwei kollinearen Linien.
Defaultwert: 40
Wertevorschläge: MaxDistanceColinear ∈ {20, 30, 40, 50, 60}
Restriktion: MaxDistanceColinear > 0

Ausführungsinformationen

- Multithreading-Typ: mutually exclusive (läuft parallel zu anderen nicht-exklusiven Operatoren mit Ausnahme von sich selbst).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

mod_parallels_xld, gen_polygons_xld, affine_trans_contour_xld

Nachfolger

get_polygon_xld, get_lines_xld

Siehe auch

lines_gauss, lines_facet, get_channel_info, edges_sub_pix

Literatur

C. Steger, C. Glock, W. Eckstein, H. Mayer, B. Radig; „Model-based Road Extraction from Images“; in „Automatic Extraction of Man-Made Objects from Aerial and Space Images“; A. Gruen, O. Kuebler, P. Agouris (Editors); Birkhäuser Verlag (1995), pp. 275-284.

C. Heipke, C. Steger, R. Multhammer; „A Hierarchical Approach to Automatic Road Extraction from Aerial Imagery“; in „Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision II“; D. M. McKeown, Jr., I. J. Dowman (Editors); Proc. SPIE 2486 (1995), pp. 222-231.

Modul

Foundation

crop_contours_xld (Contours : CroppedContours : Row1, Col1, Row2, Col2, CloseContours :)

Beschneiden von XLD-Konturen.

crop_contours_xld beschneidet die XLD-Konturen die in Contours übergeben werden. Es werden also nur Kontur-Segmente zurückgeliefert, die sich innerhalb des Rechtecks befinden, das durch die Eckpunkte Row1, Col1, Row2 und Col2 gegeben ist. Während clip_contours_xld alle Segmente weglässt, die den Rand des einschränkenden Rechtecks kreuzen werden diese hier genau an der Stelle abgeschnitten, an der sie eine Grenzlinie überschreiten. Falls notwendig werden aus einer Ausgangskontur mehrere neue Konturen erzeugt. Aus geschlossenen Konturen können mit Hilfe des Parameters CloseContours wieder geschlossene Konturen erzeugt werden. Wird CloseContours auf *'true'* gesetzt, so wird eine geschlossene Kontur in eine oder mehrere neue Konturen

überführt, die über neue Segmente auf dem Rand des einschränkenden Rechtecks geschlossen sind. Ist der Wert *'false'*, so endet jede Kontur an dem Punkt an dem sie den Bereich des Rechtecks verlässt. Die beschnittenen Konturen werden in `CroppedContours` zurückgegeben.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *object*
  Eingabekonturen.
- ▷ **`CroppedContours`** (output_object) ..... xld_cont(-array) ⇝ *object*
  Ergebniskonturen.
- ▷ **`Row1`** (input_control) ..... number ⇝ *real* / integer
  Oberkante des beschränkenden Rechtecks.
  Defaultwert: 0
  Wertevorschläge: `Row1` ∈ {0, 500, 1000, 1500, 2000}
- ▷ **`Col1`** (input_control) ..... number ⇝ *real* / integer
  Linke Kante des beschränkenden Rechtecks.
  Defaultwert: 0
  Wertevorschläge: `Col1` ∈ {0, 500, 1000, 1500, 2000}
- ▷ **`Row2`** (input_control) ..... number ⇝ *real* / integer
  Unterkante des beschränkenden Rechtecks.
  Defaultwert: 512
  Wertevorschläge: `Row2` ∈ {512, 1024, 1536, 2048}
- ▷ **`Col2`** (input_control) ..... number ⇝ *real* / integer
  Rechte Kante des beschränkenden Rechtecks.
  Defaultwert: 512
  Wertevorschläge: `Col2` ∈ {512, 1024, 1536, 2048}
- ▷ **`CloseContours`** (input_control) ..... string ⇝ *string*
  Sollen geschlossene Konturen wieder geschlossene Konturen erzeugen?
  Defaultwert: 'true'
  Werteliste: `CloseContours` ∈ {'true', 'false'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Alternativen

`clip_contours_xld`

Siehe auch

`clip_region`, `crop_part`, `clip_contours_xld`

Modul

Foundation

`merge_cont_line_scan_xld` `( CurrConts, PrevConts : CurrMergedConts, PrevMergedConts : ImageHeight, Margin, MergeBorder, MaxImagesCont : )`

Zusammenfügen von Konturen aus aufeinanderfolgenden Zeilenkamerabildern.

Der Operator `merge_cont_line_scan_xld` verbindet Konturen, die aus räumlich aneinandergrenzenden Bildern der Höhe `ImageHeight` extrahiert wurden und über die Bildränder hinaus ineinander überführt werden können. Die Bilder können beispielsweise von einer Zeilenkamera nacheinander aufgenommen worden sein. Dabei wird davon ausgegangen, dass `CurrConts` die Konturen des aktuellen Bildes enthält und `PrevConts` die des vorherigen, welches die räumliche Fortsetzung des aktuellen Bildes darstellt.

Mit Hilfe des Parameters `MergeBorder` kann angegeben werden, ob die Oberkante des aktuellen Bildes an die Unterkante des vorherigen Bildes stößt (*'top'*) oder die Unterkante des aktuellen Bildes an die Oberkante des

vorherigen Bildes (*'bottom'*). MergeBorder definiert den Abstand vom Rand, in dem die Endpunkte der Kontur liegen müssen, um für das Merging berücksichtigt zu werden.

Der Parameter MaxImagesCont bestimmt bei der rekursiven Anwendung des Operators merge_cont_line_scan_xld, wie viel Bilder eine aktuelle Ausgangskontur maximal zurückreichen kann. Alle Konturpunkte aus früheren Bildern werden aus der Ausgabekontur entfernt.

Der Operator merge_cont_line_scan_xld liefert zwei Konturarrays zurück. PrevMergedConts enthält alle Konturen, die ausschließlich in der alten Konturenliste enthalten sind und nicht mit den aktuellen Konturen verbunden werden konnten. In CurrMergedConts werden dagegen alle aktuellen und die mit diesen verbundenen Konturen eingetragen. Die miteinander verbundenen Konturen erscheinen als eine einzige neue Kontur in CurrMergedConts, wobei diese verbundenen Konturen über den Bildrand hinausgehen. Das bedeutet, dass der an eine aktuelle Kontur angebundene alte Teil um die Bildhöhe nach oben (MergeBorder=*'top'*) oder nach unten (MergeBorder=*'bottom'*) verschoben wird.

Parameter

- ▷ **CurrConts** (input_object) ..... xld_cont(-array) ⇝ *object*
  Aktuelle Eingabekonturen.
- ▷ **PrevConts** (input_object) ..... xld_cont(-array) ⇝ *object*
  Im vorhergehenden Zyklus zusammengefügte Konturen.
- ▷ **CurrMergedConts** (output_object) ..... xld_cont(-array) ⇝ *object*
  Aktuelle Konturen, die ggf. mit den alten Konturen verbunden wurden.
- ▷ **PrevMergedConts** (output_object) ..... xld_cont(-array) ⇝ *object*
  Alte Konturen, die nicht mit den aktuellen verbunden werden konnten.
- ▷ **ImageHeight** (input_control) ..... integer ⇝ *integer*
  Höhe der Ausgangsbilder.
  Defaultwert: 512
  Wertevorschläge: ImageHeight ∈ {240, 480, 512}
- ▷ **Margin** (input_control) ..... real ⇝ *real* / integer
  Maximaler Abstand der Konturen vom Rand.
  Defaultwert: 0.0
  Wertevorschläge: Margin ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0}
- ▷ **MergeBorder** (input_control) ..... string ⇝ *string*
  Im aktuellen Bild die Zeile, die mit dem vorhergehenden Bild zusammenstößt.
  Defaultwert: 'top'
  Werteliste: MergeBorder ∈ {'top', 'bottom'}
- ▷ **MaxImagesCont** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl der Bilder, über die sich eine Kontur erstrecken darf.
  Defaultwert: 3
  Wertevorschläge: MaxImagesCont ∈ {1, 2, 3, 4, 5}

Ergebnis

merge_cont_line_scan_xld liefert den Wert TRUE, falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Modul

Foundation

regress_contours_xld (Contours : RegressContours : Mode, Iterations :)

Berechnen von Regressionsgeradenparametern für Konturen

Mit `regress_contours_xld` werden für die Konturen `Contours` folgende Parameter berechnet und in der Kontur als globale Attribute abgelegt:

- die Koordinaten des Normalenvektors der optimalen Regressionsgeraden durch alle Konturpunkte einer Kontur; dieser Normalenvektor verläuft vom Ursprung zur Regressionsgeraden (Attribute: *'regr_norm_row'*, *'regr_norm_col'*)
- der mittlere Abstand der Konturpunkte zu dieser Regressionsgeraden (Attribute: *'regr_mean_dist'*)
- die Standardabweichung des Abstandes der Konturpunkte zu dieser Regressionsgeraden (Attribute: *'regr_dev_dist'*).

Für `Mode` = *'no'* werden die Parameter für die optimale Regressionsgerade durch alle Konturpunkte berechnet. Bei der Berechnung der Regressionsgeraden können aber auch drei verschiedene Ausreißerbehandlungen vorgenommen werden. Ausreißer sind Konturpunkte, die „offensichtlich“ nicht in der globalen Konturrichtung liegen und deswegen die Regressionsgeradenberechnung „verfälschen“.

`Mode` =

- *'drop'*: Alle Konturpunkte, die weiter als der durchschnittliche Abstand der Konturpunkte zur optimalen Regressionsgerade von dieser entfernt sind, werden bei der Berechnung der bereinigten Regressionsgerade unterschlagen.
- *'gauss'*: Die Konturpunktabstände werden bei der Regressionsgeradenberechnung mit ihrer Eintrittswahrscheinlichkeit bei Normalverteilung um die optimale Regressionsgerade gewichtet.
- *'median'*: Es wird ebenso eine Normalverteilung der Konturpunktabstände zur optimalen Regressionsgeraden zugrunde gelegt, allerdings mit der ausreißerunabhängigen Standardabweichung $\frac{M_d}{0.6745}$, mit M_d: Median aller Abstände. Die Abstände werden wieder gewichtet und Punkte ab einem gewissen Abstand unterschlagen.

Die Berechnung der bereinigten Regressionsgeraden kann mehrfach iteriert werden (`Iterations`).

Parameter

▷ **`Contours`** (input_object) ..... xld_cont-array ⇝ *object*
Eingabe-Konturen.

▷ **`RegressContours`** (output_object) ..... xld_cont-array ⇝ *object*
Ausgabe-Konturen.

▷ **`Mode`** (input_control) ..... string ⇝ *string*
Ausreißerbehandlung.
Defaultwert: 'no'
Werteliste: `Mode` ∈ {'no', 'drop', 'gauss', 'median'}

▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
Anzahl Iterationen der Ausreißerbehandlung.
Defaultwert: 1
Wertevorschläge: `Iterations` ∈ {1, 2, 3, 5, 10, 20}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Nachfolger

`get_regress_params_xld`

Siehe auch

`smooth_contours_xld`, `get_contour_global_attrib_xld`, `query_contour_global_attribs_xld`

Literatur

H. Suesse, K. Voss: „Adaptive Ausgleichsrechnung und Ausreißerproblematik für die digitale Bildverarbeitung“; Proc. 15. DAGM Symposium, Springer Verlag, Lübeck 1993

R. Haralick, L. Shapiro: „Computer and Robot Vision“ Vol. 2; Kapitel 14.9, Addison-Wesley 1992

Modul

Foundation

segment_contour_attrib_xld (Contour : ContourPart : Attribute, Operation, Min, Max :)

Segmentieren von Konturteilen, deren Attribute vorgegebene Bedingungen erfüllen.

segment_contour_attrib_xld segmentiert die Eingabekontur Contour in Kontursegmente ContourPart. Die Segmentierung der Kontur ist von den Werten der Konturattribute Attribute abhängig (siehe unten für mögliche Werte von Attribute). Der Operator prüft welche Attributwerte innerhalb der vorgegebenen Grenzen (Min, Max) liegen und liefert die entsprechenden Kontursegmente zusammen mit ihren Attributen in ContourPart zurück.

Ist Operation auf *'and'* gesetzt, muss jedes der angegebenen Attribute innerhalb der vorgegebenen Grenzen liegen. Ist Operation auf *'or'* gesetzt, muss mindestens ein Attribut innerhalb der vorgegebenen Grenzen liegen. Um diese Grenzen nach unten oder oben offen zu lassen, kann statt eines Wertes auch *'min'* beziehungsweise *'max'* für Min und Max gesetzt werden. Die Attribute werden in der Reihenfolge abgearbeitet, in der sie in Attribute aufgeführt sind. Wird nur ein Merkmal (Attribute) verwendet, wird der Wert von Operation ignoriert.

Es ist zu beachten, dass nicht alle Attribute in jeder Kontur vorhanden sind. Welche Attribute für eine gegebene Kontur definiert sind, hängt davon ab, wie die Kontur erzeugt wurde und ob bestimmte Operationen darauf ausgeführt wurden. So sind zum Beispiel die Attribute *'width_right'* und *'width_left'* nur vorhanden, wenn die Kontur mit dem Operator lines_gauss erzeugt wurde. Ebenso ist das Attribut *'distance'* erst vorhanden, wenn zuvor der Abstand zu einer anderen Kontur mit distance_contours_xld oder apply_distance_transform_xld berechnet wurde.

Bedingung: $Min_i \leq Feature_i(Object) \leq Max_i$

Mögliche Werte für Attribute:

'edge_direction': Kantenrichtung

'angle': Der Winkel der Richtung senkrecht zur Linie

'response': Die Größe der zweiten Ableitung

'width_right': Die Linienbreite rechts von der Linie

'width_left': Die Linienbreite links von der Linie

'contrast': Der Kontrast des Linienpunkts

'asymmetry': Die Asymmetrie des Linienpunkts

'distance': Der minimale punktweise Abstand zu einer Referenzkontur

Welche Operatoren einer Kontur die oben genannten Attribute hinzufügen und weitere Informationen zu den Attributen finden sich in get_contour_attrib_xld.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont(-array) ⇝ *object*
  Zu segmentierende Kontur.
- ▷ **ContourPart** (output_object) ..... xld_cont-array ⇝ *object*
  Kontursegmente.
- ▷ **Attribute** (input_control) ..... string(-array) ⇝ *string*
  Konturattribute, die geprüft werden.
  Defaultwert: 'distance'
  Werteliste: Attribute ∈ {'distance', 'angle', 'edge_direction', 'width_right', 'width_left', 'response', 'contrast', 'asymmetry'}
- ▷ **Operation** (input_control) ..... string ⇝ *string*
  Art der Verlinkung der individuellen Attribute.
  Defaultwert: 'and'
  Werteliste: Operation ∈ {'and', 'or'}

▷ **Min** (input_control) . . . . . real(-array) ⇝ *real* / integer / string
Untere Grenze der Features.
Defaultwert: 150.0
Typischer Wertebereich: 0.0 ≤ Min ≤ 99999.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 1.0

▷ **Max** (input_control) . . . . . real(-array) ⇝ *real* / integer / string
Obere Grenze der Features.
Defaultwert: 99999.0
Typischer Wertebereich: 0.0 ≤ Max ≤ 99999.0
Minimale Schrittweite: 0.001
Empfohlene Schrittweite: 1.0
Restriktion: Max >= Min

Beispiel

```
read_image (Image, 'fabrik')
edges_sub_pix (Image, Edges, 'canny', 1, 20, 40)
* select 'vertical' edges:
segment_contour_attrib_xld (Edges, ContourPart, 'edge_direction', \
                            'and', rad(90-20), rad(90+20))
```

Ergebnis

select_shape liefert den Wert TRUE, falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) lässt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

lines_gauss, edges_sub_pix, apply_distance_transform_xld, distance_contours_xld

Nachfolger

fit_line_contour_xld, fit_ellipse_contour_xld, fit_circle_contour_xld, fit_rectangle2_contour_xld

Siehe auch

split_contours_xld, get_contour_global_attrib_xld, query_contour_attribs_xld, get_contour_attrib_xld

Modul

Foundation

segment_contours_xld (Contours : ContoursSplit : Mode, SmoothCont, MaxLineDist1, MaxLineDist2 :)

Segmentieren von Konturen in Liniensegmente und Kreis- oder Ellipsenbögen.

segment_contours_xld segmentiert die Eingabekonturen Contours in Linien, falls Mode=*'lines'*, in Linien und Kreisbögen, falls Mode=*'lines_circles'*, oder in Linien und Ellipsenbögen, falls Mode=*'lines_ellipses'*. Die segmentierten Konturen werden in ContoursSplit zurückgegeben. Die Unterscheidung, ob eine Ausgabekontur ein Liniensegment oder einen Kreis- oder Ellipsenbogen darstellt, erfolgt über das globale Konturattribut *'cont_approx'* (siehe get_contour_global_attrib_xld). Falls *'cont_approx'*=-1, liegt ein Liniensegment vor, falls *'cont_approx'*=0 ein Ellipsenbogen und falls *'cont_approx'*=1 ein Kreisbogen.

Intern führt segment_contours_xld zunächst eine Polygonapproximation der Eingabekonturen durch. Dadurch werden die Konturen in gekrümmten Bereichen übersegmentiert. Hierauf werden iterativ benachbarte Liniensegmente zu Kreis- bzw. Ellipsenbögen verschmolzen, falls dadurch die Kontur besser approximiert werden kann. Wenn in SmoothCont ein Wert > 0 angegeben wird, werden die Konturen zunächst geglättet (siehe smooth_contours_xld). Dies kann erforderlich sein, um sehr kurze Liniensegmente bei der Polygonapproximation zu verhindern und eine stabilere Kreis- bzw. Ellipsenanpassung zu erreichen, da durch die Glättung Ausreißer auf den Konturen unterdrückt werden.

Die initiale Polygonapproximation wird mit dem Algorithmus von Ramer (siehe gen_polygons_xld) mit einem Maximalabstand von MaxLineDist1 durchgeführt. Hierauf werden in benachbarte Liniensegmente Kreis- bzw. Ellipsenbögen angepasst. Wenn der Maximalabstand des so entstehenden Bogens zur Kontur kleiner ist als der Maximalabstand der zwei betrachteten Liniensegmente, werden die zwei Liniensegmente durch diesen Bogen ersetzt. Dieser Vorgang wird solange wiederholt, bis keine Veränderungen mehr auftreten.

Hierauf werden die Teile der Kontur, die noch als Liniensegmente approximiert werden, einer Polygonsegmentierung mit dem Maximalabstand MaxLineDist2 unterzogen, und die neu entstehenden Liniensegmente wiederum zu Kreis- oder Ellipsenbögen zusammengefasst. Dies ändert die Ausgabe natürlich nur, falls MaxLineDist2 $<$ MaxLineDist1. Dieses zweistufige Verfahren ist effizienter, als ein einstufiges Verfahren mit MaxLineDist2, da im ersten Schritt weniger Liniensegmente entstehen, und dadurch die Kreis- bzw. Ellipsenanpassung weniger oft durchgeführt werden muss. Daher können Teile der Eingabekonturen, die sich durch lange Bögen approximieren lassen, effizienter gefunden werden. Im zweiten Schritt werden dann Teile der Kontur gefunden, die sich durch kurze Bögen approximieren lassen und die Endstücke der im ersten Schritt gefundenen Konturen werden verfeinert.

Die Ergebniskonturen sind mindestens 3 Pixel lang und umfassen mindestens 6 fortlaufende Konturpunkte der Eingabekontur. Alle Eingabekonturen mit einer Länge von weniger als 3 Pixel oder mit weniger als 6 Konturpunkten werden unverändert in die Ausgabekonturen übernommen.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *object*
  Konturen, die segmentiert werden sollen.
- ▷ **ContoursSplit** (output_object) ..... xld_cont-array ⇝ *object*
  Zerlegte Konturen.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus für die Segmentierung der Konturen.
  Defaultwert: 'lines_circles'
  Werteliste: Mode ∈ {'lines', 'lines_circles', 'lines_ellipses'}
- ▷ **SmoothCont** (input_control) ..... integer ⇝ *integer*
  Einzugsbereich für die Glättung der Konturen.
  Defaultwert: 5
  Wertevorschläge: SmoothCont ∈ {0, 3, 5, 7, 9}
  Restriktion: SmoothCont == 0 || SmoothCont >= 3 && odd(SmoothCont)
- ▷ **MaxLineDist1** (input_control) ..... real ⇝ *real*
  Maximaler Abstand zwischen einer Kontur und der approximierenden Gerade (erster Durchlauf).
  Defaultwert: 4.0
  Wertevorschläge: MaxLineDist1 ∈ {1.0, 1.5, 2.0, 2.5, 3.0, 3.5}
  Restriktion: MaxLineDist1 >= 0.0
- ▷ **MaxLineDist2** (input_control) ..... real ⇝ *real*
  Maximaler Abstand zwischen einer Kontur und der approximierenden Gerade (zweiter Durchlauf).
  Defaultwert: 2.0
  Wertevorschläge: MaxLineDist2 ∈ {1.0, 1.5, 2.0, 2.5, 3.0, 3.5}
  Restriktion: MaxLineDist2 >= 0.0

Beispiel

```
read_image (Image, 'pumpe')
edges_sub_pix (Image, Edges, 'canny', 1.5, 15, 40)
segment_contours_xld (Edges, ContoursSplit, 'lines_circles', 5, 4, 2)
count_obj (ContoursSplit, Number)
gen_empty_obj (Lines)
gen_empty_obj (Circles)
```

```
for I := 1 to Number by 1
  select_obj (ContoursSplit, Contour, I)
  get_contour_global_attrib_xld (Contour, 'cont_approx', Type)
  if (Type == -1)
      concat_obj (Lines, Contour, Lines)
  else
      concat_obj (Circles, Contour, Circles)
  endif
endfor
fit_line_contour_xld (Lines, 'tukey', -1, 0, 5, 2, RowBegin, ColBegin, \
                      RowEnd, ColEnd, Nr, Nc, Dist)
fit_circle_contour_xld (Circles, 'atukey', -1, 2, 0, 3, 2, Row, Column, \
                        Radius, StartPhi, EndPhi, PointOrder)
```

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, lines_gauss, edges_sub_pix

Nachfolger

fit_line_contour_xld, fit_ellipse_contour_xld, fit_circle_contour_xld, get_contour_global_attrib_xld

Siehe auch

split_contours_xld, get_contour_global_attrib_xld, smooth_contours_xld, gen_polygons_xld

Modul

Foundation

shape_trans_xld (XLD : XLDTrans : Type :)

Transformation der Form von Konturen oder Polygonen.

shape_trans dient zur Transformation von Konturen oder Polygonen in Abhängigkeit von Type:

'convex' Konvexe Hülle.

'ellipse' Ellipse mit den gleichen Momenten und Fläche wie die Eingabekontur. Es ist darauf zu achten, dass sich die geschlossene Eingabekontur bzw. das geschlossene Eingabepolygon nicht selbst schneidet, da die zurückgelieferte Ellipse ansonsten nicht aussagekräftig ist (Ob sich die Eingabekontur bzw. das Eingabepolygon selbst schneidet oder nicht, lässt sich mit test_self_intersection_xld überprüfen).

'outer_circle' Kleinster umschließender Kreis.

'rectangle1' Kleinstes umschließendes Rechteck parallel zu den Koordinatenachsen.

'rectangle2' Kleinstes umschließendes Rechteck.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *object*
Zu transformierende Konturen bzw. Polygone.

▷ **XLDTrans** (output_object) ..... xld(-array) ⇝ *object*
Transformierte Konturen bzw. Polygone.

▷ **Type** (input_control) ..... string ⇝ *string*
Art der Transformation.
Defaultwert: 'convex'
Werteliste: Type ∈ {'convex', 'ellipse', 'outer_circle', 'rectangle1', 'rectangle2'}

Ergebnis

shape_trans_xld liefert den Wert TRUE, falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabekontur vorhanden) lässt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

gen_contours_skeleton_xld, edges_sub_pix, threshold_sub_pix, gen_contour_polygon_xld, test_self_intersection_xld

Alternativen

smallest_circle_xld, smallest_rectangle1_xld, smallest_rectangle2_xld, elliptic_axis_xld

Siehe auch

smallest_circle_xld, smallest_rectangle1_xld, smallest_rectangle2_xld, elliptic_axis_xld

Modul

Foundation

smooth_contours_xld (Contours : SmoothedContours : NumRegrPoints :)

Glätten von XLD-Konturen.

smooth_contours_xld erzeugt aus den Konturen, die in Contours übergeben werden, neue Konturen SmoothedContours, indem die Konturpunkte der Ausgabekontur durch Projektion des entsprechenden Punktes der Eingabekontur auf eine lokale Regressionsgerade berechnet werden. Diese Gerade wird aus den NumRegrPoints benachbarten Punkten des jeweils aktuellen Punktes berechnet. Diese Funktion sollte z.B. aufgerufen werden, bevor Konturen skaliert werden.

Parameter

▷ **Contours** (input_object) ..... xld_cont-array ⇝ *object*
Zu glättende Konturen.

▷ **SmoothedContours** (output_object) ..... xld_cont-array ⇝ *object*
Geglättete Konturen.

▷ **NumRegrPoints** (input_control) ..... integer ⇝ *integer*
Anzahl der für die Regressionsgerade verwendeten Punkte.
Defaultwert: 5
Wertevorschläge: NumRegrPoints ∈ {3, 5, 7, 9}
Restriktion: NumRegrPoints >= 3 && odd(NumRegrPoints)

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Nachfolger

affine_trans_contour_xld, gen_polygons_xld, local_max_contours_xld

Siehe auch

`get_contour_xld`

Modul

Foundation

sort_contours_xld (Contours : SortedContours : SortMode, Order, RowOrCol :)

Sortieren von Konturen aufgrund ihrer relativen Lage.

`sort_contours_xld` ordnet die Konturen bezüglich ihrer relativen Position an. `RowOrCol` spezifiziert dabei, nach welcher Koordinate ihrer Position die Konturen zuerst sortiert werden: *'row'* lässt die Konturen zuerst bzgl. der Zeilenkoordinate der Position und dann bei gleichen Zeilenkoordinaten bzgl. der verbleibenden Spaltenkoordinate sortieren. *'column'* verwendet dagegen zuerst die Spaltenkoordinate. Die Position einer Kontur wird dabei mit Hilfe des Parameters `SortMode` durch einen einzelnen Referenzpunkt festgelegt. Hierzu stehen folgende Parameterwerte zur Verfügung:

'upper_left' : Die Position wird durch die linke, obere Ecke des umschließenden Rechtecks bestimmt.

'upper_right' Die Position wird durch die rechte, obere Ecke des umschließenden Rechtecks bestimmt.

'lower_left' Die Position wird durch die linke, untere Ecke des umschließenden Rechtecks bestimmt.

'lower_right' Die Position wird durch die rechte, untere Ecke des umschließenden Rechtecks bestimmt.

'character' : Die Position wird durch die linke, obere Ecke des umschließenden Rechtecks bestimmt. Abweichend von *'upper_left'* wird auch dann nach der verbleibenden Koordinate sortiert, wenn sich zwei Konturen in der Koordinatenrichtung überlappen, welche durch den Parameter `RowOrCol` definiert ist.

Der Parameter `Order` legt fest, ob auf- oder absteigend sortiert wird. Bei *'true'* wird aufsteigend bei *'false'* absteigend sortiert.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *object*
  Anzuordnende Konturen.
- ▷ **SortedContours** (output_object) ..... xld_cont-array ⇝ *object*
  Angeordnete Konturen.
- ▷ **SortMode** (input_control) ..... string ⇝ *string*
  Art der Sortierung.
  Defaultwert: 'upper_left'
  Werteliste: `SortMode` ∈ {'character', 'upper_left', 'lower_left', 'upper_right', 'lower_right'}
- ▷ **Order** (input_control) ..... string ⇝ *string*
  Aufsteigend oder absteigend sortieren.
  Defaultwert: 'true'
  Werteliste: `Order` ∈ {'true', 'false'}
- ▷ **RowOrCol** (input_control) ..... string ⇝ *string*
  Anordnung erst nach Zeile oder Spalte.
  Defaultwert: 'row'
  Werteliste: `RowOrCol` ∈ {'row', 'column'}

Ergebnis

Sind die Parameter korrekt, dann liefert `sort_contours_xld` den Wert TRUE. Andernfalls wird eine Fehlerbehandlung durchgeführt.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Siehe auch

sort_region

Modul

Foundation

split_contours_xld (Polygons : Contours : Mode, Weight, Smooth :)

Aufspalten von Konturen an regional markanten Punkten.

Mit split_contours_xld werden die den Polygonen Polygons zugrundeliegenden Konturen an regional markanten Punkten aufgespalten. Im Modus *'polygon'* erfolgt die Aufspaltung gemäß den Polygonen selbst. Im Modus *'dominant'* werden Konturpunkte zur Aufspaltung verwendet, für die die in ihnen ermittelte Konturrichtungsänderung den (empirischen) Grenzwert 2πWeight$/$*Konturlänge* überschreitet und in deren (empirischen) Umgebung von $\sqrt{\text{Smooth}} * \log(\textit{Konturlänge})$ Punkten keine größere Richtungsänderung auftritt. Die Konturrichtung wird mit einer optimalen Regressionsgeraden durch alle Punkte der Nachbarschaft der Breite Smooth ermittelt. Die so ermittelte Richtungsinformation für jeden Konturpunkt wird vor der Suche nach markanten Punkten mit einer Gaußmaske der Breite Smooth geglättet. Weight ist damit ein Gewichtungsfaktor für die Empfindlichkeit des Operators. Je größer Weight gewählt wird, umso weniger markante Punkte werden gefunden.

Jedes Polygon muss eine Referenz zu einer Kontur aufweisen. Fehlt diese Referenz, weil das Polygon zum Beispiel aus einer DXF-Datei eingelesen wurde, so gibt split_contours_xld einen Fehler zurück.

Parameter

- ▷ **Polygons** (input_object) ..... xld_poly(-array) ⇝ *object*
  Polygone, deren zugrundeliegende Konturen aufgespalten werden sollen.
- ▷ **Contours** (output_object) ..... xld_cont(-array) ⇝ *object*
  Aufgespaltene Konturen.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus für das Aufspalten der Konturen.
  Defaultwert: 'polygon'
  Werteliste: Mode ∈ {'polygon', 'dominant'}
- ▷ **Weight** (input_control) ..... integer ⇝ *integer*
  Gewicht für die Empfindlichkeit.
  Defaultwert: 1
- ▷ **Smooth** (input_control) ..... integer ⇝ *integer*
  Breite der Glättungsmaske.
  Defaultwert: 5

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_polygons_xld

Nachfolger

regress_contours_xld

Siehe auch

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Modul

Foundation

union_adjacent_contours_xld (Contours : UnionContours : MaxDistAbs, MaxDistRel, Mode :)

Vereinigen von Konturen, deren Endpunkte nah beieinander liegen.

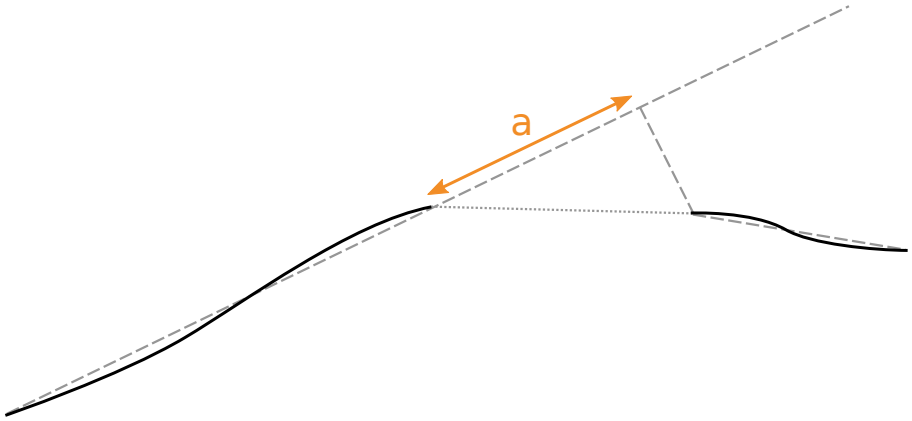
Der Operator union_adjacent_contours_xld verbindet aus einer Menge von Konturen (Contours) diejenigen miteinander, deren Endpunkte nah genug beieinander liegen. Die Vereinigung zweier Konturen ergibt sich aus der Konkatenation der Konturpunkte der beiden Eingangskonturen. Gegebenenfalls werden diese zuvor umgedreht, so dass die benachbarten Endpunkte in der resultierenden Punktliste direkte Nachbarn sind. Wenn die zu verbindenden Endpunkte aufeinander liegen, d.h. der Abstand zwischen den Konturen Null ist, wird dieser Punkt nur einmal in die Ergebniskontur eingetragen. Die Operation wird intern solange wiederholt, bis keine Konturen mehr entsprechend der Vorgaben vereinigt werden können. Als Ergebnis werden in UnionContours alle durch Vereinigung neu entstandenen Konturen sowie (unverändert) die Eingangskonturen, die mit keiner anderen Kontur verbunden werden konnten, zurückgegeben.

Parameter

Zwei Konturen werden an ihren benachbarten Endpunkten genau dann miteinander verbunden, wenn der Abstand zwischen diesen weder den in MaxDistAbs übergebenen absoluten Abstand noch einen relativen Abstand, der sich aus der Länge der längeren der beiden Konturen multipliziert mit dem Parameter MaxDistRel ergibt, überschreitet.

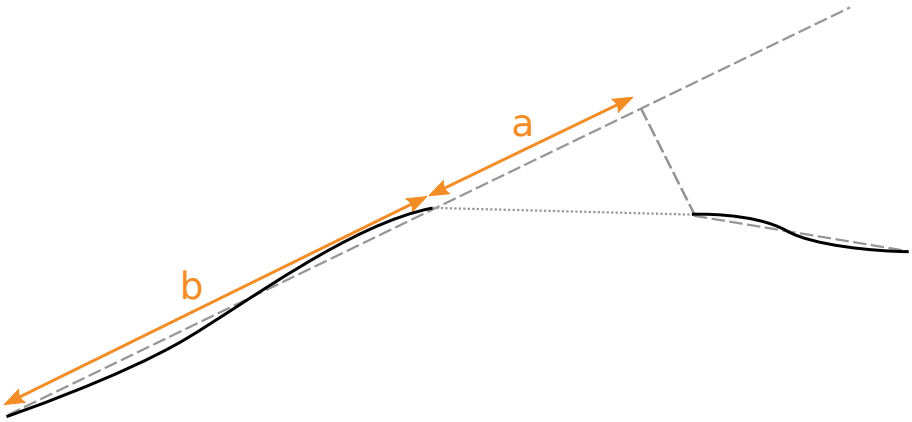
Die Reihenfolge, mit der die Konturen verbunden werden, ergibt sich in erster Linie aus den Abständen der benachbarten Endpunkte, wobei die am dichtesten benachbarten Konturen zuerst vereinigt werden. Bei gleichen Abständen wird zuerst das Konturpaar vereinigt, dessen kürzere Kontur länger ist als die kürzere Kontur des anderen Paares.

MaxDistAbs Der Parameter MaxDistAbs legt den maximalen Abstand zwischen zwei zu verbindenden Konturen fest. Der Abstand wird entlang der Regressionsgeraden der Referenzkontur gemessen. Es handelt sich also um die Länge der Projektion der Lücke zwischen den beiden Konturen auf die Regressionsgerade der Referenzkontur.



$a \leq$ MaxDistAbs

MaxDistRel Der Parameter MaxDistRel legt den maximalen relativen Abstand zwischen zwei zu verbindenden Konturen fest. Zur Berechnung des relativen Abstands wird der absolute Abstand a (siehe die Beschreibung des Parameters MaxDistAbs) durch die Länge b der Referenzkontur geteilt.



$\frac{a}{b} \leq$ MaxDistRel

Mode Mit Hilfe des Parameters Mode lässt sich steuern, wie mit eventuellen Attributen der Eingabekonturen umgegangen werden soll. Konturattribute werden beispielsweise vom Operator edges_sub_pix gesetzt und

beschreiben dort für jeden Konturpunkt die lokale Orientierung, die Kantenstärke und die Gradientenrichtung. Mit der Defaulteinstellung *'attr_keep'* werden - falls vorhanden - die Attribute in die Ausgabekonturen übernommen und gegebenenfalls, wenn eine Kontur für die Vereinigung mit einer anderen gedreht werden muss, an die neue Richtung angepasst. Bei sehr vielen Eingabekonturen kann es aus Laufzeitgründen allerdings sinnvoll sein, die Attribute zu ignorieren, wofür als Wert *'attr_forget'* angegeben werden muss. Voraussetzung dafür ist natürlich, dass für die weitere Auswertung die Attribute nicht mehr benötigt werden.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *object*
  Eingabe-Konturen.
- ▷ **UnionContours** (output_object) ..... xld_cont-array ⇝ *object*
  Ausgabe-Konturen.
- ▷ **MaxDistAbs** (input_control) ..... real ⇝ *real*
  Maximaler Abstand der Endpunkte der Konturen.
  Defaultwert: 10.0
  Typischer Wertebereich: 0.0 ≤ MaxDistAbs
- ▷ **MaxDistRel** (input_control) ..... real ⇝ *real*
  Maximaler Abstand der Endpunkte der Konturen relativ zur Länge der zu längeren Kontur.
  Defaultwert: 1.0
  Typischer Wertebereich: 0.0 ≤ MaxDistRel
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus für die Behandlung der Konturattribute
  Defaultwert: 'attr_keep'
  Werteliste: `Mode` ∈ {'attr_keep', 'attr_forget'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`split_contours_xld`, `select_contours_xld`

Alternativen

`union_collinear_contours_xld`, `union_collinear_contours_ext_xld`, `union_cocircular_contours_xld`, `union_straight_contours_xld`, `union_cotangential_contours_xld`

Siehe auch

`edges_sub_pix`, `threshold_sub_pix`, `gen_polygons_xld`, `split_contours_xld`, `select_contours_xld`, `get_contour_xld`, `get_contour_attrib_xld`

Modul

Foundation

```
union_cocircular_contours_xld ( Contours : UnionContours : MaxArcAngleDiff, MaxArcOverlap,
    MaxTangentAngle, MaxDist, MaxRadiusDiff, MaxCenterDist,
    MergeSmallContours, Iterations : )
```

Zusammenfassen von Konturen, die zu denselben Kreisen gehören.

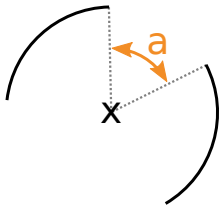
Der Operator `union_cocircular_contours_xld` vereint alle Konturen, die zu denselben Kreisen gehören. Das Ergebnis ist eine Menge von Konturen, in der Konturen verbunden sind, die auf denselben Kreisen liegen.

Der Algorithmus versucht Konturen zusammenzufassen, indem er Kreise an jede Kontur anpasst und diese bezüglich Radius, Kreismittelpunkt und Abstand auf dem Kreisbogen vergleicht. Die Liste der Konturen wird in aufsteigender Reihenfolge der Radiuslänge bearbeitet. Konturen, an die kein Kreis angepasst werden kann, werden in einem optionalen zweiten Durchgang hinzugefügt.

Die Schwellenwertparameter bestimmen, ob Konturen zu demselben Kreis gehören. Alle Schwellenwerte müssen gleichzeitig erfüllt werden, damit zwei Konturen verbunden werden.

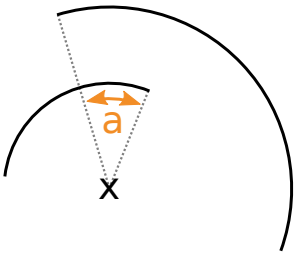
Parameter

MaxArcAngleDiff Der Parameter MaxArcAngleDiff definiert die maximale Winkeldistanz auf dem Kreisbogen zwischen dem Ende der einen und dem Start der zweiten Kontur.



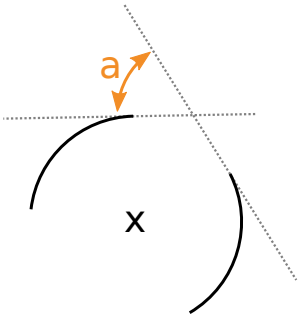
$a \leq$ MaxArcAngleDiff

MaxArcOverlap MaxArcOverlap bestimmt die Winkeldistanz, um die sich zwei Konturen überlappen dürfen.



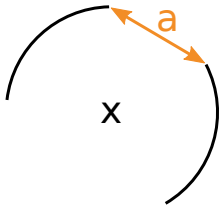
$a \leq$ MaxArcOverlap

MaxTangentAngle MaxTangentAngle beschreibt den maximalen Winkel zwischen den Kreistangenten an den zwei Konturenden und der Verbindungsstrecke zwischen ihnen.



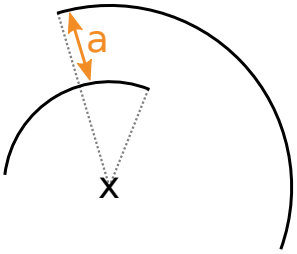
$a \leq$ MaxTangentAngle

MaxDist MaxDist ist die Schwelle für die absolute Pixeldistanz zwischen End- und Startpunkt der Konturen.



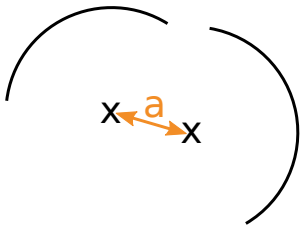
$a \leq$ MaxDist

MaxRadiusDiff MaxRadiusDiff beschreibt die maximale Differenz der Radien der angepassten Kreise,



$$a \leq \texttt{MaxRadiusDiff}$$

MaxCenterDist MaxCenterDist die maximale euklidische Distanz zwischen den Mittelpunkten der angepassten Kreise.



$$a \leq \texttt{MaxCenterDist}$$

MergeSmallContours Wenn der Parameter MergeSmallContours auf *'true'* gesetzt ist, werden auch Konturen ohne angepasste Kreise verschmolzen. Eine solche Kontur wird als passend angenommen, wenn jeder Konturpunkt einen Abstand zum Kreismittelpunkt innerhalb von MaxRadiusDiff zum Kreisradius besitzt und wenn der Winkel der Verbindungsstrecke von zwei aufeinander folgenden Punkten der Kontur innerhalb der Grenze von MaxTangentAngle zur Kreistangenten liegt. Zusätzlich muss die Bedingung bezüglich MaxDist erfüllt sein.

Iterations Die Verbindung von Konturen führt zu neuen Kreisparametern. Deshalb kann eine Iteration des Algorithmus zu weiteren Verbindungen führen. Der Parameter Iterations gibt die Anzahl der Iterationen an. Mehr als zwei Iterationen sind selten nötig.

Für jede mögliche Zusammenfassung von zwei Konturen werden die Kosten berechnet. Diese ergeben sich aus der Summation der Distanzen bezüglich der obigen Schwellenwerte. Die Distanzen werden für die Vergleichbarkeit mit den Schwellenwerten skaliert, so dass sie zwischen 0.0 und 1.0 liegen. Wenn zwei oder mehr Konturstartpunkte an einen Konturendpunkt passen, so wird die Verbindung mit den kleinsten Kosten verschmolzen.

Vor Aufruf sollte sichergestellt sein, dass sich die Konturen durch Linien und Kreisbögen annähern lassen. Dies kann zum Beispiel durch eine Vorverarbeitung mit dem Operator split_contours_xld erreicht werden.

Achtung

Bereits geschlossene Konturen werden nicht mehr in die Vereinigung einbezogen. D.h., auch wenn eine Kontur auf dem gleichen Kreis liegt wie eine bereits geschlossene Kreiskontur, werden beide Konturen getrennt zurückgeliefert

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *object*
  Zu vereinigende Konturen.
- ▷ **UnionContours** (output_object) ..... xld_cont-array ⇝ *object*
  Vereinigte Konturen.
- ▷ **MaxArcAngleDiff** (input_control) ..... angle.rad ⇝ *real* / integer
  Maximale Winkeldistanz zwischen zwei Kreisbögen.
  Defaultwert: 0.5
  Wertevorschläge: MaxArcAngleDiff ∈ {0.25, 0.5, 0.75, 1.0}
- ▷ **MaxArcOverlap** (input_control) ..... angle.rad ⇝ *real* / integer
  Maximale Überlappung von zwei Kreisbögen.
  Defaultwert: 0.1
  Wertevorschläge: MaxArcOverlap ∈ {0.0, 0.1, 0.2}
- ▷ **MaxTangentAngle** (input_control) ..... angle.rad ⇝ *real* / integer
  Maximaler Winkel zwischen Verbindungsstrecke und Tangenten der Kreisbögen.
  Defaultwert: 0.2
  Wertevorschläge: MaxTangentAngle ∈ {0.1, 0.2, 0.3, 0.4, 0.5}
- ▷ **MaxDist** (input_control) ..... number ⇝ *real* / integer
  Maximale Absolutlänge der Lücke zwischen zwei Kreisbögen in Bildpunkten.
  Defaultwert: 30
  Wertevorschläge: MaxDist ∈ {10, 30, 50, 70}
- ▷ **MaxRadiusDiff** (input_control) ..... number ⇝ *real* / integer
  Maximale Differenz der Radien der Kreise an zwei Kreisbögen.
  Defaultwert: 10
  Wertevorschläge: MaxRadiusDiff ∈ {10, 20, 30}

▷ **MaxCenterDist** (input_control) ..... number ⇝ *real* / integer
Maximale Distanz zwischen den Kreismittelpunkten zweier Kreisbögen.
Defaultwert: 10
Wertevorschläge: MaxCenterDist ∈ {10, 20, 30}

▷ **MergeSmallContours** (input_control) ..... string ⇝ *string*
Legt fest, ob auch Konturen ohne angepassten Kreis vereinigt werden sollen.
Defaultwert: 'true'
Werteliste: MergeSmallContours ∈ {'true', 'false'}

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert: 1
Wertevorschläge: Iterations ∈ {1, 2}

Komplexität

$O(n^2)$ für n Konturen.

Ergebnis

union_cocircular_contours_xld gibt TRUE zurück, wenn alle Parameter korrekt angegeben sind.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

split_contours_xld, select_contours_xld

Alternativen

union_collinear_contours_xld, union_cotangential_contours_xld, union_straight_contours_xld, union_adjacent_contours_xld, union_collinear_contours_ext_xld

Modul

Foundation

```
union_collinear_contours_ext_xld ( Contours : UnionContours : MaxDistAbs, MaxDistRel, MaxShift,
    MaxAngle, MaxOverlap, MaxRegrError, MaxCosts, WeightDist,
    WeightShift, WeightAngle, WeightLink, WeightRegr, Mode : )
```

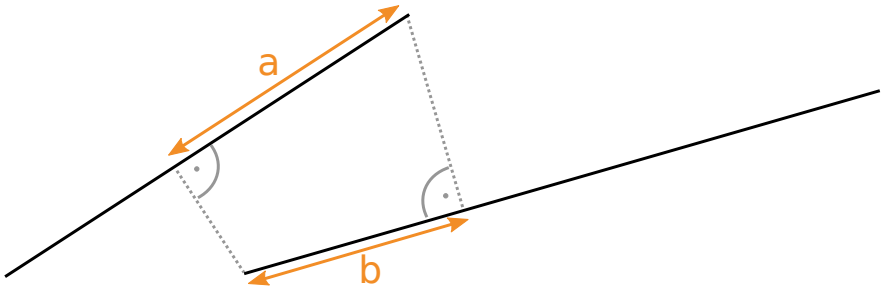
Vereinigen von kollinearen Konturen (Operator mit erweitertem Funktionsumfang).

Wie der Operator union_collinear_contours_xld vereinigt union_collinear_contours_ext_xld aus einer Menge von geraden, linienförmige Konturen (Contours) diejenigen miteinander, die zueinander kollinear sind, und gibt das Ergebnis in UnionContours zurück. Das grundsätzliche Verhalten ist identisch zu union_collinear_contours_xld und wird dort beschrieben, es stehen jedoch zahlreiche weitere Parameter zur Verfügung, die es erlauben, den Vereinigungsprozess genauer zu steuern.

Parameter

Die Parameter MaxDistAbs, MaxDistRel, MaxShift und MaxAngle sind identisch wie wie beim Operator union_collinear_contours_xld beschrieben.

MaxOverlap Zusätzlich lässt sich hier mit MaxOverlap ein Überlappungsbereich zwischen beiden Konturen definieren, so dass auch Konturen, die sich (geringfügig) überlappen, miteinander vereinigt werden können. Intern wird dabei sichergestellt, dass sich – unabhängig von dem mit MaxOverlap übergebenen Parameter – die Konturen nicht um mehr als ein Drittel ihrer Länge überlappen.



$min(a, b) \leq$ MaxOverlap

MaxRegrError Der Parameter MaxRegrError wird z.Zt. noch nicht verwendet. Er soll später sicherstellen, dass nach der Vereinigung zweier Konturen die resultierende Kontur nicht zu stark von ihrer Regressionsgeraden abweicht. Dieser Wert ist defaultmäßig ausgeschaltet (*-1*), da er unter normalen Bedingungen nicht notwendig und der Rechenaufwand sehr hoch ist. Unter bestimmten Bedingungen können jedoch Ergebniskonturen entstehen, die relativ stark um ihre Regressionsgerade schwanken („wellige“ Eingangskonturen, Shift und Winkeldifferenz nah an den – relativ hoch eingestellten – Grenzen, rekursive Vereinigung sehr vieler kurzer Konturen).

MaxCosts Mit dem Parameter MaxCosts ist es möglich, die Gesamtkosten der Vereinigung zu begrenzen und so nicht nur die Vereinigung der Konturpaare auszuschließen, die in einer Disziplin die vorgegebenen Grenzwerte überschreiten, sondern auch jener, die in mehreren Disziplinen die Grenzen nur knapp einhalten können. Dabei werden die Kosten immer auf einen Wert zwischen 0 und 1 normiert, der Wert 1 bedeutet dabei, dass alle Werte auf dem jeweiligen Grenzwert liegen.

WeightDist, WeightShift, WeightAngle, WeightLink, and WeightRegr Der Einfluss der verschiedenen Messwerte auf die Gesamtkosten wird durch die Übergabe einer entsprechenden Gewichtung mit Hilfe der Parameter WeightDist, WeightShift, WeightAngle, WeightLink und WeightRegr eingestellt. Dabei ist zu beachten, dass die Gesamtkosten immer auf 1 normiert werden. Der Parameter WeightRegr wird z.Zt. nicht verwendet.

Mode Der Parameter Mode steuert – wie beim Operator union_collinear_contours_xld – die Behandlung der Attribute der Eingabekontur.

Parameter

▷ **Contours** (input_object) . . . . . xld_cont-array ⇝ *object*
Eingabe-Konturen.

▷ **UnionContours** (output_object) . . . . . xld_cont-array ⇝ *object*
Ausgabe-Konturen.

▷ **MaxDistAbs** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand der Endpunkte in Richtung der Referenzregressionsgeraden.
Defaultwert: 10.0
Typischer Wertebereich: 0.0 ≤ MaxDistAbs

▷ **MaxDistRel** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand der Endpunkte in Richtung der Referenzregressionsgeraden relativ zur Länge der zu verlängernden Kontur.
Defaultwert: 1.0
Typischer Wertebereich: 0.0 ≤ MaxDistRel

▷ **MaxShift** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand der Konturen von der Referenzregressionsgeraden.
Defaultwert: 2.0
Typischer Wertebereich: 0.0 ≤ MaxShift

▷ **MaxAngle** (input_control) . . . . . real ⇝ *real*
Maximale Richtungsdifferenz.
Defaultwert: 0.1
Typischer Wertebereich: 0.0 ≤ MaxAngle ≤ 0.78539816339

▷ **MaxOverlap** (input_control) . . . . . real ⇝ *real*
Maximal erlaubter Überlappungsbereich.
Defaultwert: 0.0
Typischer Wertebereich: 0.0 ≤ MaxOverlap

▷ **MaxRegrError** (input_control) ..... real ⇝ *real*
Maximaler Regressionsfehler der resultierenden Konturen (NICHT BENUTZT).
Defaultwert: -1.0

▷ **MaxCosts** (input_control) ..... real ⇝ *real*
Schwellenwert zum Begrenzen der Gesamtkosten der Vereinigung
Defaultwert: 1.0
Typischer Wertebereich: 0.0 ≤ MaxCosts

▷ **WeightDist** (input_control) ..... real ⇝ *real*
Einfluss des Abstandes in Linienrichtung auf die Gesamtkosten.
Defaultwert: 1.0
Typischer Wertebereich: 0.0 ≤ WeightDist

▷ **WeightShift** (input_control) ..... real ⇝ *real*
Einfluss des Abstandes von der Regressionsgeraden auf die Gesamtkosten.
Defaultwert: 1.0
Typischer Wertebereich: 0.0 ≤ WeightShift

▷ **WeightAngle** (input_control) ..... real ⇝ *real*
Einfluss der Winkeldifferenz auf die Gesamtkosten.
Defaultwert: 1.0
Typischer Wertebereich: 0.0 ≤ WeightAngle

▷ **WeightLink** (input_control) ..... real ⇝ *real*
Einfluss von Störungen der Linie durch das Verbindungsstück (Überlappung und Winkelabweichung) auf die Gesamtkosten.
Defaultwert: 1.0
Typischer Wertebereich: 0.0 ≤ WeightLink

▷ **WeightRegr** (input_control) ..... real ⇝ *real*
Einfluss von Regressionsfehlern der resultierenden Kontur auf die Gesamtkosten (NICHT BENUTZT).
Defaultwert: 0.0
Typischer Wertebereich: 0.0 ≤ WeightRegr

▷ **Mode** (input_control) ..... string ⇝ *string*
Modus für die Behandlung der Konturattribute
Defaultwert: 'attr_keep'
Werteliste: `Mode` ∈ {'attr_keep', 'attr_forget'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`split_contours_xld`, `select_contours_xld`

Alternativen

`union_collinear_contours_xld`, `union_cocircular_contours_xld`, `union_cotangential_contours_xld`, `union_adjacent_contours_xld`, `union_straight_contours_xld`

Siehe auch

`edges_sub_pix`, `threshold_sub_pix`, `gen_polygons_xld`, `split_contours_xld`, `select_contours_xld`, `get_contour_xld`, `get_contour_attrib_xld`

Modul

Foundation

union_collinear_contours_xld (
Contours : UnionContours : MaxDistAbs, MaxDistRel, MaxShift, MaxAngle, Mode :)

Vereinigen von kollinearen Konturen.

Der Operator union_collinear_contours_xld vereinigt aus einer Menge von Konturen (Contours) diejenigen miteinander, die zueinander kollinear sind. Die Vereinigung zweier Konturen ergibt sich aus der Verkettung der Konturpunkte der beiden Eingabekonturen. Sie werden zusammen mit den Konturen, die mit keiner anderen Kontur vereinigt werden konnten, in UnionContours zurück gegeben.

Typische Anwendungen

union_collinear_contours_xld kann zum Schließen von Lücken in extrahierten Kanten oder Linien verwendet werden. Diese Lücken können z.B. durch geringen Kontrast in den Bildern entstehen. Der Operator kann auch verwendet werden, um Konturen zu verbinden, die aufgrund von Kreuzungspunkten der Kanten oder Linien voneinander getrennt sind. Typischerweise ist union_collinear_contours_xld Teil des Prozesses, in dem extrahierte Kanten oder Linien gruppiert und ausgewählt werden, um semantisch aussagekräftige Einheiten zu bilden.

Beispiel

Das folgende Beispiel zeigt die Wirkung von union_collinear_contours_xld. Auf der linken Seite ist eine Menge unzusammenhängender Konturen dargestellt. Die Konturen, die jeweils auf der selben Seite des Rechtecks liegen, sind näherungsweise kollinear. Auf der rechten Seite ist das Ergebnis von union_collinear_contours_xld dargestellt. Es zeigt, dass die Konturen, die näherungsweise kollinear waren miteinander vereinigt wurden. Die Konturen, die mehr oder weniger senkrecht zueinander stehen, wurden nicht miteinander vereinigt.



(1) (2)

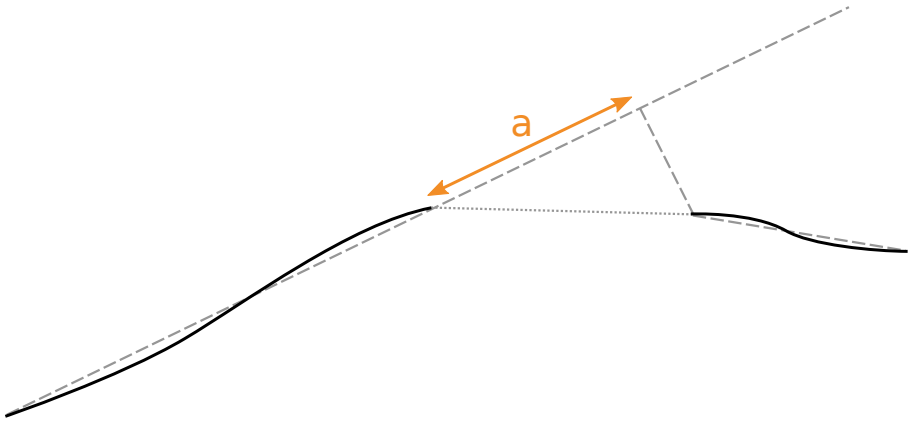
(1) Unzusammenhängende Eingabekonturen. (2) Ausgabekonturen, bei denen die näherungsweise kollinearen Konturen miteinander vereinigt sind.

Parameter

Die Parameter MaxDistAbs, MaxDistRel, MaxShift und MaxAngle legen fest, wann zwei Konturen als kollinear angesehen werden. Der Parameter Mode steuert die Behandlung von Attributen der Eingabekonturen.

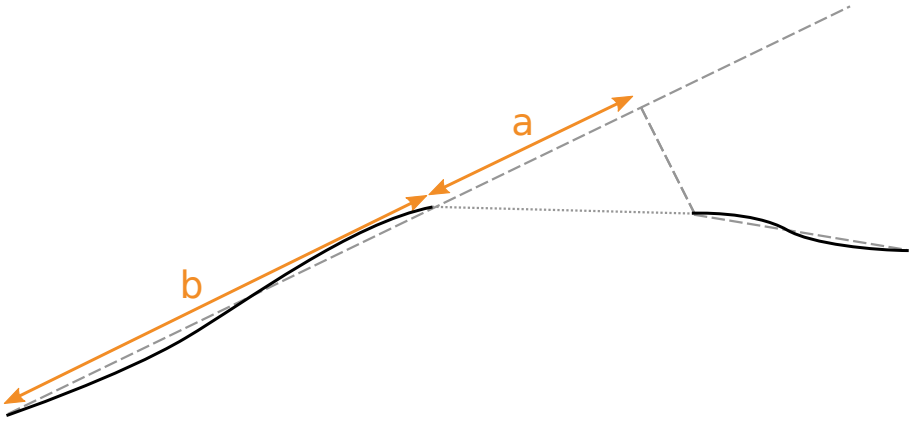
Prinzipiell sind die Maße für die ersten drei Parameter abhängig von der Reihenfolge, d.h., sie hängen davon ab, welche Kontur als Referenzkontur betrachtet wird, die durch die zweite Kontur erweitert werden soll. Um diese Abhängigkeit von der Reihenfolge zu vermeiden, werden die Maße jeweils in beiden Richtungen ermittelt. Es wird dann die Reihenfolge verwendet, die zu den kleineren Werten für die drei Maße führt. Beachten Sie, dass in den Illustrationen unten immer die linke Kontur als Referenzkontur verwendet wird.

MaxDistAbs Der Parameter MaxDistAbs legt den maximalen Abstand zwischen zwei zu verbindenden Konturen fest. Der Abstand wird entlang der Regressionsgeraden der Referenzkontur gemessen. Es handelt sich also um die Länge der Projektion der Lücke zwischen den beiden Konturen auf die Regressionsgerade der Referenzkontur.



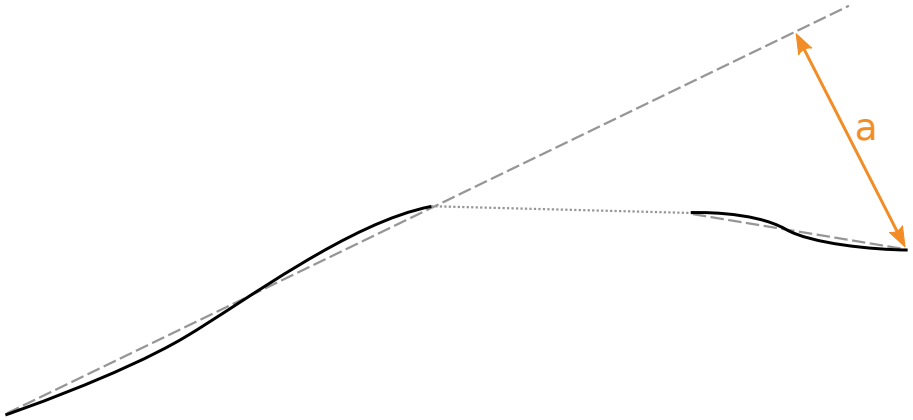
$a \leq$ MaxDistAbs

MaxDistRel Der Parameter MaxDistRel legt den maximalen relativen Abstand zwischen zwei zu verbindenden Konturen fest. Zur Berechnung des relativen Abstands wird der absolute Abstand a (siehe die Beschreibung des Parameters MaxDistAbs) durch die Länge b der Referenzkontur geteilt.



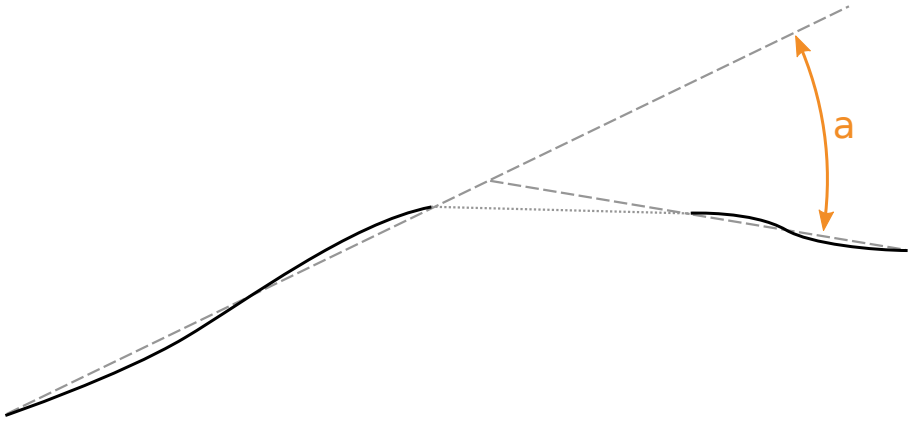
$\frac{a}{b} \leq$ MaxDistRel

MaxShift Der Parameter MaxShift legt den maximalen Abstand der zweiten Kontur von der Regressionsgeraden der Referenzkontur fest. Dieser Abstand wird senkrecht zur Regressionsgeraden der Referenzkontur gemessen.



$a \leq$ MaxShift

MaxAngle Der Parameter MaxAngle legt den maximalen Winkel (im Bogenmaß) zwischen den Regressionsgeraden der beiden Konturen fest.



$a \leq$ MaxAngle

Mode Der Parameter Mode steuert die Behandlung der Attribute, die für die Eingabekonturen gesetzt sein können, z.B., wenn die Konturen mit edges_sub_pix erzeugt wurden.

'attr_keep' Alle Attribute werden in die Ausgabekonturen kopiert.
Globale Attribute werden allerdings nur für solche Konturen übernommen, die nicht mit anderen Konturen vereinigt wurden. Im Allgemeinen ist es nicht möglich, aus den globalen Attributen von einzelnen Konturen die entsprechenden globalen Attribute einer vereinigten Kontur zu berechnen.

'attr_forget' Die Ausgabekonturen enthalten keine Attribute. Diese Einstellung kann aus Laufzeitgründen gewählt werden, wenn die Attribute in den folgenden Arbeitsschritten nicht benötigt werden.

Um abzufragen, welche Attribute verfügbar sind, kann `query_contour_attribs_xld` verwendet werden.

Implementierungsdetails

Die Eingabekonturen werden paarweise analysiert und gegebenenfalls vereinigt. Dieser Prozess wird so oft wiederholt, bis keine weiteren Konturen mehr vereinigt werden können. Falls notwendig, wird die Reihenfolge umgekehrt, in der die Punkte innerhalb einer Kontur gespeichert sind, so dass die benachbarten Endpunkte der Konturen in der vereinigten Kontur direkt aufeinanderfolgen.

Zwei Konturen werden nur dann vereinigt, wenn alle Kriterien erfüllt sind, wenn also alle Maße kleiner als die mit den entsprechenden Parametern festgelegten Schwellenwerte sind. Eine weitere Voraussetzung ist, dass sich die Konturen nur unwesentlich überlappen dürfen (siehe `union_collinear_contours_ext_xld` für eine Illustration, die Überlappung zeigt). Es wird lediglich eine Überlappung von maximal 0.5 Pixeln toleriert. Um einen größeren Überlappungsbereich zuzulassen, kann `union_collinear_contours_ext_xld` verwendet werden.

Neben der Suche nach Paaren von Konturen, die vereinigt werden können, ist die Reihenfolge, in der die Konturen vereinigt werden für das Verhalten von `union_collinear_contours_xld` wichtig. Hierfür werden alle Paare von Konturen durch die Berechnung eines Kostenmaßes bewertet. Als Kostenmaß wird die Summe der einzelnen Maße verwendet, wobei jedes Maß auf den entsprechend Schwellenwert normiert wird, der mit den Parametern `MaxDistAbs`, `MaxDistRel`, `MaxShift` und `MaxAngle` festgelegt wurde. Das Paar von Konturen, welches die geringsten Kosten aufweist, wird als erstes vereinigt. Daraufhin werden die Kosten aktualisiert, um die neu erzeugte Kontur zu berücksichtigen. Basierend auf den aktualisierten Kosten wird das als nächstes zu vereinigende Paar ermittelt.

Die Definition von Kollinearität, wie sie in `union_collinear_contours_xld` verwendet wird, hängt von den Regressionsgeraden der einzelnen Konturen ab. Um unerwartete Ergebnisse zu vermeiden, sollten die Eingabekonturen einigermaßen geraden Linien entsprechen. Dies kann z.B. mit dem Operator `segment_contours_xld` erreicht werden. Es ist zu beachten, dass geschlossene Konturen mit keiner anderen Kontur vereinigt werden.

Einschränkungen und Alternativen

`union_collinear_contours_xld` kann nur zum Vereinigen von Konturen verwendet werden, die näherungsweise auf einer Geraden liegen. Zum Vereinigen von Konturen, die genähert auf einem Kreis liegen, kann `union_cocircular_contours_xld` verwendet werden. Um Konturen zu vereinigen, die beliebige Formen darstellen, kann `union_cotangential_contours_xld` verwendet werden. Sollen nur sehr kleine Lücken zwischen Konturen geschlossen werden, unabhängig von der Richtung in der die angrenzenden Konturen verlaufen, so kann hierfür `union_adjacent_contours_xld` verwendet werden.

Wenn die Konfigurationsmöglichkeiten von `union_collinear_contours_xld` nicht ausreichen, können die Kriterien zur Vereinigung von Konturen mit `union_collinear_contours_ext_xld` noch detaillierter festgelegt werden. Unter anderem können hiermit die maximale Überlappung der Konturen festgelegt und die Gesamtkosten der Vereinigung begrenzt werden. Durch letzteres kann verhindert werden, dass Paare von Konturen vereinigt werden, bei denen die Maße für mehrere Kriterien bereits nahe an den jeweiligen Schwellenwerten liegen.

Parameter

▷ **`Contours`** (input_object) ..... xld_cont-array ⇝ *object*
Eingabekonturen.

▷ **`UnionContours`** (output_object) ..... xld_cont-array ⇝ *object*
Ausgabekonturen.

▷ **`MaxDistAbs`** (input_control) ..... real ⇝ *real*
Maximale Länge der Lücke zwischen zwei Konturen, gemessen entlang der Regressionsgeraden der Referenzkontur.
Defaultwert: 10.0
Typischer Wertebereich: $0.0 \leq$ MaxDistAbs

▷ **MaxDistRel** (input_control) . . . . . real ⇝ *real*
Maximale Länge der Lücke zwischen zwei Konturen, relativ zur Länge der Referenzkontur. Beides wird entlang der Regressionsgeraden der Referenzkontur gemessen.
Defaultwert: 1.0
Typischer Wertebereich: 0.0 ≤ MaxDistRel

▷ **MaxShift** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand der zweiten Kontur von der Regressionsgeraden der Referenzkontur.
Defaultwert: 2.0
Typischer Wertebereich: 0.0 ≤ MaxShift

▷ **MaxAngle** (input_control) . . . . . real ⇝ *real*
Maximaler Winkel zwischen den Regressionsgeraden von zwei Konturen.
Defaultwert: 0.1
Typischer Wertebereich: 0.0 ≤ `MaxAngle` ≤ 0.78539816339

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Modus, der die Behandlung der Konturattribute festlegt.
Defaultwert: 'attr_keep'
Werteliste: `Mode` ∈ {'attr_keep', 'attr_forget'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`edges_sub_pix`, `lines_gauss`, `segment_contours_xld`, `select_contours_xld`, `select_shape_xld`, `split_contours_xld`, `threshold_sub_pix`

Nachfolger

`fit_line_contour_xld`, `select_contours_xld`, `select_shape_xld`

Alternativen

`union_collinear_contours_ext_xld`, `union_straight_contours_xld`, `union_cocircular_contours_xld`, `union_cotangential_contours_xld`, `union_adjacent_contours_xld`

Siehe auch

`get_contour_xld`, `get_contour_attrib_xld`, `query_contour_attribs_xld`

Modul

Foundation

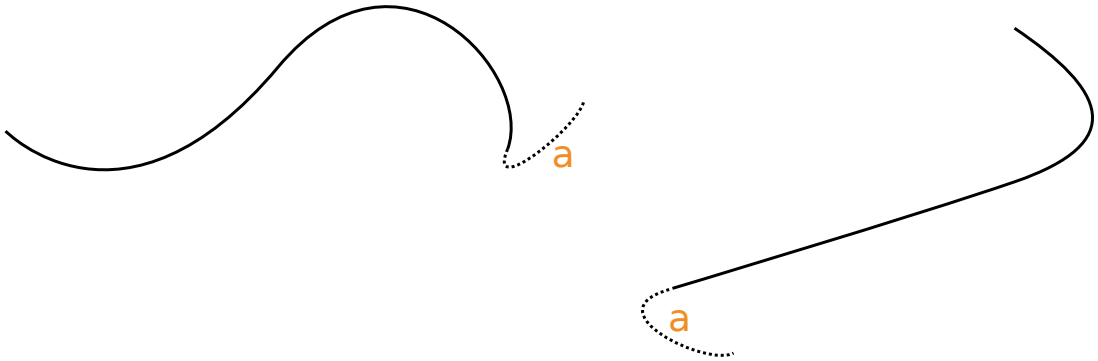
```
union_cotangential_contours_xld ( Contours : UnionContours : FitClippingLength, FitLength,
    MaxTangAngle, MaxDist, MaxDistPerp, MaxOverlap, Mode : )
```

Vereinigen von kotangentialen Konturen.

Der Operator `union_cotangential_contours_xld` vereinigt Konturen gemäß der lokalen Krümmungen an ihren Endpunkten.

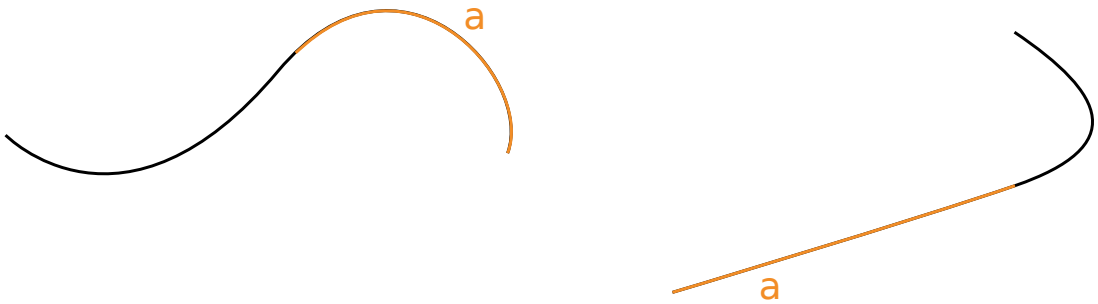
Parameter

`FitClippingLength` Dazu werden zunächst die für die jeweiligen Konturen `Contours` zu betrachtenden Endstücke bestimmt. `FitClippingLength` dient dazu, die Konturen um eine bestimmte euklidische Länge zu beschneiden, so dass eventuelle Artefakte an den Enden der Konturen für die weiteren Berechnungen ignoriert werden können.



$a \leq$ FitClippingLength

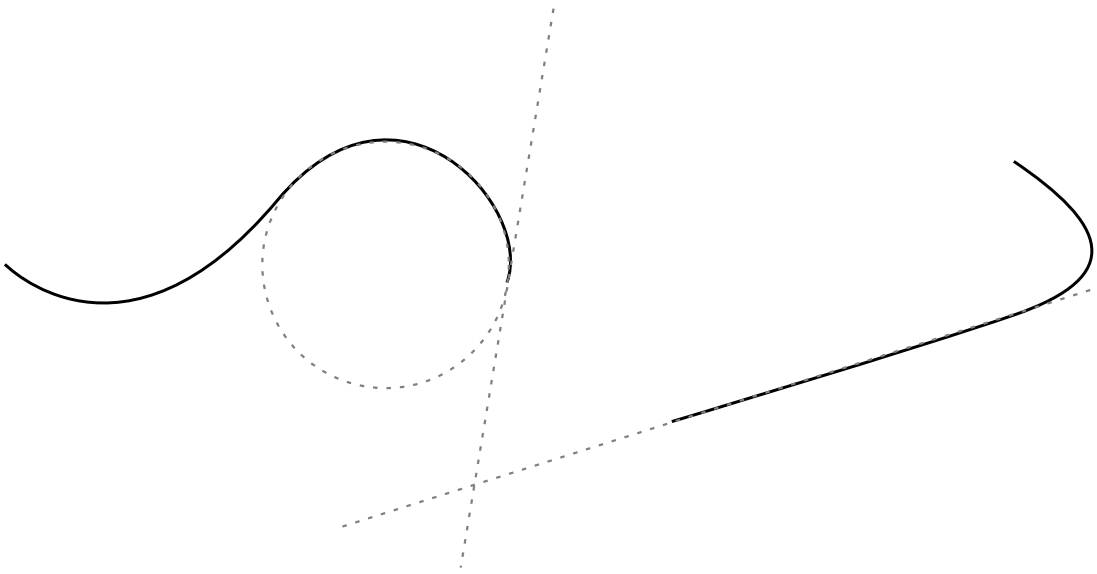
FitLength FitLength legt die euklidische Länge der für die Vereinigung zu berücksichtigenden Endstücke der Konturen fest. D.h., an jeder Seite einer Kontur werden einem Endstück soviele Punkte zugeordnet, dass die Länge des Endstücks mindestens dem übergebenen Wert entspricht. Wird kein numerischer Wert sondern der Wert *'auto'* übergeben, wird jede Eingangskontur in Geraden und Kreisbögen segmentiert. Von diesen Segmenten dienen dann die äußersten Segmente als Endstücke der Kontur. Da dieses Verfahren sehr rechenintensiv ist, ist eine numerisch festgelegte Länge, wenn möglich, vorzuziehen.



$a \leq$ FitLength

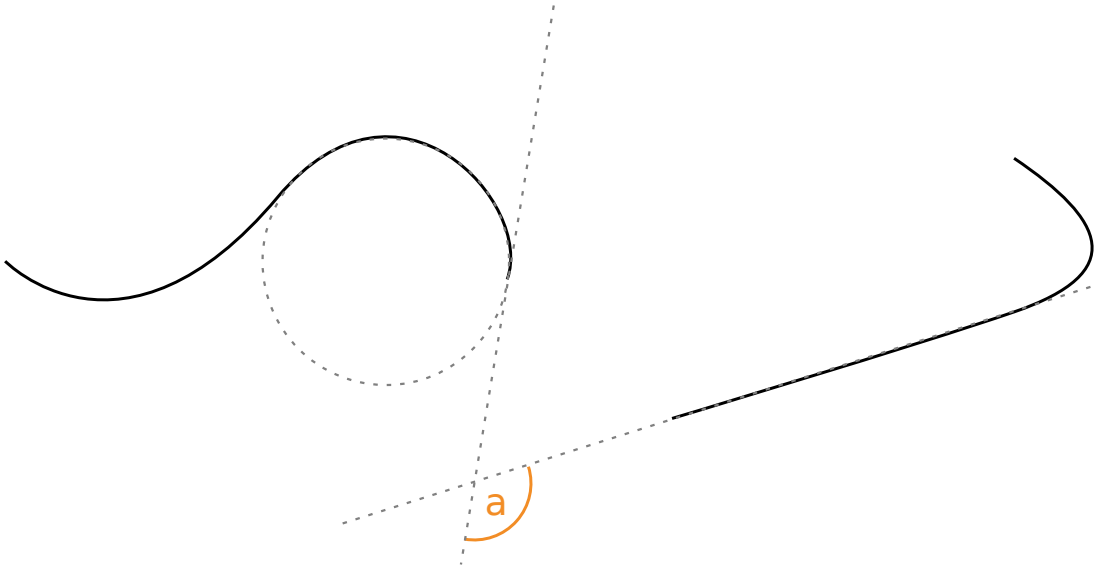
Der Operator versucht nun, an jedes Endstück einer Kontur einen Kreis anzupassen. Von der ursprünglichen Kontur werden die äußersten Punkte der Kontur ermittelt, die nah genug (gemessen am mittleren quadratischen Fehler der Kreisanpassung) an ihrem jeweiligen Kreis liegen. Die Kreistangenten an diesen Punkten bestimmen die lokalen Krümmungen an den Enden der jeweiligen Kontur.

Kann kein Kreis an ein Endstück angepasst werden, wird statt dessen versucht, eine Regressionsgerade anzupassen. Die lokale Krümmung der Kontur an ihren Enden entspricht dann der Richtung der Regressionsgeraden.



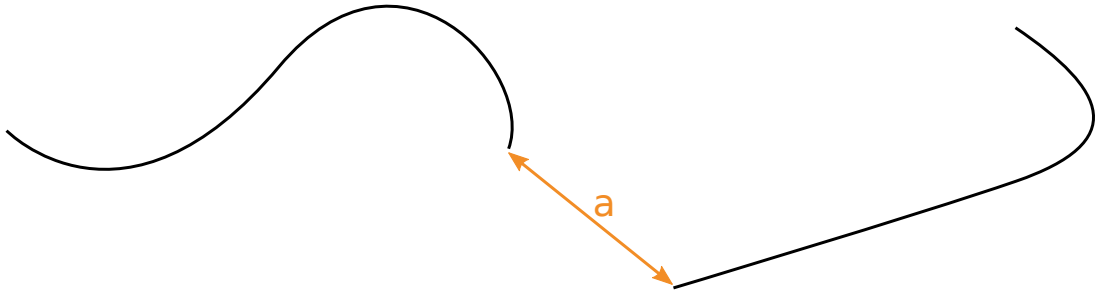
Mithilfe der berechneten lokalen Krümmungen an den Endpunkten der einzelnen Eingabekonturen können nun die verschiedenen Eingabekonturen vereinigt werden. Zwei Konturen werden dann vereinigt, wenn sämtliche Schwellenwerte, eingehalten werden:

MaxTangAngle Mit dem Parameter MaxTangAngle wird der maximale Winkel zwischen den Tangenten zweier Konturen angegeben.



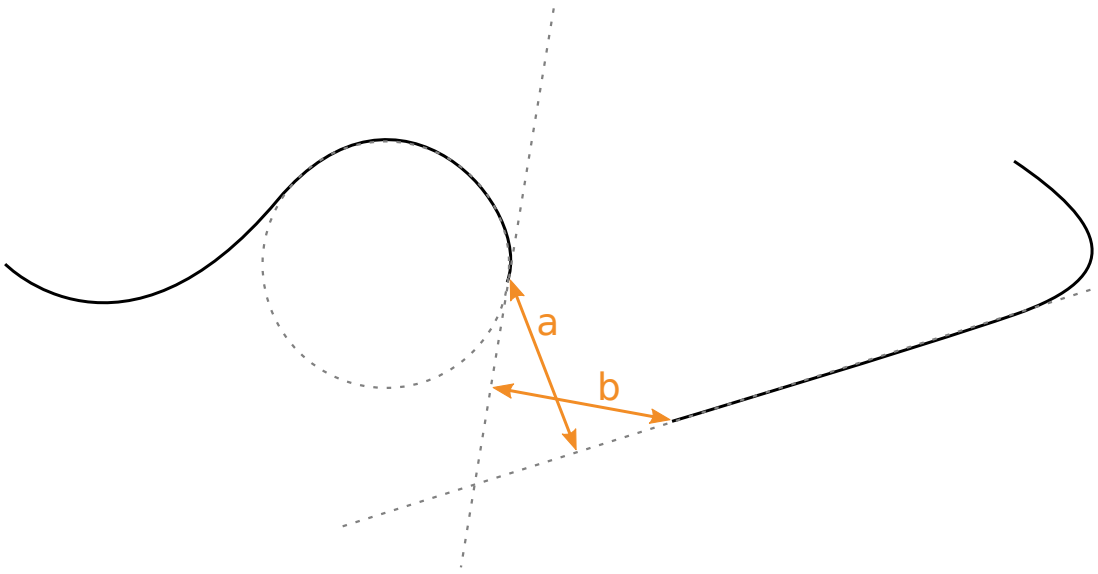
$a \leq$ MaxTangAngle

MaxDist Der Parameter MaxDist gibt die maximale Entfernung der Endpunkte der Konturen voneinander an.



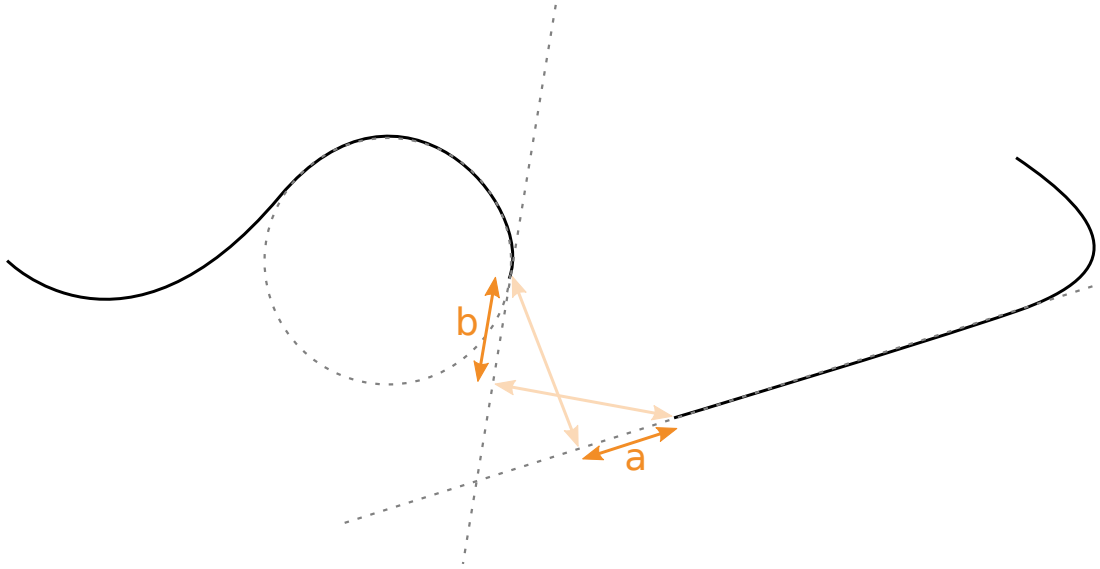
$a \leq$ MaxDist

MaxDistPerp Mit MaxDistPerp wird der maximale Abstand der Endpunkte senkrecht zu den Tangentenrichtungen bestimmt. Dieser Abstand ist das Minimum der Abstände zwischen den Endpunkten der Konturen senkrecht zur jeweils anderen Kontur.



$min(a, b) \leq$ MaxDistPerp

MaxOverlap Die Überlappung der Konturen wird mit dem Parameter MaxOverlap berücksichtigt. Das Ausmaß der Überlappung wird durch den Abstand zwischen dem Endpunkt der ersten Kontur und seiner Projektion auf die Tangente der zweiten Kontur und durch den Abstand zwischen dem Endpunkt der zweiten Kontur und seiner Projektion auf die Tangente der ersten Kontur berechnet. Beide Abstände müssen kürzer als MaxOverlap sein. Im Gegensatz zu den vorherigen Schwellenwerten kann MaxOverlap auch negativ sein. Negative Werte bezeichnen dann den minimalen Abstand der Endpunkte in Tangentenrichtung.



$min(a, b) \leq$ MaxOverlap. Hier sind *'a'* und *'b'* negativ. In der Referenz des Operators union_collinear_contours_ext_xld finden Sie eine Illustration, in der *'a'* und *'b'* positiv sind.

Es ist möglich, dass eine Kontur durch union_cotangential_contours_xld geschlossen wird oder mehrere Konturen zu einer einzigen geschlossenen Kontur vereinigt werden, falls die oben genannten Kriterien für die Endstücke erfüllt werden. Es ist zu beachten, dass bereits geschlossene Konturen nicht mehr in eine Vereinigung einbezogen werden.

Gibt es Konturpaare, für die die oben genannten Kriterien zutreffen, wird intern die Reihenfolge der Vereinigungen berechnet. Hierfür werden die einzelnen möglichen Vereinigungen bewertet. Je näher die oben beschriebenen Winkel und Abstände an 0 liegen, desto höher ist ihre Bewertungszahl. Die Reihenfolge der Konturenvereinigung richtet sich dann nach der Höhe dieser Bewertung.

Der Parameter Mode gibt an, ob bestehende Attribute in die vereinigten Konturen kopiert werden sollen. *'attr_keep'* kopiert die Attribute, was die Laufzeit negativ beeinflussen kann, und *'attr_forget'* veranlasst den Operator, die Attribute zu ignorieren. Weitere Details finden sich bei der Beschreibung des Parameters Mode des Operators union_adjacent_contours_xld).

Parameter

- ▷ **Contours** (input_object) . . . . . xld_cont-array ⇝ *object*
  Eingabekonturen.
- ▷ **UnionContours** (output_object) . . . . . xld_cont-array ⇝ *object*
  Ausgabekonturen.
- ▷ **FitClippingLength** (input_control) . . . . . real ⇝ *real*
  Länge des für die Tangentenbestimmung zu überspringenden Konturenstücks.
  Defaultwert: 0.0
  Typischer Wertebereich: 0.0 ≤ FitClippingLength
- ▷ **FitLength** (input_control) . . . . . real ⇝ *real* / string
  Länge des für die Tangentenbestimmung zu benutzenden Konturenstücks.
  Defaultwert: 30.0
  Wertevorschläge: FitLength ∈ {10.0, 20.0, 30.0, 'auto'}
  Typischer Wertebereich: 0.0 ≤ FitLength
- ▷ **MaxTangAngle** (input_control) . . . . . angle.rad ⇝ *real*
  Maximale Richtungsdifferenz der Tangenten.
  Defaultwert: 0.78539816
  Typischer Wertebereich: 0.0 ≤ MaxTangAngle ≤ 3.1415926

▷ **MaxDist** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand der Endpunkte.
Defaultwert: 25.0
Wertevorschläge: `MaxDist` ∈ {5.0, 10.0, 25.0, 50.0}
Typischer Wertebereich: 0.0 ≤ MaxDist

▷ **MaxDistPerp** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand der Endpunkte senkrecht zu den Tangenten.
Defaultwert: 10.0
Wertevorschläge: `MaxDistPerp` ∈ {2.0, 5.0, 10.0, 20.0}
Typischer Wertebereich: 0.0 ≤ MaxDistPerp

▷ **MaxOverlap** (input_control) . . . . . real ⇝ *real*
Maximale Überlappung zweier Konturen.
Defaultwert: 2.0
Wertevorschläge: `MaxOverlap` ∈ {2.0, 5.0, 10.0, 20.0}

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Modus für die Behandlung der Konturattribute
Defaultwert: 'attr_forget'
Werteliste: `Mode` ∈ {'attr_keep', 'attr_forget'}

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger
`split_contours_xld`, `select_contours_xld`

Alternativen
`union_collinear_contours_xld`, `union_collinear_contours_ext_xld`, `union_cocircular_contours_xld`, `union_straight_contours_xld`, `union_adjacent_contours_xld`

Siehe auch
`edges_sub_pix`, `threshold_sub_pix`, `gen_polygons_xld`, `get_contour_xld`, `get_contour_attrib_xld`

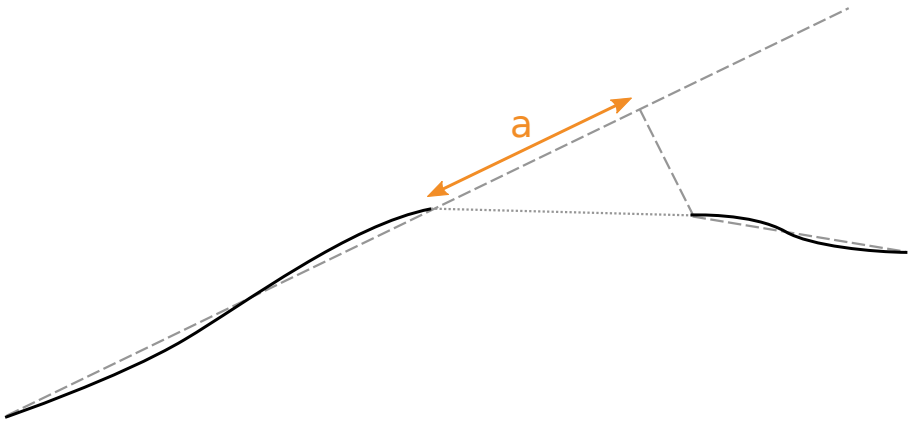
Modul
Foundation

union_straight_contours_xld (Contours : UnionContours : MaxDist, MaxDiff, Percent, Mode, Iterations :)

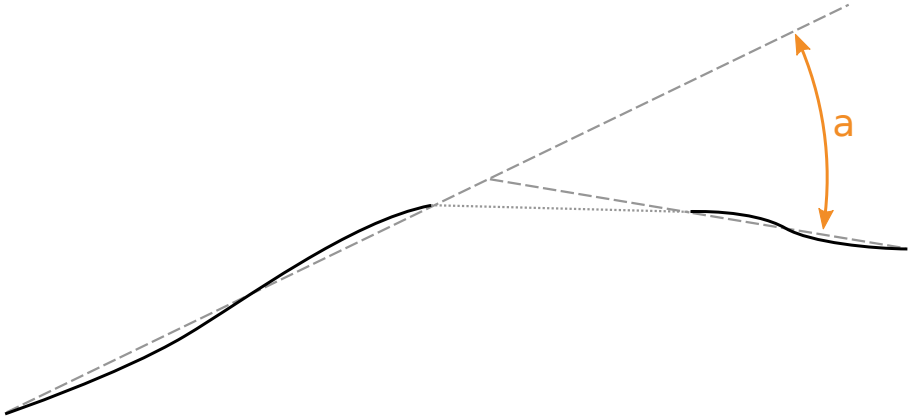
Vereinigen von benachbarten geraden Konturen mit ähnlicher Richtung.

Mit `union_straight_contours_xld` werden die Konturen `Contours` verglichen und unter gewissen Voraussetzungen vereinigt. In einem Iterationsschritt werden maximal zwei Konturen, die die entsprechenden Voraussetzungen erfüllen, vereinigt. Mit dem Parameter `Iterations` kann gesteuert werden, wie oft dieser Vereinigungsschritt wiederholt wird.

Zwei Konturen werden vereinigt, wenn deren kürzester Endpunktabstand (die Endpunkte sind die Fußpunkte der Lote von den Konturendpunkten auf die optimale Regressionsgerade) kleiner als `MaxDist` ist und wenn ihre Richtungsdifferenz (bzgl. der zugehörigen optimalen Regressionsgeraden) kleiner als `MaxDiff` (Bogenmaß) ist.



$a \leq$ MaxDist



$a \leq$ MaxDiff

Wenn nur eine der beiden Bedingungen zutrifft, kann die Entscheidung zugunsten einer Vereinigung durch die Gewichtung Percent beeinflusst werden, indem die prozentuale Überschreitung des angegebenen Grenzwertes gegen die prozentuale Unterschreitung des anderen Grenzwertes angerechnet wird. Dabei wird der Endpunktabstand mit Percent, die Richtungsdifferenz mit $100 -$ Percent bewertet.

Das bedeutet, dass zwei Konturen dann vereinigt werden, wenn sie die folgende Bedingung erfüllen:

$\frac{shortest\ distance}{\texttt{MaxDist}} \cdot \texttt{Percent} + \frac{direction\ difference}{\texttt{MaxDiff}} \cdot (100 - \texttt{Percent}) \leq 100$

Wenn zwei Konturen beispielsweise einen minimalen Endpunktabstand von 5.0 und eine Richtungsdifferenz von 0.5 haben (bei Grenzwerten MaxDist = *4.0* und MaxDiff = *0.625*), so weichen beide Werte von den Grenzwerten um 25% ab. Durch die Wahl von Percent = *60%* fällt nun der große Endpunktabstand stärker ins Gewicht als die geringe Richtungsdifferenz, weshalb die Konturen nicht vereinigt werden. Wenn in diesem Fall Percent = *40%* gewählt würde, würden die Konturen vereinigt werden.

Bei Percent = *100%* geht nur der Endpunktabstand in die Entscheidung ein, bei Percent = *0%* geht nur die Richtungsdifferenz in die Entscheidung ein, bei Percent = *50%* sind beide Merkmale gleichberechtigt.

Bei parallel verlaufenden Konturen besteht die Gefahr, dass nebeneinander liegende Konturen vereinigt werden. Soll dieser Effekt verhindert werden, so muss für Mode *'noparallel'*, ansonsten *'paralleltoo'* angegeben werden. Für *'every'* werden die Konturen bedingungslos vereinigt. Alle anderen Parameter haben in diesem Fall keinen Einfluss.

Für jede durch Vereinigung neu entstandene Kontur werden die Regressionsgeradenparameter neu berechnet.

Achtung

Bevor Konturen mit union_straight_contours_xld vereinigt werden können, müssen die Regressionsgeradenparameter mit regress_contours_xld berechnet werden.

Außerdem ist zu beachten, dass union_straight_contours_xld keine Konturen vereinigen kann, deren Punkte unterschiedliche Attribute aufweisen. Dieser Fall kann eintreten, wenn Konturen mit verschiedenen Operatoren extrahiert werden (z.B. mit lines_gauss und lines_facet). Informationen zu den Konturattributen und in welchen Fällen diese definiert werden, können in den jeweiligen Operatorenreferenzen gefunden werden. Die Punktattribute einer Kontur können mit query_contour_attribs_xld abgefragt werden.

Parameter

▷ **Contours** (input_object) . . . . . xld_cont-array ⇝ *object*
Eingabe-Konturen.

▷ **UnionContours** (output_object) . . . . . xld_cont-array ⇝ *object*
Ausgabe-Konturen.

▷ **MaxDist** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand der Endpunkte.
Defaultwert: 5.0

▷ **MaxDiff** (input_control) . . . . . angle.rad ⇝ *real*
Maximale Richtungsdifferenz.
Defaultwert: 0.5

▷ **Percent** (input_control) . . . . . real ⇝ *real*
Gewichtungsfaktor für die zwei Selektionskriterien.
Defaultwert: 50.0

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Berücksichtigung paralleler Konturen.
Defaultwert: 'noparallel'
Werteliste: `Mode` ∈ {'noparallel', 'paralleltoo', 'every'}

▷ **Iterations** (input_control) . . . . . string ⇝ *string* / integer
Anzahl Iterationen oder 'maximum'.
Defaultwert: 'maximum'
Wertevorschläge: `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'maximum'}
Typischer Wertebereich: 1 ≤ `Iterations` ≤ 500 (lin)
Minimale Schrittweite: 1
Empfohlene Schrittweite: 1

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`regress_contours_xld`

Alternativen

`union_collinear_contours_xld`, `union_collinear_contours_ext_xld`, `union_cocircular_contours_xld`, `union_cotangential_contours_xld`, `union_adjacent_contours_xld`

Siehe auch

`fit_line_contour_xld`, `get_contour_xld`, `get_contour_attrib_xld`, `gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `get_regress_params_xld`, `get_contour_global_attrib_xld`, `query_contour_global_attribs_xld`

Modul

Foundation

29.6 Zugriff

get_contour_xld (Contour : : : Row, Col)

Auslesen der Koordinaten einer XLD-Kontur.

`get_contour_xld` liefert für die XLD-Kontur `Contour` folgende Werte:

`Row`	Zeilen-Koordinate des Konturpunktes
`Col`	Spalten-Koordinate des Konturpunktes

Parameter

▷ **Contour** (input_object) . . . . . xld_cont ⇝ *object*
Eingabe-Kontur.

▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
Zeilen-Koordinate des Konturpunktes.

▷ **Col** (output_control) . . . . . point.x-array ⇝ *real*
Spalten-Koordinate des Konturpunktes.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Siehe auch

`get_contour_attrib_xld`, `query_contour_attribs_xld`, `get_contour_global_attrib_xld`, `query_contour_global_attribs_xld`, `gen_contour_polygon_xld`

Modul

Foundation

get_lines_xld (Polygon : : : BeginRow, BeginCol, EndRow, EndCol, Length, Phi)

Ausgeben aller Polygonseiten als Linien inkl. Länge und Orientierung.

`get_lines_xld` gibt alle Eingabepolygone `Polygon` als Linien aus. Im einzelnen werden zurückgegeben:

`BeginRow`:	Zeilen-Koordinate des Anfangspunktes
`BeginCol`:	Spalten-Koordinate des Anfangspunktes
`EndRow`:	Zeilen-Koordinate des Endpunktes
`EndCol`:	Spalten-Koordinate des Endpunktes
`Length`:	Seitenlänge
`Phi`:	Seitenorientierung

Parameter

▷ **Polygon** (input_object) . . . . . xld_poly(-array) ⇝ *object*
Polygone.

▷ **BeginRow** (output_control) . . . . . line.begin.y-array ⇝ *real*
Zeilen-Koordinate des Anfangspunktes.

▷ **BeginCol** (output_control) . . . . . line.begin.x-array ⇝ *real*
Spalten-Koordinate des Anfangspunktes.

▷ **EndRow** (output_control) . . . . . line.end.y-array ⇝ *real*
Zeilen-Koordinate des Endpunktes.

▷ **EndCol** (output_control) . . . . . line.end.x-array ⇝ *real*
Spalten-Koordinate des Endpunktes.

▷ **Length** (output_control) . . . . . real-array ⇝ *real*
Seitenlänge.

▷ **Phi** (output_control) . . . . . angle.rad-array ⇝ *real*
Seitenorientierung.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Automatisch parallelisiert auf Tupelebene.

Vorgänger

`gen_polygons_xld`

Alternativen

`get_polygon_xld`

Modul

Foundation

get_parallels_xld (Parallels : : : Row1, Col1, Length1, Phi1, Row2, Col2, Length2, Phi2)

Ausgeben der parallelen Polygonseiten inkl. Länge und Orientierung.

`get_parallels_xld` liefert für das erste Element aus `Parallels` (d.h. das erste parallele Polygon-Paar) die folgenden Werte:

`Row1`:	Zeilen-Koordinate des Polygonpunktes auf Polygon P1
`Col1`:	Spalten-Koordinate des Polygonpunktes auf Polygon P1
`Length1`:	Seitenlänge auf Polygon P1
`Phi2`:	Seitenorientierung auf Polygon P1
`Row2`:	Zeilen-Koordinate des Polygonpunktes auf Polygon P2
`Col2`:	Spalten-Koordinate des Polygonpunktes auf Polygon P2
`Length2`:	Seitenlänge auf Polygon P2
`Phi2`:	Seitenorientierung auf Polygon P2

Parameter

▷ **Parallels** (input_object) . . . . . xld ⇝ *object*
Parallelen.

▷ **Row1** (output_control) . . . . . polygon.y-array ⇝ *real*
Zeilen-Koordinate des Polygonpunktes auf Polygon P1.

▷ **Col1** (output_control) . . . . . polygon.x-array ⇝ *real*
Spalten-Koordinate des Polygonpunktes auf Polygon P1.

▷ **Length1** (output_control) . . . . . real-array ⇝ *real*
Seitenlänge auf Polygon P1.

▷ **Phi1** (output_control) . . . . . angle.rad-array ⇝ *real*
Seitenorientierung auf Polygon P1.

▷ **Row2** (output_control) . . . . . polygon.y-array ⇝ *real*
Zeilen-Koordinate des Polygonpunktes auf Polygon P2.

▷ **Col2** (output_control) . . . . . polygon.x-array ⇝ *real*
Spalten-Koordinate des Polygonpunktes auf Polygon P2.

▷ **Length2** (output_control) . . . . . real-array ⇝ *real*
Seitenlänge auf Polygon P2.

▷ **Phi2** (output_control) . . . . . angle.rad-array ⇝ *real*
Seitenorientierung auf Polygon P2.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_parallels_xld

Siehe auch

get_polygon_xld, get_lines_xld

Modul

Foundation

get_polygon_xld (Polygon : : : Row, Col, Length, Phi)

Auslesen der Daten eines XLD-Polygons.

get_polygon_xld liefert für das XLD-Polygon Polygon folgende Werte:

Row	Zeilen-Koordinate des Polygonpunktes
Col	Spalten-Koordinate des Polygonpunktes
Length	Länge der Polygonseite zwischen Punkt i und $i+1$
Phi	Winkel der Polygonseite zwischen Punkt i und $i+1$

Parameter

- ▷ **Polygon** (input_object) ..... xld_poly ⇝ *object*
  Eingabe-Polygon.
- ▷ **Row** (output_control) ..... point.y-array ⇝ *real*
  Zeilen-Koordinate des Polygonpunktes.
- ▷ **Col** (output_control) ..... point.x-array ⇝ *real*
  Spalten-Koordinate des Polygonpunktes.
- ▷ **Length** (output_control) ..... real-array ⇝ *real*
  Länge der Polygonseite.
- ▷ **Phi** (output_control) ..... angle.rad-array ⇝ *real*
  Winkel der Polygonseite.

Ausführungsinformationen

- Multithreading-Typ: reentrant (läuft parallel zu nicht-exklusiven Operatoren).
- Multithreading-Bereich: global (kann von jedem Thread aufgerufen werden).
- Wird ohne Parallelisierung verarbeitet.

Vorgänger

gen_polygons_xld

Alternativen

get_lines_xld

Modul

Foundation

Index

ähnliches Wort in Lexikon finden, 2160
über einen Schlüssel referenziertes Objekt eines Handles, 2926
über einen Schlüssel referenziertes Tupel eines Handles, 2928
1D-Beispielfunktion, 2633
1D-Funktion einlesen, 2632
1D-Funktion erzeugen, 2624, 2625
1D-Funktion glätten, 2634, 2635
1D-Funktion invertieren, 2629
1D-Funktion schreiben, 2637
1D-Funktion transformieren, 2636
1D-Funktionen matchen, 2630
1D-Funktionen zusammenstellen, 2623
2D-Messtechnik, 34, 36, 39, 42, 44, 46, 49, 51–60, 62, 64–69, 71, 73, 76
2D-Punkt auf Ellipse, 2663
2D-Punkt projizieren, 2771
2D-Punkte auf 2D-Linie projizieren, 2674
2D-Ähnlichkeitstransformation aus Punkt-zu-Gerade-Korrespondenzen, 2767
2D-Ähnlichkeitstransformation aus Punktkorrespondenzen, 2783
3D-Alignment, 1801, 1824, 2823, 2825
3D-Distanz aus binokularer Stereo-Disparität, 261
3D-Distanz mittels Fokusbildern messen, 299
3D-Distanzbild mittels Mehrgitter-Stereo messen, 256
3D-Distanzbild mittels korrelationsbasiertem Stereo messen, 252
3D-Distanzbild mittels photometrischer Stereomessung, 329, 333
3D-Formmodell erzeugen, 1856
3D-Formmodell finden, 1835, 1865, 1897, 1918
3D-Formmodell in Pixelkoordinaten transformieren, 115
3D-Formmodell löschen, 1836
3D-Formmodell wiederverwenden, 1835, 1865, 1918, 1938
3D-Höhenmodell mittels photometrischer Stereomessung, 334, 336, 337
3D-Information mittels Lichtschnitt rekonstruieren, 341, 355, 359
3D-Information mittels Mehrbild-Stereo rekonstruieren, 309, 311
3D-Information mittels binokularem Stereo rekonstruieren, 252, 256, 258, 260, 261, 263, 272
3D-Koordinaten aus 3D-Objektmodell berechnen, 208
3D-Koordinaten aus Stereobildern berechnen, 309
3D-Koordinaten aus binokularer Stereodisparität, 260, 263
3D-Koordinaten mittels Lichtschnittverfahren messen, 353, 354
3D-Koordinatensystem eines Kameraaufbaumodells setzen, 1454
3D-Lagetyp umrechnen, 2817
3D-Linie mit dualer Quaternion transformieren, 2813
3D-Matching-Ergebnis erhalten (deformierbar oberflächenbasiert), 92
3D-Matching-Ergebnis erhalten (oberflächenbasiert), 135
3D-Matching-Ergebnis löschen (deformierbar oberflächenbasiert), 84
3D-Matching-Ergebnis löschen (oberflächenbasiert), 119
3D-Matching-Modell (deformierbar oberflächenbasiert) schreiben, 97
3D-Matching-Modell (deformierbar oberflächenbasiert) wiederverwenden, 94, 97
3D-Matching-Modell (formbasiert) einlesen, 117
3D-Matching-Modell (formbasiert) schreiben, 119
3D-Matching-Modell (formbasiert) wiederverwenden, 117, 119
3D-Matching-Modell (oberflächenbasiert) einlesen, 139
3D-Matching-Modell (oberflächenbasiert) schreiben, 148
3D-Matching-Modell (oberflächenbasiert) wiederverwenden, 139, 148
3D-Matching-Modell einlesen (deformierbar oberflächenbasiert), 94
3D-Matching-Modell erzeugen (deformierbar oberflächenbasiert), 85
3D-Matching-Modell erzeugen (formbasiert), 100
3D-Matching-Modell erzeugen (oberflächenbasiert), 121
3D-Matching-Modell finden (deformierbar oberflächenbasiert), 89
3D-Matching-Modell finden (formbasiert), 107
3D-Matching-Modell finden (oberflächenbasiert), 125, 134, 218
3D-Matching-Modell löschen (deformierbar oberflächenbasiert), 85
3D-Matching-Modell löschen (formbasiert), 98
3D-Matching-Modell löschen (oberflächenbasiert), 120
3D-Matching-Modell prüfen (deformierbar oberflä-

chenbasiert), 93
3D-Matching-Modell prüfen (formbasiert), 112, 113
3D-Matching-Modell prüfen (oberflächenbasiert), 137
3D-Modell eines Objekts löschen, 149
3D-Modell eines Objekts prüfen, 177
3D-Modell eines Objekts segmentieren, 194, 195, 198
3D-Modell eines Objekts vorbereiten, 210
3D-Oberfläche aus Stereo-Bildern, 311
3D-Objekt-Pose in 3D-Formodellkoordinaten transformieren, 118
3D-Objektmodell, 152–157, 162–165, 168, 171, 182–184, 186, 187, 200, 202, 215, 221, 224, 226, 228, 230, 1048
3D-Objektmodell abtasten, 222
3D-Objektmodell anzeigen, 1029
3D-Objektmodell auf Punkte innerhalb einer Ansicht reduzieren, 192
3D-Objektmodell aus 3D-Koordinaten erzeugen, 239
3D-Objektmodell einlesen, 158
3D-Objektmodell in Pixelkoordinaten transformieren, 213
3D-Objektmodell kopieren, 149
3D-Objektmodell rendern, 220
3D-Objektmodell schreiben, 170
3D-Objektmodell vereinfachen, 224
3D-Objektmodelle fusionieren, 202
3D-Pose aus Punktkorrespondenzen, 2836
3D-Pose der Kamera für 3D-Matching erzeugen, 98
3D-Pose eines Kreises, 2823
3D-Pose eines Rechtecks, 2825
3D-Pose einlesen, 2834
3D-Pose erzeugen, 2818
3D-Pose in Quaternion umrechnen, 2832
3D-Pose in duale Quaternion umrechnen, 2831
3D-Pose in homogene 3D-Matrix umrechnen, 2800
3D-Pose schreiben, 2839
3D-Pose verschieben, 2835
3D-Pose-Typ, 2824
3D-Posen verbinden, 2829
3D-Primitive zu 3D Objektmodell anpassen, 190
3D-Punkt aus Sichtlinien messen, 272
3D-Punkt in Pixelkoordinaten transformieren, 1460
3D-Punkt mit Quaternion drehen, 2844
3D-Punkt projizieren, 1461, 1462, 2801, 2802
3D-Punkt von kartesischen Koordinaten in sphärische Koordinaten transformieren, 2846
3D-Punkt von sphärischen Koordinaten in kartesische Koordinaten transformieren, 2848
3D-Szene anzeigen, 1005
3D-Szene erzeugen, 1003
3D-Szene rendern, 1009
3D-Ähnlichkeitstransformation von Punktkorrespondenzen, 2804

Abbildung für Gitterrektifizierung, 2683
Abbildung von Bildkoordinaten in 3D-Koordinaten, 1469
Abbildung zur Rektifizierung einer beliebingen Verzerrung, 2682
Abbildung zur Rektifizierung von Stereo-Bildern, 266, 269
Abbildungstyp umrechnen, 807
abfragen der Parameter eines Klassifikators (Deep Learning), 1607
abfragen der Resultate eines Klassifikators (Deep Learning), 1609
abfragen der Trainingsresultate eines Klassifikators (Deep Learning), 1610
Ableitungen von 1D-Funktion, 2626
Ableitungsfilter, 758
`abs_diff_image`, 721
`abs_funct_1d`, 2623
`abs_image`, 722
`abs_matrix`, 2001
`abs_matrix_mod`, 2002
absolute Werte von Matrixelementen, 2001, 2002
absoluter Wert der Tupel-Elemente, 2851, 2864
absoluter Wert des Bildes, 722
absoluter Wert von 1D-Funktion, 2623
Abstand von 1D-Funktionen, 2626
Abstand von Regionen, 2656
Abstand von XLD-Kontur zu Ellipse, 3008, 3009
Abstand von XLD-Kontur zu Rechteck, 3011
Abstand zwischen 2D-Gerade und Region, 2651
Abstand zwischen 2D-Punkt und 2D-Strecke, 2655
Abstand zwischen 2D-Punkt und Region, 2654
Abstand zwischen 2D-Punkt und XLD-Kontur, 2652
Abstand zwischen 2D-Punkten, 2653
Abstand zwischen 2D-Punkten und 2D-Gerade, 2652
Abstand zwischen 2D-Strecke und 2D-Gerade, 2658
Abstand zwischen 2D-Strecke und Region, 2660
Abstand zwischen 2D-Strecke und XLD-Kontur, 2658
Abstand zwischen 2D-Strecken, 2661
Abstand zwischen XLD-Kontur und 2D-Gerade, 2650
Abstand zwischen XLD-Konturen, 2646
Abstandsbild des Schattens, 338
Abweichung von Grauwert-Rampe, 473
`access_channel`, 427
Achsenparalleles Rechteck als Zeichenobjekt erzeugen, 1085
`acos_image`, 723
`activate_compute_device`, 2493
`adapt_shape_model_high_noise`, 1835
`adapt_template`, 1665
`add_channels`, 389
`add_class_train_data_gmm`, 1489
`add_class_train_data_knn`, 1511
`add_class_train_data_mlp`, 1533
`add_class_train_data_svm`, 1573
`add_deformable_surface_model_reference_point`, 82

add_deformable_surface_model_sample, 83
add_dl_pruning_batch, 627
add_image, 723
add_matrix, 2002
add_matrix_mod, 2003
add_metrology_object_circle_measure, 34
add_metrology_object_ellipse_measure, 36
add_metrology_object_generic, 39
add_metrology_object_line_measure, 42
add_metrology_object_rectangle2_measure, 44
add_noise_distribution, 953
add_noise_white, 954
add_noise_white_contour_xld, 3057
add_sample_class_gmm, 1490
add_sample_class_knn, 1512
add_sample_class_mlp, 1534
add_sample_class_svm, 1574
add_sample_class_train_data, 1564
add_sample_identifier_preparation_data, 1266
add_sample_identifier_training_data, 1268
add_samples_image_class_gmm, 2397
add_samples_image_class_knn, 2398
add_samples_image_class_mlp, 2399
add_samples_image_class_svm, 2400
add_scene_3d_camera, 999
add_scene_3d_instance, 1000
add_scene_3d_label, 1000
add_scene_3d_light, 1002
add_texture_inspection_model_image, 1324
adjust_mosaic_images, 2710
affine 2D-Transformation aus Punkt-zu-Gerade-Korrespondenzen, 2767
affine 2D-Transformation einer Region, 2278
affine 2D-Transformation einer XLD-Kontur, 2985
affine 2D-Transformation eines Bildes, 803, 805
affine 2D-Transformation eines Pixels, 2742
affine 2D-Transformation eines Polygons, 2986
affine 2D-Transformation eines Punktes, 2743
affine 2D-Transformation eines Verschiebungsvektorfeldes, 2774
affine 2D-Transformation von Punktkorrespondenzen, 2776
affine 3D-Matrix, 2763
affine 3D-Transformation eines 3D-Objekt-Modells, 197
affine 3D-Transformation eines Punkts, 2785
affine 3D-Transformation von Punktkorrespondenzen, 2804
affine Transformation, 786
affine_trans_contour_xld, 2985
affine_trans_image, 803
affine_trans_image_size, 805
affine_trans_object_model_3d, 197
affine_trans_pixel, 2742
affine_trans_point_2d, 2743
affine_trans_point_3d, 2785
affine_trans_polygon_xld, 2986
affine_trans_region, 2278
Aktion auf I/O-Übertragungskanälen ausführen, 2513
Aktion auf I/O-Geräteinstanz ausführen, 2513
Aktion auf I/O-Schnittstelle ausführen, 2514
Aktualisierungsmodus für Ausführungsmarke konfigurieren, 715
Aktualisierungsmodus für Fenster konfigurieren, 716
Aktualisierungsmodus für Variablen konfigurieren, 716
Aktualisierungsmodus für Zeitmessung konfigurieren, 715
aktuelle Farbe, 1105, 1106
aktuelle Farbnachschlagetabelle (Look-Up-Tabelle, LUT), 1631
aktuelles Arbeitsverzeichnis, 508
aktuelles Arbeitsverzeichnis setzen, 510
aktuelles Grafikfenster, 690
aktuelles Grafikfenster setzen, 712
align_metrology_model, 46
allgemeine Filtermaske, 957
alphanumerische Barcode-Ergebnisse, 1233
alphanumerisches Barcode-Ergebnis, 1189
Alternative mit Bedingung, 2469, 2473
Alternative zu Bedingung, 2472, 2473
an Barriere warten (Multithreading), 2563
angezeigtes Bildteil setzen, 708, 1125
angle_ll, 2638
angle_lx, 2640
anisotrope 2D-Ähnlichkeitstransformation aus Punkt-zu-Gerade-Korrespondenzen, 2767
anisotrope 2D-Ähnlichkeitstransformation aus Punktkorrespondenzen, 2775
anisotrope Diffusion, 867
anisotropic_diffusion, 827
Anomaliemodell trainieren (Deep Learning), 556
Anzahl der Einträge in der HALCON-Datenbank, 2502
Anzahl der Parameter, 2530
Anzahl der Stützstellen einer 1D-Funktion, 2632
Anzahl der Support-Vektoren eines OCR-Klassifikators, 2219
Anzahl der Trainingsmuster, 1568
Anzahl der Trainingsmuster (GMM), 1502
Anzahl der Trainingsmuster (kNN), 1518
Anzahl der Trainingsmuster (MLP), 1549
Anzahl der Trainingsmuster (SVM), 1587
Anzahl von Support-Vektoren, 1589
Anzeige einer 3D-Pose ändern, 1065
Anzeigeform von Regionen setzen, 710, 1129
Anzeigemodus, 1629
Anzeigemodus für Grauwert, 1103
Anzeigemodus für Grauwerte setzen, 707, 1122

Anzeigemodus für Region, 1107
Anzeigemodus setzen, 1646
Anzeigemodus von Regionen setzen, 705, 1117
anzeigen, 684, 705–708, 1040, 1113, 1115, 1117, 1121, 1122, 1125, 1126, 1132, 1140–1142
append_channel, 428
append_ocr_trainf, 2229
apply_bead_inspection_model, 1289
apply_color_trans_lut, 781
apply_deep_ocr, 2133
apply_distance_transform_xld, 2641
apply_dl_classifier, 1603
apply_dl_model, 628
apply_metrology_model, 49
apply_sample_identifier, 1270
apply_sheet_of_light_calibration, 341
apply_texture_inspection_model, 1325
approx_chain, 1733
approx_chain_simple, 1736
Approximationsfehler für Konturenanzeige setzen, 1647
ARC/INFO-Datei, 517, 519, 522, 525
ARC/INFO-Weltdatei, 509
area_center, 2300
area_center_gray, 448
area_center_points_xld, 3002
area_center_xld, 3003
area_holes, 2302
area_intersection_rectangle2, 2642
area_object_model_3d, 171
Areakosinus hyperbolicus von Tupel-Elementen, 2852
Areasinus hyperbolicus von Tupel-Elementen, 2854
Areatangens hyperbolicus von Tupel-Elementen, 2856
Arkuskosinus eines Bildes, 723
Arkuskosinus von Tupel-Elementen, 2851
Arkussinus eines Bildes, 725
Arkussinus von Tupel-Elementen, 2853
Arkustangens eines Bildes, 726
Arkustangens von Tupel-Elementen, 2855
Arkustangens von zwei Bildern, 725
asin_image, 725
assign, 2467
assign_at, 2468
atan2_image, 725
atan_image, 726
attach_background_to_window, 1076
attach_drawing_object_to_window, 1077
Attribute einer XLD-Kontur, 3027, 3030
auf Condition warten (Multithreading), 2561, 2564
auf Event warten (Multithreading), 2562, 2564
auf externe Bilder zugreifen, 404, 405, 409, 410
auf Mausklick warten, 1067, 1068
auf projektive 2D-Matrix projizieren, 2763
auf XLD-Konturen zugreifen und sie umrechnen, 445, 3002, 3008, 3012, 3014, 3027, 3030, 3039, 3040, 3042, 3044, 3088
Aufmerksamkeitsbereich (ROI) entfernen, 391
Aufmerksamkeitsbereich (ROI) erzeugen, 389, 390, 392, 393
aus Datenbank löschen, 2242
Ausführung anhalten, 2483
Ausführung mit Ausnahme starten, 2486
Ausgabemodus für Grauwerte, 1628
Ausgabemodus für Grauwerte setzen, 1644
Ausmaße einer Schriftart (Font), 1135
Ausnahme werfen, 2485
Ausnahmedaten, 687
Ausnahmen abfangen, 2469
Ausrichtung (Orientierung) einer Tangente einer XLD-Kontur, 3026
Ausrichtung einer Region, 2327
Ausrichtung einer XLD-Kontur, 3044
auto_threshold, 2426
automatische Operator-Parallelisierung optimieren, 2567
automatische Operatorenparallelisierungsdaten schreiben, 2573
axis_angle_to_quat, 2840

Bündelausgleichsmosaik, 2716
background_seg, 2351
Bandfilter, 756
Bandpass-Filter, 757, 768, 769
bandpass_image, 916
Barcode lesen, 1179, 1181, 1185, 1189
Barcode-Druckqualität prüfen, 1189
Barcode-Ergebnisobjekt, 1185
Barcode-Ergebnisse prüfen, 1185, 1189
Barcode-Modell anpassen, 1200, 1209
Barcode-Modell einlesen, 1199
Barcode-Modell erzeugen, 1177, 1200, 1209
Barcode-Modell löschen, 1177
Barcode-Modell optimieren, 1186, 1188, 1197
Barcode-Modell schreiben, 1212
Barcode-Modell trainieren, 1179, 1181
Barriere erzeugen (Multithreading), 2534, 2538
Bassins extrahieren, 2462, 2463, 2465
Beleuchtung schätzen, 326
Beobachtung aus dem Kalibrierdatenmodell entfernen, 1442
Beobachtungs-Indizes eines Kamerakalibrierdatenmodells, 1439
Beobachtungs-Konturen eines Kamerakalibrierdatenmodells, 1436
Beobachtungs-Pose eines Kamerakalibrierdatenmodells, 1385
Beobachtungs-Pose eines Kamerakalibrierdatenmodells setzen, 1391
Beobachtungspunkte der Kamerakalibrierung, 1437
best_match, 1666
best_match_mg, 1667
best_match_pre_mg, 1669
best_match_rot, 1670
best_match_rot_mg, 1671

Betriebssystem, 2935
bilateral_filter, 829
Bilaterale Filterung eines Bildes, 829
Bild (-kanäle) für Farbverarbeitung bearbeiten, 884, 917
Bild an Socket senden, 2616
Bild anzeigen, 686, 1020, 1028
Bild aus schneller Fourier-Transformation (FFT) falten, 748, 749
Bild beleuchten, 982
Bild bitweise nach rechts verschieben (Rechts-Shift), 745
Bild drehen, 818
Bild durch Abbildung transformieren, 809
Bild durch Regiongrowing segmentieren, 2385, 2387, 2388
Bild durch Wasserausschüttung segmentierung, 2459
Bild eines Barcodes vorverarbeiten, 730
Bild einlesen, 491, 504
Bild einziehen, 375, 377, 378, 380
Bild entzerren, 940
Bild erzeugen, 403–405, 407, 412, 414, 418–420
Bild für Blob-Analyse segmentieren, 464, 845, 2414, 2435, 2436, 2462, 2463, 2465
Bild für Blob-Analyse vorverarbeiten (Filterung), 833, 837, 839, 845, 849, 851, 856, 858, 859, 863
Bild für Kantenextraktion filtern (pixelgenau), 879, 883, 884, 889, 895, 900, 902, 905–909, 911, 912, 914
Bild für OCR vorverarbeiten (Filterung), 943, 2075, 2077, 2078, 2080, 2081, 2083, 2084, 2086, 2088
Bild für optische Zeichenerkennung (OCR) segmentieren, 1716, 2189, 2213
Bild glätten, 984
Bild in 3D-Koordinaten transformieren, 1474
Bild in Polarkoordinaten transformieren, 811
Bild invertieren, 730
Bild mit 2D-Histogramm segmentieren, 2401, 2403
Bild mit Abweichungsschwellwert segmentieren, 2441
Bild mit anisotroper Diffusion glätten, 827
Bild mit automatischem Schwellwert segmentieren, 2426
Bild mit Bewegungsunschärfe erzeugen, 992
Bild mit Bewegungsunschärfe wiederherstellen, 993, 995
Bild mit binärem Schwellwert segmentieren, 2427
Bild mit Binomial-Filter glätten, 833
Bild mit Deriche-Filter glätten, 863
Bild mit Gauß-Filter glätten, 839, 863
Bild mit Hysterese-Schwellwert segmentieren, 2394
Bild mit isotroper Diffusion glätten, 844
Bild mit Kalibrierplatte erzeugen, 1412
Bild mit Linksverschiebung, 742
Bild mit lokalem Schwellwert segmentieren, 2433, 2437
Bild mit Median-Filter glätten, 849, 851, 853, 854
Bild mit Midrange-Filter glätten, 855
Bild mit Mittelwert-Filter glätten, 834, 836, 845, 847
Bild mit Mittlungsverfahren durch Mittelwert-Filter glätten, 847
Bild mit Nachschlagetabelle (Look-Up-Tabelle, LUT) transformieren, 964
Bild mit Rang-Filter glätten, 856, 858, 859, 864
Bild mit schnellem Schwellwert segmentieren, 2435
Bild mit Schwellwert für Zeichen segmentieren, 2429
Bild mit Schwellwert segmentieren, 2439
Bild mit Shen-Filter glätten, 863
Bild mit Sigma-Filter glätten, 861
Bild mit Subpixel-Schwellwert segmentieren, 2440
Bild mit Variationsmodell vergleichen, 1347, 1348
Bild pixelweise vergleichen, 2430
Bild radiometrisch kalibrieren, 964
Bild rektifizieren, 809, 1466, 1474
Bild segmentieren mit Pixelklassifikation (euklidisch), 2405
Bild segmentieren mit Pixelklassifikation (GMM), 2407
Bild segmentieren mit Pixelklassifikation (Hyperbox), 1728
Bild segmentieren mit Pixelklassifikation (LUT), 2409
Bild segmentieren mit Pixelklassifikation (MLP), 2410
Bild segmentieren mit Pixelklassifikation (SVM), 2412
Bild spiegeln, 810
Bild und vorverarbeitete Daten einziehen, 376
Bild verbessern, 981
Bild von Polarkoordinaten transformieren, 813
Bild von Socket einlesen, 2612
Bild zerlegen, 1726
Bildbereich vergrößern, 960
Bildbereiche interpolieren, 837
Bildeinzug in Echtzeit, 376, 378, 380
Bildeinzugsgerät öffnen, 381, 383
Bildeinzugsgerät schließen, 371
Bildeinzugsparameter, 374
Bilder addieren, 723
Bilder dividieren, 727
Bilder einer schnellen Fourier-Transformation (FFT) korrelieren, 750, 772
Bilder eines Variationsmodells, 1353
Bilder kacheln, 399, 400, 402
Bilder multiplizieren, 734
Bilder schreiben, 497
Bilder subtrahieren, 721, 738
Bilderzeugungszeit, 488
Bildgröße, 488
Bildgröße ändern (skalieren, Zoom), 819, 820
Bildgröße für Messmodell setzen, 68
Bildgröße setzen, 393–398
Bildkanäle, 434–440
Bildkanäle zählen, 434
Bildkanal, 427

Bildkoordinaten für 3D Anzeige, 1064
Bildmittelpunkt, 1423
Bildpaare des Stereo-Modells erhalten, 306
Bildpaare des Stereo-Modells setzen, 318
Bildpyramide, 928
Bildsequenz einlesen, 494
Bildtyp, 489
Bildtyp umrechnen, 478
bin_threshold, 1728
binäres Bild erzeugen, 424
binary_threshold, 2427
binocular_calibration, 1376
binocular_disparity, 241
binocular_disparity_mg, 244
binocular_disparity_ms, 250
binocular_distance, 252
binocular_distance_mg, 256
binocular_distance_ms, 258
binokulares Stereo-Kamerasystem kalibrieren, 269
Binomial-Filter, 833
binomial_filter, 833
bit_and, 741
bit_lshift, 742
bit_mask, 742
bit_not, 743
bit_or, 744
bit_rshift, 745
bit_slice, 746
bit_xor, 747
Bitextraktion, 746
bitweises Exklusiv-ODER von Bildern, 747
bitweises Exklusiv-ODER von Tupeln, 2881
bitweises Komplement eines Bildes, 743
bitweises NICHT von Tupeln, 2880
bitweises ODER von Bilder, 744
bitweises ODER von Tupeln, 2881
bitweises UND von Bild und Maske, 742
bitweises UND von Bilder, 741
bitweises UND von Tupeln, 2879
Blob-Analyse beschleunigen, 2435, 2439, 2508, 2580
Blob-Analyse mit hoher Genauigkeit, 448, 2300
Bogenmaß in Grad umrechnen, 2860
Bottom-Hat- und Top-Hat-Operator, 1697
Bottom-Hat-Operator, 2074, 2093
bottom_hat, 2093
boundary, 2094
Boxen in 3D-Daten finden, 79
break, 2468
Breite für Kantenfilter schätzen, 899
Breitenprofil einer Region, 2314
Brennweite, 1423
broadcast_condition, 2533
bundle_adjust_mosaic, 2713

CAD-Datei einlesen, 509
calibrate_cameras, 1423
calibrate_hand_eye, 1380
calibrate_sheet_of_light, 342
Callback-Funktion für Bildeinzugsgerät, 372
Callback-Funktion für Bildeinzugsgerät setzen, 386
Callback-Funktion für ein Zeichenobjekt setzen, 1094
caltab_points, 1393
cam_mat_to_cam_par, 1414
cam_par_pose_to_hom_mat3d, 1460
cam_par_to_cam_mat, 1415
camera_calibration, 1456
Canny-Filter, 884, 887, 889, 892
case, 2469
catch, 2469
Ceiling-Funktion von Tupel-Elementen, 2857
cfa_to_rgb, 782
change_domain, 390
change_format, 393
change_radial_distortion_cam_par, 1463
change_radial_distortion_contours_xld, 1465
change_radial_distortion_image, 1466
change_radial_distortion_points, 1467
channels_to_image, 428
char_threshold, 2429
check_difference, 2430
circularity, 2302
circularity_xld, 3004
class_2dim_sup, 2401
class_2dim_unsup, 2403
class_ndim_box, 1728
class_ndim_norm, 2405
classify_class_gmm, 1491
classify_class_knn, 1513
classify_class_mlp, 1535
classify_class_svm, 1575
classify_image_class_gmm, 2407
classify_image_class_knn, 2408
classify_image_class_lut, 2409
classify_image_class_mlp, 2410
classify_image_class_svm, 2412
clear_all_bar_code_models, 1738
clear_all_barriers, 1738
clear_all_calib_data, 1739
clear_all_camera_setup_models, 1739
clear_all_class_gmm, 1740
clear_all_class_knn, 1740
clear_all_class_lut, 1741
clear_all_class_mlp, 1741
clear_all_class_svm, 1742
clear_all_class_train_data, 1742
clear_all_color_trans_luts, 1742
clear_all_component_models, 1743
clear_all_conditions, 1743
clear_all_data_code_2d_models, 1744
clear_all_deformable_models, 1744
clear_all_descriptor_models, 1745
clear_all_events, 1745
clear_all_lexica, 1746

clear_all_matrices, 1746
clear_all_metrology_models, 1747
clear_all_mutexes, 1747
clear_all_ncc_models, 1747
clear_all_object_model_3d, 1748
clear_all_ocr_class_knn, 1748
clear_all_ocr_class_mlp, 1749
clear_all_ocr_class_svm, 1749
clear_all_sample_identifiers, 1750
clear_all_scattered_data_interpolators, 1750
clear_all_serialized_items, 1751
clear_all_shape_model_3d, 1751
clear_all_shape_models, 1752
clear_all_sheet_of_light_models, 1752
clear_all_stereo_models, 1753
clear_all_surface_matching_results, 1753
clear_all_surface_models, 1754
clear_all_templates, 1754
clear_all_text_models, 1754
clear_all_text_results, 1755
clear_all_training_components, 1755
clear_all_variation_models, 1756
clear_bar_code_model, 1177
clear_barrier, 2534
clear_bead_inspection_model, 1290
clear_calib_data, 1424
clear_camera_setup_model, 1424
clear_class_gmm, 1492
clear_class_knn, 1514
clear_class_lut, 1526
clear_class_mlp, 1536
clear_class_svm, 1576
clear_class_train_data, 1565
clear_color_trans_lut, 784
clear_component_model, 1939
clear_condition, 2534
clear_data_code_2d_model, 1215
clear_deformable_model, 1777
clear_deformable_surface_matching_result, 84
clear_deformable_surface_model, 85
clear_descriptor_model, 1818
clear_distance_transform_xld, 2643
clear_dl_classifier, 1605
clear_dl_classifier_result, 1605
clear_dl_classifier_train_result, 1606
clear_dl_model, 630
clear_drawing_object, 1078
clear_event, 2535
clear_handle, 2904
clear_lexicon, 2157
clear_matrix, 2039
clear_message, 2536
clear_message_queue, 2536
clear_metrology_model, 50
clear_metrology_object, 51
clear_mutex, 2537
clear_ncc_model, 1980
clear_obj, 2242
clear_object_model_3d, 149
clear_ocr_class_cnn, 2125
clear_ocr_class_knn, 2142
clear_ocr_class_mlp, 2161
clear_ocr_class_svm, 2207
clear_rectangle, 1624
clear_sample_identifier, 1272
clear_samples_class_gmm, 1492
clear_samples_class_mlp, 1536
clear_samples_class_svm, 1577
clear_sampset, 1652
clear_scattered_data_interpolator, 2703
clear_scene_3d, 1003
clear_serial, 2599
clear_serialized_item, 2595
clear_shape_model, 1836
clear_shape_model_3d, 98
clear_sheet_of_light_model, 345
clear_stereo_model, 304
clear_structured_light_model, 1303
clear_surface_matching_result, 119
clear_surface_model, 120
clear_template, 1673
clear_text_model, 2186
clear_text_result, 2187
clear_texture_inspection_model, 1327
clear_texture_inspection_result, 1328
clear_train_data_variation_model, 1346
clear_training_components, 1939
clear_variation_model, 1346
clear_window, 1042
clip_contours_xld, 3058
clip_end_points_contours_xld, 3059
clip_region, 2352
clip_region_rel, 2353
close_all_bg_esti, 1756
close_all_class_box, 1757
close_all_files, 1757
close_all_framegrabbers, 1758
close_all_measures, 1758
close_all_ocrs, 1759
close_all_ocvs, 1759
close_all_serials, 1760
close_all_sockets, 1760
close_bg_esti, 2685
close_class_box, 1653
close_contours_xld, 3059
close_edges, 877
close_edges_length, 878
close_file, 527
close_framegrabber, 371
close_io_channel, 2511

close_io_device, 2512
close_measure, 3
close_ocr, 1711
close_ocv, 1295
close_serial, 2600
close_socket, 2605
close_window, 1043
closest_point_transform, 2355
closing, 2096
Closing-Operator, 1683, 2075–2077, 2096, 2097, 2099
closing_circle, 2097
closing_golay, 1683
closing_rectangle1, 2099
Cluster-Modellkomponenten, 1940
cluster_model_components, 1940
coherence_enhancing_diff, 978
combine_roads_xld, 3060
comment, 2471
compactness, 2304
compactness_xld, 3005
compare_ext_variation_model, 1347
compare_obj, 2238
compare_variation_model, 1348
complement, 2289
complex_to_real, 478
compose2, 429
compose3, 429
compose4, 430
compose5, 431
compose6, 432
compose7, 433
compose_funct_1d, 2623
Compute Device öffnen, 2497
Compute Device aktivieren, 2493
Compute Device deaktivieren, 2494
Compute Device freigeben, 2499, 2500
Compute Device initialisieren, 2497
concat_obj, 2243
concat_ocr_trainf, 2230
Condition erzeugen (Multithreading), 2539
Condition löschen (Multithreading), 2534
Condition signalisieren (Multithreading), 2533, 2560
connect_and_holes, 2305
connect_grid_points, 2679
connection, 2357
connection_object_model_3d, 198
continue, 2471
contlength, 2305
contour_point_num_xld, 3005
contour_to_world_plane_xld, 1467
Control-Parameter aus einer Nachricht lesen (Inter-Thread Kommunikation), 2551
Control-Parameter in eine Nachricht schreiben (Inter-Thread Kommunikation), 2559
control_io_channel, 2513
control_io_device, 2513
control_io_interface, 2514
convert_coordinates_image_to_window, 1098
convert_coordinates_window_to_image, 1099
convert_image_type, 478
convert_map_type, 807
convert_point_3d_cart_to_spher, 2846
convert_point_3d_spher_to_cart, 2848
convert_pose_type, 2817
convert_tuple_to_vector_1d, 2471
convert_vector_to_tuple, 2472
convex_hull_object_model_3d, 200
convexity, 2307
convexity_xld, 3006
convol_fft, 748
convol_gabor, 749
convol_image, 957
cooc_feature_image, 449
cooc_feature_matrix, 450
Cooccurrence-Matrix, 449, 461
copy_dict, 2883
copy_file, 506
copy_image, 403
copy_matrix, 2040
copy_metrology_model, 52
copy_metrology_object, 1599
copy_obj, 2244
copy_object_model_3d, 149
copy_rectangle, 1044
corner_response, 941
correlation_fft, 750
cos_image, 726
count_channels, 434
count_obj, 2239
count_relation, 2502
count_seconds, 2491
create_aniso_shape_model, 1837
create_aniso_shape_model_xld, 1842
create_bar_code_model, 1177
create_barrier, 2538
create_bead_inspection_model, 1291
create_bg_esti, 2686
create_calib_data, 1425
create_calib_descriptor_model, 1819
create_caltab, 1395
create_cam_pose_look_at_point, 98
create_camera_setup_model, 1426
create_class_box, 1653
create_class_gmm, 1493
create_class_knn, 1515
create_class_lut_gmm, 1526
create_class_lut_knn, 1528
create_class_lut_mlp, 1530
create_class_lut_svm, 1531
create_class_mlp, 1537
create_class_svm, 1577
create_class_train_data, 1566
create_color_trans_lut, 784
create_component_model, 1942

create_condition, 2539
create_data_code_2d_model, 1216
create_deep_ocr, 2135
create_deformable_surface_model, 85
create_dict, 2884
create_distance_transform_xld, 2643
create_dl_layer_activation, 557
create_dl_layer_batch_normalization, 559
create_dl_layer_class_id_conversion, 562
create_dl_layer_concat, 565
create_dl_layer_convolution, 566
create_dl_layer_dense, 570
create_dl_layer_depth_max, 572
create_dl_layer_dropout, 573
create_dl_layer_elementwise, 575
create_dl_layer_input, 576
create_dl_layer_loss_cross_entropy, 579
create_dl_layer_loss_ctc, 581
create_dl_layer_loss_distance, 585
create_dl_layer_loss_focal, 587
create_dl_layer_loss_huber, 589
create_dl_layer_lrn, 591
create_dl_layer_pooling, 592
create_dl_layer_reshape, 595
create_dl_layer_softmax, 597
create_dl_layer_transposed_convolution, 598
create_dl_layer_zoom_factor, 601
create_dl_layer_zoom_size, 602
create_dl_layer_zoom_to_layer_size, 604
create_dl_model, 606
create_dl_model_detection, 668
create_dl_pruning, 630
create_drawing_object_circle, 1079
create_drawing_object_circle_sector, 1080
create_drawing_object_ellipse, 1081
create_drawing_object_ellipse_sector, 1082
create_drawing_object_line, 1084
create_drawing_object_rectangle1, 1085
create_drawing_object_rectangle2, 1086
create_drawing_object_text, 1087
create_drawing_object_xld, 1088
create_event, 2540
create_funct_1d_array, 2624
create_funct_1d_pairs, 2625
create_generic_shape_model, 1847
create_lexicon, 2157
create_local_deformable_model, 1777
create_local_deformable_model_xld, 1780
create_matrix, 2041
create_message, 2541
create_message_queue, 2542
create_metrology_model, 52
create_mutex, 2543
create_ncc_model, 1981
create_ocr_class_box, 1712
create_ocr_class_knn, 2142
create_ocr_class_mlp, 2161
create_ocr_class_svm, 2208
create_ocv_proj, 1296
create_planar_calib_deformable_model, 1782
create_planar_calib_deformable_model_xld, 1784
create_planar_uncalib_deformable_model, 1787
create_planar_uncalib_deformable_model_xld, 1791
create_pose, 2818
create_rectification_grid, 2680
create_sample_identifier, 1273
create_scaled_shape_model, 1847
create_scaled_shape_model_xld, 1852
create_scattered_data_interpolator, 2704
create_scene_3d, 1003
create_serialized_item_ptr, 2596
create_shape_model, 1856
create_shape_model_3d, 100
create_shape_model_xld, 1861
create_sheet_of_light_calib_object, 346
create_sheet_of_light_model, 347
create_stereo_model, 305
create_structured_light_model, 1304
create_surface_model, 121
create_template, 1673
create_template_rot, 1675
create_text_model, 1714
create_text_model_reader, 2187
create_texture_inspection_model, 1329
create_trained_component_model, 1946
create_uncalib_descriptor_model, 1820
create_variation_model, 1350
critical_points_sub_pix, 2448
crop_contours_xld, 3061
crop_domain, 394
crop_domain_rel, 395
crop_part, 396
crop_rectangle1, 397
crop_rectangle2, 398
Cursorform, 1632
Cursorform setzen, 1650
Cursorposition setzen, 1137, 1141

dargestellter Bildteil, 1104
darstellbare Grauwerte, 1110

Datacode lesen, 1220, 1227, 1233
Datacode prüfen, 1227, 1233
Datacode-Druckqualität prüfen, 1233
Datacode-Ergebnisobjekte, 1227
Datacode-Modell einlesen, 1220, 1251
Datacode-Modell erzeugen, 1216
Datacode-Modell löschen, 1215
Datacode-Modell optimieren, 1229, 1250, 1253
Datacode-Modell schreiben, 1262
Datacode-Modell trainieren, 1220, 1233, 1253
Datacode-Modell wiederverwenden, 1220, 1251, 1262
Datei aktualisieren, 1772
Datei einlesen, 1772
Datei kopieren, 506
Datei löschen, 507, 509, 510
Datei-Dialog öffnen, 692
Dateien in Verzeichnis auflisten, 508
Daten an Socket senden, 2615
Daten auf I/O-Übertragungskanäle schreiben, 2523
Daten aus automatischer Operator-Parallelisierung, 2566
Daten aus automatischer Operator-Parallelisierung einlesen, 2571
Daten aus automatischer Operator-Parallelisierung setzen, 2572
Daten aus dem Kalibrierdatenmodell entfernen, 1442
Daten deserialisieren, 3, 53, 88, 107, 124, 152, 350, 491, 501, 503, 511, 512, 516, 631, 1181, 1297, 1351, 1416, 1428, 1496, 1516, 1542, 1567, 1582, 1607, 1655, 1677, 1680, 1715, 1794, 1824, 1949, 2031, 2146, 2744, 2786, 2822
Daten klassifizieren, 2407–2410, 2412
Daten klassifizieren (GMM), 1491
Daten klassifizieren (Hyperbox), 1656, 1657, 1664
Daten klassifizieren (kNN), 1513
Daten klassifizieren (MLP), 1535
Daten klassifizieren (SVM), 1575, 1582
Daten serialisieren, 22, 67, 96, 117, 143, 164, 359, 497, 502, 505, 513, 521, 652, 1199, 1220, 1252, 1300, 1306, 1315, 1330, 1338, 1357, 1418, 1443, 1444, 1507, 1522, 1553, 1571, 1595, 1612, 1662, 1720, 1811, 1833, 1949, 1973, 1974, 1983, 1996, 2033, 2154, 2165, 2179, 2772, 2803, 2807, 2816, 2835, 2841, 2845
Daten von Socket einlesen, 2611
Datensatz (Hyperbox) löschen, 1652
deactivate_all_compute_devices, 2494
deactivate_compute_device, 2494
decode_bar_code_rectangle2, 1179
decode_structured_light_pattern, 1305
decompose2, 434
decompose3, 435
decompose4, 436
decompose5, 437
decompose6, 438
decompose7, 439
decompose_matrix, 2051
default, 2472
defokussiertes Bild erzeugen, 991
defokussiertes Bild wiederherstellen, 993, 995
delete_file, 507
Depth from Focus, 299
depth_from_focus, 299
dequeue_message, 2544
Deriche-Filter, 884, 887, 889, 892
derivate_funct_1d, 2626
derivate_gauss, 879
derivate_vector_field, 930
descript_class_box, 1654
deserialize_bar_code_model, 1181
deserialize_calib_data, 1428
deserialize_cam_par, 1416
deserialize_camera_setup_model, 1428
deserialize_class_box, 1655
deserialize_class_gmm, 1496
deserialize_class_knn, 1516
deserialize_class_mlp, 1542
deserialize_class_svm, 1582
deserialize_class_train_data, 1567
deserialize_component_model, 1949
deserialize_data_code_2d_model, 1220
deserialize_deformable_model, 1794
deserialize_deformable_surface_model, 88
deserialize_descriptor_model, 1824
deserialize_distance_transform_xld, 2645
deserialize_dl_classifier, 1607
deserialize_dl_model, 631
deserialize_dual_quat, 2807
deserialize_fft_optimization_data, 750
deserialize_handle, 511
deserialize_hom_mat2d, 2744
deserialize_hom_mat3d, 2786
deserialize_image, 491
deserialize_matrix, 2031
deserialize_measure, 3
deserialize_metrology_model, 53
deserialize_ncc_model, 1983
deserialize_object, 501
deserialize_object_model_3d, 152
deserialize_ocr, 1715
deserialize_ocr_class_cnn, 2125
deserialize_ocr_class_knn, 2146
deserialize_ocr_class_mlp, 2165
deserialize_ocr_class_svm, 2211
deserialize_ocv, 1297
deserialize_pose, 2822
deserialize_quat, 2841
deserialize_region, 503
deserialize_sample_identifier, 1274
deserialize_shape_model, 1865
deserialize_shape_model_3d, 107

deserialize_sheet_of_light_model, 350
deserialize_structured_light_model, 1306
deserialize_surface_model, 124
deserialize_template, 1677
deserialize_texture_inspection_model, 1330
deserialize_training_components, 1949
deserialize_tuple, 512
deserialize_variation_model, 1351
deserialize_xld, 516
detach_background_from_window, 1089
detach_drawing_object_from_window, 1090
detect_edge_segments, 2392
determinant_matrix, 2043
Determinante einer homogenen 2D-Matrix, 2746
Determinante einer homogenen 3D-Matrix, 2788
Determinante einer Matrix, 2043
determine_deformable_model_params, 1795
determine_ncc_model_params, 1983
determine_shape_model_params, 1865
dev_clear_obj, 681
dev_clear_window, 681
dev_close_inspect_ctrl, 682
dev_close_tool, 683
dev_close_window, 683
dev_disp_text, 684
dev_display, 686
dev_error_var, 687
dev_get_exception_data, 687
dev_get_preferences, 689
dev_get_system, 689
dev_get_window, 690
dev_inspect_ctrl, 690
dev_map_par, 1619
dev_map_prog, 1620
dev_map_var, 1620
dev_open_dialog, 691
dev_open_file_dialog, 692
dev_open_tool, 693
dev_open_window, 697
dev_set_check, 700
dev_set_color, 701
dev_set_colored, 703
dev_set_contour_style, 704
dev_set_draw, 705
dev_set_line_width, 705
dev_set_lut, 706
dev_set_paint, 707
dev_set_part, 708
dev_set_preferences, 709
dev_set_shape, 710
dev_set_tool_geometry, 711
dev_set_window, 712
dev_set_window_extents, 712
dev_show_tool, 714
dev_unmap_par, 1620
dev_unmap_prog, 1621
dev_unmap_var, 1621
dev_update_pc, 715
dev_update_time, 715
dev_update_var, 716
dev_update_window, 716
deviation_image, 974
deviation_n, 959
diagonale Matrixelemente, 2058
diagonale Matrixelemente setzen, 2062
Dialog öffnen, 691
diameter_region, 2308
diameter_xld, 3007
Dictionary einlesen, 2889
Dictionary erzeugen, 2884
Dictionary kopieren, 2883
Dictionary schreiben, 2893
Dictionary-Eintrag mit einem Objekt als Wert, 2885, 2891
Dictionary-Eintrag mit Tupel als Wert, 2888
Dictionary-Schlüssel entfernen, 2890
diff_of_gauss, 883
difference, 2290
difference_closed_contours_xld, 2993
difference_closed_polygons_xld, 2994
Differenz von ikonischen Tupeln, 2247
Differenz von Regionen, 2290
Differenz von XLD-Konturen, 2993, 2997, 2999
Differenz von XLD-Polygonen, 2994
Differenz von zwei Tupeln, 2923
Dilation-Operator, 1684, 1686, 2078–2080, 2100, 2101, 2103, 2104, 2113, 2114
dilation1, 2100
dilation2, 2101
dilation_circle, 2103
dilation_golay, 1684
dilation_rectangle1, 2104
dilation_seq, 1686
disp_arc, 1017
disp_arrow, 1019
disp_caltab, 1401
disp_channel, 1020
disp_circle, 1021
disp_color, 1022
disp_cross, 1023
disp_distribution, 1626
disp_ellipse, 1024
disp_image, 1026
disp_line, 1027
disp_lut, 1627
disp_obj, 1028
disp_object_model_3d, 1029
disp_polygon, 1033
disp_rectangle1, 1034
disp_rectangle2, 1036
disp_region, 1037
disp_text, 1132

disp_xld, 1038
Disparität aus binokularem Stereoabstand, 264
Disparität eines rektifizierten Bildpaars mit Korrelationsmethoden berechnen, 250
Disparitätsbild mittels korrelationsbasiertem Stereo, 241
Disparitätsbild mittels Lichtschnittverfahren, 353
Disparitätsbild mittels Mehrgitter-Stereo, 244
disparity_image_to_xyz, 260
disparity_to_distance, 261
disparity_to_point_3d, 263
display_scene_3d, 1005
dist_ellipse_contour_points_xld, 3008
dist_ellipse_contour_xld, 3009
dist_rectangle2_contour_points_xld, 3011
distance_cc, 2646
distance_cc_min, 2647
distance_cc_min_points, 2648
distance_contours_xld, 2649
distance_funct_1d, 2626
distance_lc, 2650
distance_lr, 2651
distance_object_model_3d, 172
distance_pc, 2652
distance_pl, 2652
distance_pp, 2653
distance_pr, 2654
distance_ps, 2655
distance_rr_min, 2656
distance_rr_min_dil, 2657
distance_sc, 2658
distance_sl, 2658
distance_sr, 2660
distance_ss, 2661
distance_to_disparity, 264
distance_transform, 2358
Distanzen zwischen 3D-Objektmodellpunkten und 3D-Objektmodell berechnen, 172
Distanztransformation einer Region, 2358
div_element_matrix, 2004
div_element_matrix_mod, 2005
div_image, 727
Divisionsmodell der Objektivverzeichnung, 1423
Divisionsrest von Tupel-Elementen, 2865, 2871
do_ocr_multi, 1715
do_ocr_multi_class_cnn, 2126
do_ocr_multi_class_knn, 2146
do_ocr_multi_class_mlp, 2166
do_ocr_multi_class_svm, 2212
do_ocr_single, 1716
do_ocr_single_class_cnn, 2127
do_ocr_single_class_knn, 2147
do_ocr_single_class_mlp, 2167
do_ocr_single_class_svm, 2213
do_ocr_word_cnn, 2128
do_ocr_word_knn, 2148
do_ocr_word_mlp, 2168
do_ocr_word_svm, 2213
do_ocv_simple, 1298
dots_image, 943
drag_region1, 1144
drag_region2, 1145
drag_region3, 1146
draw_circle, 1147
draw_circle_mod, 1148
draw_ellipse, 1149
draw_ellipse_mod, 1151
draw_line, 1152
draw_line_mod, 1153
draw_nurbs, 1154
draw_nurbs_interp, 1156
draw_nurbs_interp_mod, 1158
draw_nurbs_mod, 1161
draw_point, 1163
draw_point_mod, 1164
draw_polygon, 1165
draw_rectangle1, 1166
draw_rectangle1_mod, 1167
draw_rectangle2, 1168
draw_rectangle2_mod, 1169
draw_region, 1170
draw_xld, 1171
draw_xld_mod, 1172
Dual-Rank-Filter, 2071
dual_quat_compose, 2807
dual_quat_conjugate, 2808
dual_quat_interpolate, 2809
dual_quat_normalize, 2810
dual_quat_to_hom_mat3d, 2811
dual_quat_to_pose, 2822
dual_quat_to_screw, 2812
dual_quat_trans_line_3d, 2813
dual_rank, 2071
dual_threshold, 2431
duale Quaternion aus Schraubungsparametern erzeugen, 2814
duale Quaternion in 3D-Pose umrechnen, 2822
duale Quaternion in homogene 3D-Matrix umrechnen, 2811
duale Quaternion konjugieren, 2808
duale Quaternion normalisieren, 2810
duale Quaternionen interpolieren, 2809
duale Quaternionen multiplizieren, 2807
dump_window, 1045
dump_window_image, 1047
Durchschnittsmenge, 2642, 2665–2674
Durchschnittsmenge von Regionen, 2291
Durchschnittsmenge von XLD-Konturen, 2995
Durchschnittsmenge von XLD-Polygonen, 2996
Durchschnittsmenge von zwei Tupeln, 2924
DXF-Datei, 517, 520
DXF-Format, 523, 526
dyn_threshold, 2433

Ebenenlinien glätten, 874
eccentricity, 2309

eccentricity_points_xld, 3012
eccentricity_xld, 3013
Ecken extrahieren, 951
Ecken verbessern, 941
edges_color, 884
edges_color_sub_pix, 887
edges_image, 889
edges_object_model_3d, 200
edges_sub_pix, 892
eigenvalues_general_matrix, 2034
eigenvalues_symmetric_matrix, 2035
Eigenwerte einer Matrix, 2034–2036, 2038
ein Modell der Objektdetektion erstellen (Deep Learning), 668
elementweises Exklusiv-ODER von Tupeln, 2920
elementweises Maximum von Tupeln, 2870
elementweises Minimum von Tupeln, 2870
elementweises NICHT von Tupeln, 2919
elementweises ODER von Tupeln, 2919
elementweises UND von Tupeln, 2918
eliminate_min_max, 834
eliminate_runs, 2359
eliminate_sp, 836
Ellipse ändern, 1151
Ellipse als Zeichenobjekt erzeugen, 1081
Ellipse anzeigen, 1024
Ellipse zeichnen, 1149
Ellipsensektor als Zeichenobjekt erzeugen, 1082
elliptic_axis, 2310
elliptic_axis_gray, 452
elliptic_axis_points_xld, 3014
elliptic_axis_xld, 3015
elliptische Region erzeugen, 2258, 2259
else, 2473
elseif, 2473
emphasize, 980
Ende der Ausführung mit Ausnahme, 2474
Ende einer for-Schleife, 2473
Ende einer until-Schleife, 2487
Ende einer while-Schleife, 2475
Ende eines Konditionalblocks, 2474
endfor, 2473
endif, 2474
Endpunkte einer XLD-Kontur abschneiden, 3059
Endpunkte eines Skeletts, 2363
endswitch, 2474
endtry, 2474
endwhile, 2475
Energie eines Zweikanalbildes, 751
energy_gabor, 751
enqueue_message, 2545
enquire_class_box, 1656
enquire_reject_class_box, 1657
entropy_gray, 452
entropy_image, 975
Entzerrungsgitter erzeugen, 2680, 2681
equ_histo_image, 981
Erfolg der Kamerakalibrierung überprüfen, 1423, 1429
Ergebnis (Automatic Text Reader) löschen, 2187
Ergebnis (Manual Text Finder) löschen, 2187
Ergebnis einer Kamerakalibrierung erhalten, 1429
Ergebnisse der 1D-Vermessung in 3D-Koordinaten transformieren, 2975
Ergebnisse der 1D-Vermessung visualisieren, 2975
Ergebnisse der Blob-Analyse in 3D-Koordinaten transformieren, 2975
Ergebnisse des 3D-Matchings visualisieren, 115, 1897
Ergebnisse des Matchings (komponentenbasiert) prüfen, 1964
Ergebnisse einer Texturinspektion, 1328, 1334
Ergebnisse in 3D transformieren, 1465, 1467
Ergebnisse visualisieren, 213
Erosion-Operator, 1687, 1688, 2081–2083, 2106, 2107, 2109, 2110, 2116, 2117
erosion1, 2106
erosion2, 2107
erosion_circle, 2109
erosion_golay, 1687
erosion_rectangle1, 2110
erosion_seq, 1688
Essential-Matrix aus Punktentsprechungen, 290
Essential-Matrix aus RANAC-Punktmatching, 273
Essential-Matrix in Fundamental-Matrix transformieren, 265
essential_to_fundamental_matrix, 265
estimate_al_am, 326
estimate_noise, 454
estimate_sl_al_lr, 327
estimate_sl_al_zc, 327
estimate_tilt_lr, 328
estimate_tilt_zc, 328
euklidischen Klassifikator trainieren, 2413
Euler-Zahl einer Region, 2311
euler_number, 2311
evaluate_class_gmm, 1496
evaluate_class_mlp, 1542
evaluate_class_svm, 1582
Event erzeugen (Multithreading), 2540
Event löschen (Multithreading), 2535
Event signalisieren (Multithreading), 2561
executable_expression, 2475
exhaustive_match, 925
exhaustive_match_mg, 926
exit, 2475
exp_image, 728
expand_domain_gray, 960
expand_gray, 2381
expand_gray_ref, 2383
expand_line, 1730
expand_region, 2360
Exponentialfunktion eines Bildes, 728
Exponentialfunktion von Tupeln, 2862
Exponentialfunktion zur Basis 10 von Tupeln, 2863
Exponentialfunktion zur Basis 2 von Tupeln, 2863
export_def, 2476
externe Kameraparameter, 1423

externe Kameraparameter kalibrieren, 1423
Exzentrizität einer Region, 2309
Exzentrizität einer XLD-Kontur, 3012, 3013

Förstner-Operator, 944
Farbbild anzeigen, 1022
Farbbild in Grauwertbild transformieren, 788, 789
Farbe eines Mosaikbildes korrigieren, 2710
Farbe setzen, 701, 1113, 1118, 1127, 1128
Farben klassifizieren, 2408–2410, 2412
Farben trainieren, 469, 1508, 1561, 1595
Farbkanten extrahieren, 884, 887
Farblinien extrahieren, 917
Farbraum transformieren, 781, 786, 790, 797
Farbverarbeitung beschleunigen, 2401, 2405, 2508, 2580
fast_match, 1677
fast_match_mg, 1678
fast_threshold, 2435
Fehler der Konturanzeige approximieren, 1630
Fehlerbehandlungmodus setzen, 700
Fehlerbehandlungsmodus, 2505
Fehlerbehandlungsmodus konfigurieren, 2508
Fehlercode, 2506
Fehlerfunktion von Tupel-Elementen, 2861
Fehlermeldung, 2506
Fehlersuchmodus, 2507
Fehlersuchmodus setzen, 2510
Fehlervariable erzeugen, 687
Fenstergröße, 1052
Fenstergröße setzen, 1062
Fensterinhalt schreiben, 1045, 1047
Fensterparameter, 1051, 1107
Fensterparameter setzen, 1061, 1130
Fensterposition, 1052
Fensterposition setzen, 1062
Fenstertyp, 1054
Fenstertyp setzen, 1063
fft_generic, 752
fft_image, 754
fft_image_inv, 755
file_exists, 507
fill_interlace, 837
fill_up, 2362
fill_up_shape, 2362
Filter erzeugen, 760
Filter für schnelle Fourier-Transformation (FFT) erzeugen, 756–758, 761, 763–765, 768, 769
filter_kalman, 1760
find_aniso_shape_model, 1868
find_aniso_shape_models, 1874
find_bar_code, 1181
find_box_3d, 79
find_calib_descriptor_model, 1824
find_calib_object, 1402
find_caltab, 1405
find_component_model, 1950
find_data_code_2d, 1220
find_deformable_surface_model, 89
find_generic_shape_model, 1882
find_local_deformable_model, 1798
find_marks_and_pose, 1406
find_ncc_model, 1985
find_ncc_models, 1989
find_neighbors, 2312
find_planar_calib_deformable_model, 1801
find_planar_uncalib_deformable_model, 1803
find_rectification_grid, 2681
find_scaled_shape_model, 1883
find_scaled_shape_models, 1889
find_shape_model, 1897
find_shape_model_3d, 107
find_shape_models, 1902
find_surface_model, 125
find_surface_model_image, 134
find_text, 2189
find_uncalib_descriptor_model, 1826
fit_circle_contour_xld, 3016
fit_ellipse_contour_xld, 3018
fit_line_contour_xld, 3021
fit_primitives_object_model_3d, 190
fit_rectangle2_contour_xld, 3023
fit_surface_first_order, 456
fit_surface_second_order, 457
fitting, 1689
Fitting von XLD-Konturen durchführen, 2975, 2978, 2983, 3016, 3018, 3021, 3023
Fläche der Löcher einer Region, 2302
Fläche einer Region, 448, 2300
Fläche einer XLD-Kontur, 3002, 3003
Floor-Funktion von Tupel-Elementen, 2865
flush_buffer, 1048
fnew_line, 528
for, 2477
for-Schleife starten, 2477
Form einer Region transformieren, 2370
Form einer XLD-Kontur transformieren, 3068
Formmodell (formbasiert) prüfen, 1897
Formmodell prüfen, 1107, 1856
fread_bytes, 528
fread_char, 529
fread_line, 530
fread_serialized_item, 2597
fread_string, 532
Frei-Filter, 895, 896
frei_amp, 895
frei_dir, 896
full_domain, 391
funct_1d_to_pairs, 2627
Fundamental-Matrix aus Punktkorrespondenzen, 292, 294
Fundamental-Matrix aus RANSAC-Punktmatching, 277, 281
Fundamental-Matrix aus relativer Kamera-Lage (3D-Pose), 288
fuse_object_model_3d, 202

Fuzzy-Entropie, 459
Fuzzy-Messfunktion setzen, 23
Fuzzy-Messfunktion zurücksetzen, 22
Fuzzy-Umfang, 460
fuzzy_entropy, 459
fuzzy_measure_pairing, 4
fuzzy_measure_pairs, 6
fuzzy_measure_pos, 8
fuzzy_perimeter, 460
fwrite_bytes, 533
fwrite_serialized_item, 2597
fwrite_string, 534

Gabor-Filter, 749, 751, 761
gamma_image, 729
Gammafunktion von Tupel-Elementen, 2867, 2879
Gammakorrektur des Bildes, 729
Gauß-Filter, 763
Gauß-Filter-Funktionen, 839, 1622
Gauß-Funktion, 2634
Gaußsche Ableitungen, 879
Gaußsche Rauschverteilung erzeugen, 955
gauss_distribution, 955
gauss_filter, 839
gauss_image, 1622
Geführte Filterung eines Bildes, 840
gefundene Komponenten eines Matching-Modells, 1964
gegebene Punktkorrespondenzen, 2765
gekrümmte Grauwertfläche einpassen, 456, 457
gekrümmte Grauwertoberfläche, 419, 420
gen_arbitrary_distortion_map, 2682
gen_bandfilter, 756
gen_bandpass, 757
gen_binocular_proj_rectification, 266
gen_binocular_rectification_map, 269
gen_box_object_model_3d, 152
gen_bundle_adjusted_mosaic, 2716
gen_caltab, 1409
gen_checker_region, 2253
gen_circle, 2254
gen_circle_contour_xld, 2971
gen_circle_sector, 2256
gen_contour_nurbs_xld, 2972
gen_contour_polygon_rounded_xld, 2974
gen_contour_polygon_xld, 2975
gen_contour_region_xld, 2975
gen_contours_skeleton_xld, 2977
gen_cooc_matrix, 461
gen_cross_contour_xld, 2978
gen_cube_map_mosaic, 2717
gen_cylinder_object_model_3d, 153
gen_derivative_filter, 758
gen_disc_se, 2073
gen_dl_model_heatmap, 632
gen_dl_pruned_model, 634
gen_ellipse, 2258
gen_ellipse_contour_xld, 2978
gen_ellipse_sector, 2259
gen_empty_obj, 2245
gen_empty_object_model_3d, 154
gen_empty_region, 2261
gen_filter_mask, 760
gen_gabor, 761
gen_gauss_filter, 763
gen_gauss_pyramid, 928
gen_grid_rectification_map, 2683
gen_grid_region, 2261
gen_highpass, 764
gen_image1, 404
gen_image1_extern, 405
gen_image1_rect, 407
gen_image3, 409
gen_image3_extern, 410
gen_image_const, 412
gen_image_gray_ramp, 414
gen_image_interleaved, 415
gen_image_proto, 418
gen_image_surface_first_order, 419
gen_image_surface_second_order, 420
gen_image_to_world_plane_map, 1469
gen_initial_components, 1956
gen_lowpass, 765
gen_mean_filter, 766
gen_measure_arc, 10
gen_measure_rectangle2, 13
gen_nurbs_interp, 2980
gen_object_model_3d_from_points, 154
gen_parallel_contour_xld, 2987
gen_parallels_xld, 2981
gen_plane_object_model_3d, 155
gen_polygons_xld, 2982
gen_principal_comp_trans, 785
gen_projective_mosaic, 2719
gen_psf_defocus, 988
gen_psf_motion, 989
gen_radial_distortion_map, 1471
gen_random_region, 2263
gen_random_regions, 2264
gen_rectangle1, 2266
gen_rectangle2, 2267
gen_rectangle2_contour_xld, 2983
gen_region_contour_xld, 2268
gen_region_histo, 2269
gen_region_hline, 2270
gen_region_line, 2271
gen_region_points, 2272
gen_region_polygon, 2273
gen_region_polygon_filled, 2274
gen_region_polygon_xld, 2275
gen_region_runs, 2276
gen_sin_bandpass, 768
gen_sphere_object_model_3d, 156
gen_sphere_object_model_3d_center, 157
gen_spherical_mosaic, 2721

gen_std_bandpass, 769
gen_struct_elements, 1690
gen_structured_light_pattern, 1307
generalized_eigenvalues_general_matrix, 2036
generalized_eigenvalues_symmetric_matrix, 2038
Geocodierung einlesen, 509
Gerätekontext eines Grafikfensters setzen, 1061
gerade Kantenstücke detektieren, 2392
geschätztes Hintergrundbild, 2690
get_aop_info, 2566
get_bar_code_object, 1185
get_bar_code_param, 1186
get_bar_code_param_specific, 1188
get_bar_code_result, 1189
get_bead_inspection_param, 1293
get_bg_esti_params, 2688
get_calib_data, 1429
get_calib_data_observ_contours, 1436
get_calib_data_observ_points, 1437
get_calib_data_observ_pose, 1385
get_camera_setup_param, 1438
get_channel_info, 2239
get_chapter_info, 2524
get_check, 2505
get_circle_pose, 2823
get_class_box_param, 1658
get_class_train_data_gmm, 1498
get_class_train_data_knn, 1516
get_class_train_data_mlp, 1543
get_class_train_data_svm, 1583
get_component_model_params, 1959
get_component_model_tree, 1960
get_component_relations, 1962
get_comprise, 1628
get_compute_device_info, 2495
get_compute_device_param, 2496
get_contour_angle_xld, 3026
get_contour_attrib_xld, 3027
get_contour_global_attrib_xld, 3030
get_contour_style, 1100
get_contour_xld, 3088
get_current_dir, 508
get_current_hthread_id, 2547
get_data_code_2d_objects, 1227
get_data_code_2d_param, 1229
get_data_code_2d_results, 1233
get_deep_ocr_param, 2136
get_deformable_model_contours, 1807
get_deformable_model_origin, 1808
get_deformable_model_params, 1809
get_deformable_surface_matching_result, 92
get_deformable_surface_model_param, 93
get_descriptor_model_origin, 1829
get_descriptor_model_params, 1829
get_descriptor_model_points, 1830
get_descriptor_model_results, 1831
get_diagonal_matrix, 2058
get_dict_object, 2885
get_dict_param, 2886
get_dict_tuple, 2888
get_disp_object_model_3d_info, 1048
get_display_scene_3d_info, 1006
get_distance_transform_xld_contour, 2662
get_distance_transform_xld_param, 2662
get_dl_classifier_param, 1607
get_dl_classifier_result, 1608
get_dl_classifier_train_result, 1609
get_dl_device_param, 547
get_dl_layer_param, 607
get_dl_model_layer, 608
get_dl_model_layer_activations, 609
get_dl_model_layer_gradients, 609
get_dl_model_layer_param, 610
get_dl_model_layer_weights, 610
get_dl_model_param, 634
get_dl_pruning_param, 646
get_domain, 391
get_draw, 1100
get_drawing_object_iconic, 1091
get_drawing_object_params, 1091
get_error_text, 2506
get_extended_error_info, 2506
get_features_ocr_class_knn, 2150
get_features_ocr_class_mlp, 2170
get_features_ocr_class_svm, 2215
get_fix, 1628
get_fixed_lut, 1629
get_font, 1134
get_font_extents, 1135
get_found_component_model, 1964
get_framegrabber_callback, 372
get_framegrabber_lut, 373
get_framegrabber_param, 374
get_full_matrix, 2059
get_generic_shape_model_param, 1909
get_generic_shape_model_result, 1910
get_generic_shape_model_result_object, 1912
get_grayval, 481
get_grayval_contour_xld, 482
get_grayval_interpolated, 483
get_handle_object, 2926
get_handle_param, 2927
get_handle_tuple, 2928
get_hsi, 1101
get_icon, 1102
get_image_pointer1, 484
get_image_pointer1_rect, 486
get_image_pointer3, 487
get_image_size, 488
get_image_time, 488
get_image_type, 489

get_insert, 1629
get_io_channel_param, 2515
get_io_device_param, 2516
get_keywords, 2524
get_line_approx, 1630
get_line_of_sight, 1392
get_line_style, 1102
get_line_width, 1103
get_lines_xld, 3089
get_lut, 1039
get_lut_style, 1630
get_mbutton, 1067
get_mbutton_sub_pix, 1068
get_message_obj, 2547
get_message_param, 2548
get_message_queue_param, 2550
get_message_tuple, 2551
get_metrology_model_param, 54
get_metrology_object_fuzzy_param, 55
get_metrology_object_indices, 56
get_metrology_object_measures, 57
get_metrology_object_model_contour, 58
get_metrology_object_num_instances, 59
get_metrology_object_param, 60
get_metrology_object_result, 62
get_metrology_object_result_contour, 64
get_modules, 2504
get_mposition, 1068
get_mposition_sub_pix, 1069
get_mshape, 1070
get_ncc_model_origin, 1994
get_ncc_model_params, 1994
get_ncc_model_region, 1995
get_next_socket_data_type, 2606
get_obj_class, 2240
get_object_model_3d_params, 177
get_operator_info, 2525
get_operator_name, 2527
get_os_window_handle, 1049
get_paint, 1103
get_pair_funct_1d, 2627
get_parallels_xld, 3090
get_param_info, 2527
get_param_names, 2529
get_param_num, 2530
get_param_types, 2531
get_params_class_gmm, 1499
get_params_class_knn, 1517
get_params_class_mlp, 1544
get_params_class_svm, 1584
get_params_ocr_class_cnn, 2130
get_params_ocr_class_knn, 2151
get_params_ocr_class_mlp, 2171
get_params_ocr_class_svm, 2216
get_part, 1104
get_part_style, 1105
get_pixel, 1631
get_points_ellipse, 2663
get_polygon_xld, 3091
get_pose_type, 2824
get_prep_info_class_gmm, 1500
get_prep_info_class_mlp, 1545
get_prep_info_class_svm, 1585
get_prep_info_ocr_class_mlp, 2171
get_prep_info_ocr_class_svm, 2217
get_rectangle_pose, 2825
get_region_chain, 1724
get_region_contour, 2375
get_region_convex, 2376
get_region_index, 2313
get_region_points, 2377
get_region_polygon, 2378
get_region_runs, 2378
get_region_thickness, 2314
get_regress_params_xld, 3033
get_regularization_params_class_mlp, 1546
get_regularization_params_ocr_class_mlp, 2173
get_rejection_params_class_mlp, 1547
get_rejection_params_ocr_class_mlp, 2174
get_rgb, 1105
get_rgba, 1106
get_sample_class_gmm, 1501
get_sample_class_knn, 1518
get_sample_class_mlp, 1548
get_sample_class_svm, 1586
get_sample_class_train_data, 1568
get_sample_identifier_object_info, 1275
get_sample_identifier_param, 1277
get_sample_num_class_gmm, 1502
get_sample_num_class_knn, 1518
get_sample_num_class_mlp, 1549
get_sample_num_class_svm, 1587
get_sample_num_class_train_data, 1568
get_serial_param, 2600
get_serialized_item_ptr, 2598
get_shape, 1107
get_shape_model_3d_contours, 112
get_shape_model_3d_params, 113
get_shape_model_clutter, 1913
get_shape_model_contours, 1914
get_shape_model_origin, 1914
get_shape_model_params, 1915
get_sheet_of_light_param, 351
get_sheet_of_light_result, 353
get_sheet_of_light_result_object_model_3d, 354
get_size_matrix, 2044
get_socket_descriptor, 2606
get_socket_param, 2607
get_spy, 2507

get_stereo_model_image_pairs, 306
get_stereo_model_object, 306
get_stereo_model_object_model_3d, 307
get_stereo_model_param, 308
get_string_extents, 1136
get_structured_light_model_param, 1311
get_structured_light_object, 1312
get_sub_matrix, 2060
get_support_vector_class_svm, 1588
get_support_vector_num_class_svm, 1589
get_support_vector_num_ocr_class_svm, 2219
get_support_vector_ocr_class_svm, 2219
get_surface_matching_result, 135
get_surface_model_param, 137
get_system, 2575
get_system_info, 2579
get_system_time, 2491
get_text_model_param, 2190
get_text_object, 2191
get_text_result, 2192
get_texture_inspection_model_image, 1331
get_texture_inspection_model_param, 1332
get_texture_inspection_result_object, 1334
get_threading_attrib, 2552
get_thresh_images_variation_model, 1352
get_tposition, 1136
get_training_components, 1966
get_tshape, 1632
get_value_matrix, 2061
get_variation_model, 1353
get_window_attr, 1051
get_window_background_image, 1092
get_window_extents, 1052
get_window_param, 1107
get_window_pointer3, 1053
get_window_type, 1054
get_y_value_funct_1d, 2628
give_bg_esti, 2690
global, 2479
globale Variable deklarieren (HDevelop), 2479
GMM-Klassifikator einlesen, 1503
GMM-Klassifikator erzeugen, 1493, 1505
GMM-Klassifikator löschen, 1492
GMM-Klassifikator schreiben, 1510
GMM-Klassifikator trainieren, 1490, 1492, 1501, 1502, 1508, 2397
GMM-Klassifikator wiederverwenden, 1499, 1500, 1503, 1510
Gnuplot, 1633–1636
Gnuplot-Datei öffnen, 1633
Gnuplot-Dateien schließen, 1632
Gnuplot-Pipe öffnen, 1634
gnuplot_close, 1632
gnuplot_open_file, 1633
gnuplot_open_pipe, 1634
gnuplot_plot_ctrl, 1634
gnuplot_plot_funct_1d, 1635
gnuplot_plot_image, 1636
golay_elements, 1691
Größe eines Grafikfensters setzen, 712
größter Wert einer 1D-Funktion (Maximum), 2637, 2638
grab_data, 375
grab_data_async, 376
grab_image, 377
grab_image_async, 378
grab_image_start, 380
Grad in Bogenmaß umrechnen, 2874
Grafik-Rechteck kopieren, 1044
Grafikfenster öffnen, 697, 1054, 1056
Grafikfenster erzeugen, 1054
Grafikfenster löschen, 681, 1042, 1624
Grafikfenster schließen, 683, 1043
Grauwert, 481
Grauwert setzen, 441–445, 447, 1649
Grauwert von XLD-Konturpunkten, 482
Grauwert-Anisotropie, 452
Grauwert-Bereich, 470, 2088
Grauwert-Energie, 449, 450
Grauwert-Entropie, 452, 975
Grauwert-Histogramm, 464–466, 469
Grauwert-Hochebene extrahieren, 2456
Grauwert-Homogenität, 449, 450
Grauwert-Kontrast, 449, 450
Grauwert-Korrelat, 449, 450
Grauwert-Merkmalshistogramm, 476, 477
Grauwert-Mittelwert, 470
Grauwert-Projektion, 467
Grauwert-Rampe, 414
Grauwert-Skelett, 963
Grauwert-Streuung, 470, 974
Grauwert-Subpixel, 483
Grauwert-Symmetrie, 964
Grauwert-Tiefebene extrahieren, 2454
grauwertbasiertes Matching-Modell anpassen, 1665
Grauwertbild anzeigen, 1026
Grauwerte aus Kanälen auswählen, 301
Grauwerte für Regionenausgabe setzen, 1117
Grauwerte skalieren, 736, 986
Grauwertebene anpassen, 472
Grauwertmerkmale einer XLD-Parallelen, 3036
Grauwertprofil, 19
gray_bothat, 2074
gray_closing, 2075
gray_closing_rect, 2076
gray_closing_shape, 2077
gray_dilation, 2078
gray_dilation_rect, 2079
gray_dilation_shape, 2080

gray_erosion, 2081
gray_erosion_rect, 2082
gray_erosion_shape, 2083
gray_features, 463
gray_histo, 464
gray_histo_abs, 465
gray_histo_range, 466
gray_inside, 962
gray_opening, 2084
gray_opening_rect, 2085
gray_opening_shape, 2086
gray_projections, 467
gray_range_rect, 2088
gray_skeleton, 963
gray_tophat, 2089
Grenze einer Region, 2094
guided_filter, 840

Höhenmodell des Schattens, 338
HALCON spy, 2507, 2508, 2510
HALCON Systemparameter, 2575, 2579
HALCON-Datenbank zurücksetzen, 2504
HALCON-Systemparameter setzen, 2580
Hamming-Abstand einer Region, 2315, 2316
hamming_change_region, 1725
hamming_distance, 2315
hamming_distance_norm, 2316
Hand-Auge-Kalibrierung, 1380
Hand-Auge-Systemparameter kalibrieren, 1386
hand_eye_calibration, 1386
Handle für Trainingsdaten erzeugen, 1566
Handle für Trainingsdaten löschen, 1565
handle_to_integer, 2907
harmonic_interpolation, 866
harmonische Interpolation, 866
Harris-Operator, 947, 949
Hauptachsentransformation, 788
Hauptachsentransformation (PCA), 785, 786
Hauptachsentransformation (PCA) der Bildmatrix, 785
HDevelop-Einstellungen, 689
HDevelop-Einstellungen setzen, 709
HDevelop-System, 689
height_width_ratio, 2317
height_width_ratio_xld, 3035
Hessesche Normalform, 2697–2700, 2702
highpass_image, 898
Hintergrundbild ändern, 2694
Hintergrundbild mit HALCON-Fenster verknüpfen, 1076
Hintergrundbild schätzen, 2691
Hintergrundbild von einem graphischen HALCON Fenster, 1092
Hintergrundbild von einem graphischen HALCON Fenster entfernen, 1089
Hintergrundschätzer erzeugen, 2686
Hintergrundschätzer löschen, 2685
histo_2dim, 469
histo_to_thresh, 2436
Histogramm eines Tupels, 2929
Hit-or-Miss-Operator, 1695, 1696, 2112
hit_or_miss, 2112
hit_or_miss_golay, 1695
hit_or_miss_seq, 1696
Hochpassfilter, 764, 898
hom_mat2d_compose, 2745
hom_mat2d_determinant, 2746
hom_mat2d_identity, 2746
hom_mat2d_invert, 2747
hom_mat2d_reflect, 2748
hom_mat2d_reflect_local, 2749
hom_mat2d_rotate, 2751
hom_mat2d_rotate_local, 2752
hom_mat2d_scale, 2753
hom_mat2d_scale_local, 2755
hom_mat2d_slant, 2756
hom_mat2d_slant_local, 2758
hom_mat2d_to_affine_par, 2759
hom_mat2d_translate, 2760
hom_mat2d_translate_local, 2761
hom_mat2d_transpose, 2763
hom_mat3d_compose, 2786
hom_mat3d_determinant, 2788
hom_mat3d_identity, 2788
hom_mat3d_invert, 2789
hom_mat3d_project, 2763
hom_mat3d_rotate, 2790
hom_mat3d_rotate_local, 2792
hom_mat3d_scale, 2793
hom_mat3d_scale_local, 2795
hom_mat3d_to_pose, 2796
hom_mat3d_translate, 2797
hom_mat3d_translate_local, 2798
hom_mat3d_transpose, 2799
hom_vector_to_proj_hom_mat2d, 2765
homogene 2D-Einheitsmatrix erzeugen, 2746
homogene 2D-Matrix an lokalen Achsen spiegeln, 2749
homogene 2D-Matrix drehen, 2751
homogene 2D-Matrix in affine Parameter umrechnen, 2759
homogene 2D-Matrix invertieren, 2747
homogene 2D-Matrix mit Slant, 2756
homogene 2D-Matrix skalieren, 2753
homogene 2D-Matrix spiegeln, 2748
homogene 2D-Matrix transponieren, 2763
homogene 2D-Matrix um lokale Achse neigen, 2758
homogene 2D-Matrix um lokale Achsen drehen, 2752
homogene 2D-Matrix um lokale Achsen skalieren, 2755
homogene 2D-Matrix um lokale Achsen verschieben, 2761
homogene 2D-Matrix verschieben, 2760
homogene 2D-Matrizen multiplizieren, 2745
homogene 3D-Einheitsmatrix erzeugen, 2788
homogene 3D-Matrix drehen, 2790
homogene 3D-Matrix in 3D-Pose umrechnen, 2796

homogene 3D-Matrix in projektive 2D-Matrix umrechnen, 2763
homogene 3D-Matrix invertieren, 2789
homogene 3D-Matrix skalieren, 2793
homogene 3D-Matrix transponieren, 2799
homogene 3D-Matrix um lokale Achsen drehen, 2792
homogene 3D-Matrix um lokale Achsen skalieren, 2795
homogene 3D-Matrix um lokale Achsen verschieben, 2798
homogene 3D-Matrix verschieben, 2797
homogene 3D-Matrizen multiplizieren, 2786
homogene Matrix, 2765
Hough-Bild für Geraden berechnen, 2697, 2698
Hough-Transformation für Kreise berechnen, 2696
hough_circle_trans, 2695
hough_circles, 2696
hough_line_trans, 2697
hough_line_trans_dir, 2698
hough_lines, 2699
hough_lines_dir, 2700
HSI-Codierung der aktuellen Farbe, 1101
HSI-Farbraum, 1118
Hyperbox-Klassifikator einlesen, 1661
Hyperbox-Klassifikator erzeugen, 1654
Hyperbox-Klassifikator löschen, 1653
Hyperbox-Klassifikator schreiben, 1665
Hyperbox-Klassifikator trainieren, 1658, 1659, 1731
Hyperbox-Klassifikator wiederverwenden, 1658, 1661, 1665
hyperzentrische Kamera, 1423
Hypotenuse von Tupel-Elementen, 2866
hysteresis_threshold, 2394

I/O-Übertragungskanal öffnen, 2516
I/O-Übertragungskanal schließen, 2511
I/O-Gerät öffnen, 2517
I/O-Gerät schließen, 2512
ID des ausgeführten Threads zurückgeben, 2547
Identifikation, 1266, 1268, 1270, 1272–1275, 1277–1283, 1285, 1286
Identifikation (Samplebasiert), 1266, 1268, 1270, 1272–1275, 1277–1283, 1285, 1286
Identifikation mittels Samples, 1266, 1268, 1270, 1272–1275, 1277–1283, 1285, 1286
if, 2480
ifelse, 1732
ikonische Ergebnisse des Modells für strukturiertes Licht, 1312
ikonische Objekte vergleichen, 2238, 2241
ikonische Objekte verknüpfen, 2243
ikonische Objekte zählen, 2239
ikonische Zwischenergebnisse einer Stereo-Rekonstruktion, 306
ikonisches Objekt, 1091
ikonisches Objekt aus einer Nachricht lesen (Inter-Thread Kommunikation), 2547
ikonisches Objekt einfügen, 2245
ikonisches Objekt einlesen, 501
ikonisches Objekt entfernen, 2249
ikonisches Objekt ersetzen, 2249
ikonisches Objekt in eine Nachricht schreiben (Inter-Thread Kommunikation), 2555
ikonisches Objekt kopieren, 2244
ikonisches Objekt löschen, 681
ikonisches Objekt schreiben, 502
illuminate, 982
image_points_to_world_plane, 1472
image_to_channels, 440
image_to_world_plane, 1474
import, 2480
import_lexicon, 2158
Impulsantwort von Bewegungsunschärfe erzeugen, 989
Impulsantwort von Defokussierung erzeugen, 988
Index des ersten gefundenen Tupel zurückgeben, 2895
Index des letzten gefundenen Tupel zurückgeben, 2895
Index-Schlüsselwörter, 2524
Inferieren eines Samples und generieren einer Heatmap (Deep Learning), 632
info_edges, 899
info_framegrabber, 381
info_ocr_class_box, 1717
info_parallels_xld, 3036
info_smooth, 843
Information über Compute Device, 2495
Information über Glättungsfilter, 843
Information über Operator, 2525
Information über Parameter, 2527
Informationen über ein Handle zurückgeben, 2927
Informationen über I/O-Schnittstelle, 2520
Informationen über Kanäle eines I/O-Geräts, 2519
Informationen zu dargestellten Objektmodellen einer 3D-Szene, 1006
Informationsgehalt (GMM), 1500
Informationsgehalt (MLP), 1545
Informationsgehalt (SVM), 1585
Informationsgehalt eines OCR-Klassifikators, 2171, 2217
Inhalt eines Handles freigeben, 2904
init_compute_device, 2497
inner_circle, 2318
inner_rectangle1, 2319
innerer Kreis einer Region, 2318
inneres Rechteck einer Region, 2319
inpainting_aniso, 867
inpainting_ced, 869
inpainting_ct, 871
inpainting_mcf, 874
inpainting_texture, 875
insert, 2481
insert_obj, 2245
inspect_clustered_components, 1968
inspect_lexicon, 2159
inspect_shape_model, 1916

Instanz eines 3D-Objektmodells zu 3D-Szene hinzufügen, 1000
integer_to_handle, 2908
integer_to_obj, 2246
Integral einer 1D-Funktion, 2629
integrate_funct_1d, 2629
intensity, 470
interaktive Anzeige, 1065, 1651
interjacent, 1726
interleave_channels, 422
interne Kameraparameter, 1423
interne Kameraparameter einlesen, 1417
interne Kameraparameter in Kameramatrix umrechnen, 1415
interne Kameraparameter kalibrieren, 1423
interne Kameraparameter schreiben, 1418
interne Kameraparameter und 3D-Pose in homogene 3D-Matrix umrechnen, 1460
interpolate_scattered_data, 2705
interpolate_scattered_data_image, 2706
interpolate_scattered_data_points_to_image, 2707
Interpolation verstreuter Datenpunkte, 2703–2707
Interpolationsmodus für Grauwertausgabe setzen, 1126
Interpolationsmodus für Grauwerte, 1105
interrupt_operator, 2552
intersect_lines_of_sight, 272
intersect_plane_object_model_3d, 207
intersection, 2291
intersection_circle_contour_xld, 2665
intersection_circles, 2666
intersection_closed_contours_xld, 2995
intersection_closed_polygons_xld, 2996
intersection_contours_xld, 2667
intersection_line_circle, 2668
intersection_line_contour_xld, 2669
intersection_lines, 2670
intersection_ll, 1764
intersection_segment_circle, 2671
intersection_segment_contour_xld, 2672
intersection_segment_line, 2673
intersection_segments, 2674
intitiale Komponenten eines Matching-Modells, 1956
Inverse berechnen, 2629
inverse Polartransformation, 813
inverse Polartransformation einer Region, 2284
inverse Polartransformation einer XLD-Kontur, 2990
inverse schnelle Fourier-Transformation (FFT), 755
inverses Tupel, 2902
invert_funct_1d, 2629
invert_image, 730
invert_matrix, 2006
invert_matrix_mod, 2008
isotropic_diffusion, 844

junctions_skeleton, 2363

Kalibrierobjekt für Kamerakalibrierung setzen, 1448
Kalibrierobjekt für Lichtschnitt-Kalibrierung erzeugen, 346
Kalibrierobjekt finden, 1402
Kalibrierparameter einer Kamera setzen, 1445
Kalibrierplatte anzeigen, 1401
Kalibrierplatte erzeugen, 1395, 1409
Kalibrierplatte finden, 1405
Kalibrierplattenpunkte einlesen, 1393
kalibrierte externe Kameraparameter erhalten, 1429
kalibrierte interne Kameraparameter erhalten, 1429
kalibriertes Kameraaufbaumodell, 1429
Kalman-Filter einlesen, 1767
Kamera aus 3D-Szene entfernen, 1007
Kamera zu 3D-Szene hinzufügen, 999
Kamera-Kalibrierparameter erhalten, 1429
Kameraaufbaumodell erzeugen, 1426
Kameraaufbaumodell löschen, 1424
Kameraaufbaumodell lesen, 1441
Kameraaufbaumodell schreiben, 1455
Kamerakalibrierdatenmodell erzeugen, 1425
Kamerakalibrierdatenmodell löschen, 1424
Kamerakalibrierdatenmodell lesen, 1441
Kamerakalibrierdatenmodell schreiben, 1455
Kameramatrix in interne Kameraparameter umrechnen, 1414
Kameramodell, 1423
Kameraparameter kalibrieren, 1423
Kamerapose in 3D-Szene setzen, 1009
Kameraskalierungsfaktor, 1423
Kanäle zur Farbverarbeitung aufspalten, 427, 434, 435
Kanäle zur Farbverarbeitung zusammenstellen, 428, 429, 433
Kanal zum Bild hinzufügen, 428
Kanten bearbeiten (pixelgenau), 877, 878, 2119, 2120, 2351, 2357, 2373
Kanten eines 3D-Objektmodells finden, 200
Kanten extrahieren (pixelgenau), 889, 1916, 2371, 2394, 2396, 2439
Kanten oder Linien extrahieren (subpixelgenau) , 819, 879, 883, 884, 887, 892, 917, 919, 921, 2446, 2447
Kantenlücken schließen, 877, 878
Kantenpaar-Positionen messen (1D-Vermessung), 15
Kantenposition messen (1D-Vermessung), 17
Kapitel, 2524
Kirsch-Filter, 900, 902
kirsch_amp, 900
kirsch_dir, 902
Klasse evaluieren (GMM), 1496
Klassifikations-Nachschlagetabelle (LUT) löschen, 1526

Klassifikationsresultate löschen (Deep Learning), 1605
Klassifikator anwenden (Deep Learning), 1603
Klassifikator einlesen (Deep Learning), 1610
Klassifikator löschen (Deep Learning), 1605
Klassifikator schreiben (Deep Learning), 1618
Klassifikator trainieren (Deep Learning), 1616
Klassifikator wiederverwenden, 1590
Klassifizierungsparameter (GMM), 1499
Klassifizierungsparameter (Hyperbox), 1658
Klassifizierungsparameter (Hyperbox) setzen, 1663
Klassifizierungsparameter (kNN), 1517
Klassifizierungsparameter (kNN) setzen, 1523
Klassifizierungsparameter (MLP), 1544, 1546
Klassifizierungsparameter (MLP) setzen, 1554
Klassifizierungsparameter (SVM), 1584
Klassifizierungsparameter setzen (Deep Learning), 1613
kleinster umschließender Kreis einer Region, 2343
kleinster umschließender Kreis einer XLD-Kontur, 3052
kleinster Wert einer 1D-Funktion (Minimum), 2637, 2638
kleinstes Tupel-Element (Minimum), 2933
kleinstes umschließendes Rechteck einer Region, 2317, 2345, 2346
kleinstes umschließendes Rechteck einer XLD-Kontur, 3035, 3053, 3054
kNN-Klassifikator einlesen, 1519
kNN-Klassifikator erzeugen, 1515, 1520
kNN-Klassifikator löschen, 1514
kNN-Klassifikator schreiben, 1525
kNN-Klassifikator trainieren, 1512, 1524, 2398
Kohärenztransport, 871
kohärenzverbessernde Diffusion, 869, 978
Kommentarzeile, 2471
Kompaktheit einer Region, 2304
Kompaktheit einer XLD-Kontur, 3005
Komplement einer Region, 2289
komplexes Bild, 478, 479
Komponenten von Matching-Modell prüfen, 1968
Konditionalblock, 2480, 2484
Konditionalblock mit Alternative, 1732
Konfidenz (Hyperbox) der optische Schriftzeichenerkennung (OCR) testen, 1721
Kontrast verbessern, 980, 982
Kontrolldaten einer NURBS-Kurve erzeugen, 2980
Kontrollvariablen für Plot, 1635
Kontrollvariablenfenster öffnen, 690
Kontrolvariablen-Fenster schließen, 682
Kontur ändern, 1172
Kontur einer Region als Kettencode, 1724
Kontur nach Position sortieren, 3070
Konturanzeige, 704, 1100, 1116
Konturattribute für Kantenextraktion (subpixelgenau) bestimmen, 3027, 3030, 3045, 3046
Konturen eines 3D-Matching-Modells (formbasiert) erhalten, 112
Konturen eines Matching-Modells (formbasiert), 1914
Konturen eines Matching-Modells (perspektivisch deformierbar), 1807
Konturen sortieren, 3070
Konturlänge einer Region, 2305
konvexe Hülle einer Region, 2376
Konvexität einer Region, 2307
Konvexität einer XLD-Kontur, 3006
Kosinus eines Bildes, 726
Kosinus hyperbolicus von Tupel-Elementen, 2858
Kosinus von Tupel-Elementen, 2858
Krümmungsflussgleichung, 984
Kreis ändern, 1148
Kreis als Zeichenobjekt erzeugen, 1079
Kreis anzeigen, 1021
Kreis zeichnen, 1147
Kreisbogen anzeigen, 1017
Kreise mit Hough-Transformation extrahieren, 2695
kreisförmige Region erzeugen, 2254, 2256
kreisförmigen Druck lesen, 811, 2281, 2284
Kreissektor als Zeichenobjekt erzeugen, 1080
Kreuz anzeigen, 1023
Kreuzungen eines Skeletts, 2363
Kubikwurzel von Tupel-Elementen, 2857
kumulative Summen von Tupel-Elementen, 2859

Länge einer XLD-Kontur, 3036
Löcher in Region füllen, 2362
Lücken zwischen Regionen füllen, 2381, 2383
Label-Bild erzeugen, 425, 426
`label_to_region`, 2277
Labelbild segmentieren, 2277
Lage eines 3D-Matching-Modells (deformierbar oberflächenbasiert) verfeinern, 95
Lageschätzung, 1423, 2823, 2825
Lanser-Filter, 889, 892
`laplace`, 903
Laplace-of-Gaussian (LoG), 905
Laplace-Operator berechnen, 903
`laplace_of_gauss`, 905
Lauflängen aus Eingaberegion eliminieren, 2359
Lauflängenkodierungsdaten einer Region, 2333, 2334, 2378
Laws-Filter, 976
Ldexp-Funktion von Tupeln, 2867
`learn_class_box`, 1658
`learn_ndim_box`, 1731
`learn_ndim_norm`, 2413
`learn_sampset_box`, 1659
leere Region erzeugen, 2261
leeres ikonisches Objekt erzeugen, 2245
`length_xld`, 3036
Lepetit-Operator, 950
Lexikon einlesen, 2158
Lexikon erzeugen, 2157
Lexikon löschen, 2157
Lichtquelle aus 3D-Szene entfernen, 1008
Lichtquelle zu 3D-Szene hinzufügen, 1002

Lichtschnitt-Modell einlesen, 357
Lichtschnitt-Modell erzeugen, 347
Lichtschnitt-Modell kalibrieren, 342
Lichtschnitt-Modell löschen, 345
Lichtschnitt-Modell schreiben, 363
line_orientation, 2708
line_position, 2709
linear_trans_color, 786
lines_color, 917
lines_facet, 919
lines_gauss, 921
Linie ändern, 1153
Linie anzeigen, 1027
Linie zeichnen, 1152
Linien extrahieren, 916, 919, 921
Linien mit Hough-Transformation extrahieren, 2699, 2700
Linienbreite, 1103
Linienbreite setzen, 705, 1121
Liniensegment als Zeichenobjekt erzeugen, 1084
Linienstil, 1102
Linienstil setzen, 1120
list_files, 508
load_dl_model_weights, 613
local_max, 2450
local_max_contours_xld, 3037
local_max_sub_pix, 2451
local_min, 2452
local_min_max_funct_1d, 2630
local_min_sub_pix, 2453
local_threshold, 2437
Lochkamera, 1423
lock_mutex, 2554
log_image, 731
Logarithmus eines Bildes, 731
Logarithmus zur Basis 10 von Tupel-Elementen, 2868
Logarithmus zur Basis 2 von Tupel-Elementen, 2869
Lokale Maxima einer 1D-Funktion, 2630
lokale Maxima extrahieren (pixelgenau), 2450
lokale Maxima extrahieren (subpixelgenau), 2448, 2451
Lokale Minima einer 1D-Funktion, 2630
lokale Minima extrahieren (pixelgenau), 2452
lokale Minima extrahieren (subpixelgenau), 2448, 2453
lookup_lexicon, 2159
lowlands, 2454
lowlands_center, 2455
lut_trans, 964

make_dir, 509
map_image, 809
Marker und 3D-Pose von Kalibrierplatte finden, 1406
match_essential_matrix_ransac, 273
match_funct_1d_trans, 2630
match_fundamental_matrix_distortion_ransac, 277
match_fundamental_matrix_ransac, 281
match_rel_pose_ransac, 284
Matching-Modell (deskriptorbasiert) einlesen, 1832
Matching-Modell (deskriptorbasiert) löschen, 1818
Matching-Modell (deskriptorbasiert) prüfen, 1829–1831
Matching-Modell (deskriptorbasiert) schreiben, 1834
Matching-Modell (deskriptorbasiert) trainieren, 1819, 1820
Matching-Modell (deskriptorbasiert) wiederverwenden, 1832, 1834
Matching-Modell (formbasiert), 1916
Matching-Modell (formbasiert) einlesen, 1835, 1865, 1918
Matching-Modell (formbasiert) löschen, 1836
Matching-Modell (formbasiert) prüfen, 1856, 1914–1916
Matching-Modell (formbasiert) schreiben, 1918, 1938
Matching-Modell (formbasiert) trainieren, 1837, 1842, 1847, 1852, 1856, 1861, 1918, 1938
Matching-Modell (formbasiert) wiederverwenden, 1835, 1865, 1918, 1938
Matching-Modell (grauwertbasiert) einlesen, 1679
Matching-Modell (grauwertbasiert) schreiben, 1682
Matching-Modell (grauwertbasiert) trainieren, 1674, 1675
Matching-Modell (grauwertbasiert) wiederverwenden, 1679, 1682
Matching-Modell (komponentenbasiert) einlesen, 1949, 1972
Matching-Modell (komponentenbasiert) löschen, 1939
Matching-Modell (komponentenbasiert) prüfen, 1959, 1962, 1966
Matching-Modell (komponentenbasiert) schreiben, 1979
Matching-Modell (komponentenbasiert) trainieren, 1940, 1942, 1946, 1956, 1968, 1970, 1974, 1979
Matching-Modell (komponentenbasiert) wiederverwenden, 1949, 1972, 1980
Matching-Modell (korrelationsbasiert) einlesen, 1996
Matching-Modell (korrelationsbasiert) löschen, 1980
Matching-Modell (korrelationsbasiert) prüfen, 1994, 1995
Matching-Modell (korrelationsbasiert) schreiben, 1998
Matching-Modell (korrelationsbasiert) trainieren, 1981
Matching-Modell (korrelationsbasiert) wiederverwenden, 1996, 1998
Matching-Modell (lokal deformierbar) einlesen, 1810
Matching-Modell (lokal deformierbar) prüfen, 1807, 1809

Matching-Modell (lokal deformierbar) trainieren, 1813
Matching-Modell (lokal deformierbar) wiederverwenden, 1810
Matching-Modell (perspektivisch deformierbar) ansehen, 1807–1809
Matching-Modell (perspektivisch deformierbar) einlesen, 1810
Matching-Modell (perspektivisch deformierbar) löschen, 1777
Matching-Modell (perspektivisch deformierbar) trainieren, 1777, 1780, 1782, 1784, 1787, 1791, 1813–1816
Matching-Modell (perspektivisch deformierbar) wiederverwenden, 1810, 1818
Matching-Modell (punktbasiert) trainieren, 944, 947
Matching-Modell finden (deskriptorbasiert), 1824, 1826
Matching-Modell finden (formbasiert), 1847, 1868, 1874, 1882, 1883, 1889, 1897, 1902, 1909, 1910, 1912, 1919, 1920, 1937
Matching-Modell finden (grauwertbasiert), 1666, 1667, 1669–1671, 1677, 1678
Matching-Modell finden (komponentenbasiert), 1950
Matching-Modell finden (korrelationsbasiert), 1985, 1989
Matching-Modell finden (perspektivisch deformierbar), 1798, 1801, 1803
Matching-Modell finden (punktbasiert), 816, 2719, 2724, 2727, 2731, 2734
Matching-Modell Matching (grauwertbasiert) löschen, 1673
Matching-Modell perspektivisch (perspektivisch deformierbar) schreiben, 1818
Matching-Modell trainieren, 412, 1913, 1930, 1936, 1997, 2974
Matrix aufspalten, 2051, 2053
Matrix einlesen, 2032
Matrix erzeugen, 2041, 2042
Matrix invertieren, 2006, 2008
Matrix kopieren, 2040
Matrix löschen, 2039
Matrix schreiben, 2033
Matrix skalieren, 2022, 2023
Matrix transponieren, 2030, 2031
Matrixelemente, 2059, 2061
Matrixelemente multiplizieren, 2010, 2011
Matrixelemente setzen, 2064, 2066
Matrixgleichung lösen, 2024
Matrixgröße, 2044
Matrizen addieren, 2002, 2003
Matrizen elementweise dividieren, 2004, 2005
Matrizen multiplizieren, 2012, 2014
Matrizen subtrahieren, 2028, 2029
Mausinteraktion, 1067–1069, 1147
Mausposition, 1067, 1068
Mauszeigerform, 1071
Mauszeigerform setzen, 1075
max_diameter_object_model_3d, 182
max_image, 732
max_matrix, 2045
max_parallels_xld, 3038
maximaler Durchmesser einer Region, 2308
maximaler Durchmesser einer XLD-Kontur, 3007
Maximum einer Matrix, 2045
Maximum von Bildern, 732
Maximum von Tupel-Element, 2931
mean_curvature_flow, 984
mean_image, 845
mean_matrix, 2046
mean_n, 847
mean_sp, 847
measure_pairs, 15
measure_pos, 17
measure_profile_sheet_of_light, 355
measure_projection, 19
measure_thresh, 19
Median von Tupel-Elementen, 2932
median_image, 849
median_rect, 851
median_separate, 853
median_weighted, 854
mehrere Farben setzen, 703, 1115
mehrere Kameras kalibrieren, 1423
mehrkanaliges Bild, 785, 786, 788
mehrkanaliges Bild erzeugen, 409, 410, 415, 428–433
merge_cont_line_scan_xld, 3062
merge_regions_line_scan, 2364
Merkmale (kNN) für optische Schriftzeichenerkennung (OCR), 2150
Merkmale (MLP) für optische Schriftzeichenerkennung (OCR), 2170
Merkmale (SVM) für optische Schriftzeichenerkennung (OCR), 2215
Merkmale für Blob-Analyse extrahieren, 448, 452, 464–466, 470, 2300, 2304, 2308–2310, 2317–2319, 2345, 2346
Merkmale für die Klassifizierung (GMM) automatisch auswählen, 1505
Merkmale für die Klassifizierung (kNN) automatisch auswählen, 1520
Merkmale für die Klassifizierung (MLP) automatisch auswählen, 1551
Merkmale für die Klassifizierung (SVM) automatisch auswählen, 1593
Merkmale für optische Schriftzeichenerkennung (OCR), 1719
Merkmale von XLD-Konturen extrahieren, 3003, 3005–3007, 3013, 3035, 3043, 3044, 3047, 3052–3054
Merkmalssatz für optische Schriftzeichenerkennung (OCR) automatisch auswählen (kNN), 2152
Merkmalssatz für optische Schriftzeichenerkennung (OCR) automatisch auswählen (MLP), 2175, 2177

Merkmalssatz für optische Schriftzeichenerkennung (OCR) automatisch auswählen (SVM), 2222, 2223
Merkmalsvektor evaluieren (MLP), 1542
Merkmalsvektoren (GMM), 1500
Merkmalsvektoren (MLP), 1545
Merkmalsvektoren (SVM), 1585
messen (1D-Vermessung), 4, 6, 8, 15, 17, 23
Messmodell, 49, 52, 53, 65, 67, 68, 76
Messmodell erzeugen, 52
Messobjekt einlesen, 21
Messobjekt für 1D-Vermessung erzeugen, 10, 13, 27, 1139
Messobjekt löschen, 3
Messobjekt schreiben, 28
Messobjekt wiederverwenden, 27
Metadaten einer Bilddatei einlesen, 493
Metadaten einer Bilddatei schreiben, 500
Metrik eines Matching-Modells (formbasiert) setzen, 1933
Metrology-Messobjekt, 34, 36, 39, 42, 44, 46, 50, 51, 54–60, 62, 64, 66, 69, 71, 73
midrange_image, 855
min_image, 733
min_matrix, 2047
min_max_gray, 470
minimaler Abstand von Regionen, 2657
minimaler Abstand von XLD-Konturen, 2647, 2648
Minimum einer Matrix, 2047
Minimum von Bildern, 733
Minkowski-Addition, 2113, 2114
Minkowski-Subtraktion, 2116, 2117
minkowski_add1, 2113
minkowski_add2, 2114
minkowski_sub1, 2116
minkowski_sub2, 2117
mirror_image, 810
mirror_region, 2280
mit Grauwertschwellwert messen (1D-Vermessung), 19
Mittelwert einer Matrix, 2046
Mittelwert von 3D-Posen berechnen, 2828
Mittelwert von Tupel-Elementen, 2931
Mittelwert-Filter in Frequenzbereich erzeugen, 766
MLP-Klassifikator einlesen, 1542, 1549
MLP-Klassifikator erzeugen, 1537, 1551
MLP-Klassifikator löschen, 1536
MLP-Klassifikator schreiben, 1563
MLP-Klassifikator trainieren, 1534, 1536, 1548, 1561, 2399
MLP-Klassifikator wiederverwenden, 1542, 1544, 1549, 1563
mod_parallels_xld, 2984
Modell (Automatic Text Reader) erzeugen, 2187
Modell (Automatic Text Reader) löschen, 2186
Modell (Manual Text Finder) erzeugen, 2187
Modell (Manual Text Finder) löschen, 2186
Modell anwenden (Deep Learning), 628
Modell einlesen (Deep Learning), 647
Modell für strukturiertes Licht einlesen, 1314
Modell für strukturiertes Licht erzeugen, 1304
Modell für strukturiertes Licht löschen, 1303
Modell für strukturiertes Licht schreiben, 1319
Modell löschen (Deep Learning), 630
Modell schreiben (Deep Learning), 659
Modell trainieren (Deep Learning), 657
Modell zur Kleberaupeninspektion anwenden, 1289
Modell zur Kleberaupeninspektion erzeugen, 1291
Modell zur Kleberaupeninspektion löschen, 1290
Modell-Parameter (Automatic Text Reader) setzen, 2200
Modell-Parameter (Manual Text Finder) setzen, 2200
Modell-Parameter erhalten (Automatic Text Reader), 2190
Modell-Parameter erhalten (Manual Text Finder), 2190
Modellkomponenten (komponentenbasiert), 1966
Modellparameter abfragen (Deep Learning), 634, 2136
Modellparameter für 3D-Matching (deformierbar oberflächenbasiert) erhalten, 93
Modellparameter für 3D-Matching (formbasiert) erhalten, 113
Modellparameter für 3D-Matching (oberflächenbasiert) erhalten, 137
Modellparameter für 3D-Modell, 177
Modellparameter für deformierbares oberflächenbasiertes 3D-Matching setzen, 82
Modellparameter setzen (Deep Learning), 653
Modifikationsausdruck ausführen, 2475
modify_component_relations, 1970
Momente einer Region, 2320–2326
Momente einer XLD-Kontur, 3039, 3040, 3042, 3043
moments_any_points_xld, 3039
moments_any_xld, 3040
moments_gray_plane, 472
moments_object_model_3d, 183
moments_points_xld, 3042
moments_region_2nd, 2320
moments_region_2nd_invar, 2321
moments_region_2nd_rel_invar, 2322
moments_region_3rd, 2323
moments_region_3rd_invar, 2324
moments_region_central, 2325
moments_region_central_invar, 2326
moments_xld, 3043
Monotonie von Grauwerten, 929
monotony, 929
morph_hat, 1697
morph_skeleton, 1698
morph_skiz, 1699
Mosaikbild erzeugen, 2716, 2719
Mosaikbilder mit Bündelausgleich, 2713
move_rectangle, 1637
move_region, 2281
mult_element_matrix, 2010

mult_element_matrix_mod, 2011
mult_image, 734
mult_matrix, 2012
mult_matrix_mod, 2014
Multithreading-Attribute, 2552
Muster-Bilder für Verfahren mit strukturiertem Licht, 1305, 1307
Musterdeformation zu deformierbarem Oberflächenmodell hinzufügen, 83
Mutex erzeugen (Multithreading), 2543
Mutex freigeben (Multithreading), 2563
Mutex löschen (Multithreading), 2537
Mutex sperren (Multithreading), 2554, 2562

nächste Punkte einer Region, 2355
Nachricht an eine Warteschlange senden (Inter-Thread Kommunikation), 2545
Nachricht aus einer Warteschlange empfangen (Inter-Thread Kommunikation), 2544
Nachricht lesen, 2554
Nachricht schreiben, 2565
Nachrichtenobjekt erzeugen (Inter-Thread Kommunikation), 2541
Nachrichtenobjekt löschen (Inter-Thread Kommunikation), 2536
Nachschlagetabelle (Look-Up-Tabelle, LUT) des Grafikfensters, 1039
Nachschlagetabelle (Look-Up-Tabelle, LUT) des Grafikfensters schreiben, 1652
Nachschlagetabelle (Look-Up-Tabelle, LUT) eines Grafikfensters setzen, 1040
Nachschlagetabelle (Look-Up-Tabelle, LUT) für Bildeinzug, 373
Nachschlagetabelle (Look-Up-Tabelle, LUT) für Bildeinzug setzen, 387
Nachschlagetabelle (Look-Up-Tabelle, LUT) für Farbraumtransformationen, 781
Nachschlagetabelle (Look-Up-Tabelle, LUT) für Farbraumtransformationen erzeugen, 784
Nachschlagetabelle (Look-Up-Tabelle, LUT) für Farbraumtransformationen löschen, 784
Nachschlagetabelle (Look-Up-Tabelle, LUT) setzen, 706
Nachschlagetabelle (Look-Up-Tabelle, LUT) von Grafikfenster anzeigen, 1627
Nachschlagetabelle (LUT) aus Klassifikator erzeugen, 1526, 1528, 1530, 1531
natürlicher Logarithmus von Tupel-Elementen, 2868
negate_funct_1d, 2632
new_extern_window, 1054
new_line, 1137
noise_distribution_mean, 956
nonmax_suppression_amp, 2395
nonmax_suppression_dir, 2396
Norm einer Matrix, 2048
norm_matrix, 2048
normalisierte Fuzzy-Messfunktion setzen, 25
Nulldurchgänge einer 1D-Funktion, 2638
Nulldurchgänge extrahieren, 2446
num_points_funct_1d, 2632
NURBS-Kurve ändern, 1158, 1161
NURBS-Kurve zeichnen, 1154, 1156

Oberfläche rekonstruieren, 334, 336, 337
Oberflächen-Albedo schätzen, 327
obj_diff, 2247
obj_to_integer, 2248
object_model_3d_to_xyz, 208
Objekt aus Objekttupel auswählen, 2250
Objektinstanz aus 3D-Szene entfernen, 1007
Objektivverzeichnung einer Abbildung ändern, 1471
Objektivverzeichnung eines Bildes ändern, 1466
Objektivverzeichnung selbstkalibrieren, 1476
Objektivverzeichnung von Kameraparametern ändern, 1463
Objektivverzeichnung von Pixelkoordinaten ändern, 1467
Objektivverzeichnung von XLD-Konturen ändern, 1465
OCR-Klassifikator (CNN) einlesen, 2125, 2131
OCR-Klassifikator (CNN) löschen, 2125
OCR-Klassifikator (CNN) schreiben, 2132
OCR-Klassifikator (CNN) wiederverwenden, 2125, 2131, 2132
OCR-Klassifikator (Hyperbox) einlesen, 1715, 1719
OCR-Klassifikator (Hyperbox) löschen, 1711
OCR-Klassifikator (Hyperbox) schreiben, 1720, 1723
OCR-Klassifikator (Hyperbox) wiederverwenden, 1715, 1719, 1720, 1723
OCR-Klassifikator (kNN) einlesen, 2152
OCR-Klassifikator (kNN) löschen, 2142
OCR-Klassifikator (kNN) schreiben, 2156
OCR-Klassifikator (MLP) einlesen, 2165, 2174
OCR-Klassifikator (MLP) löschen, 2161
OCR-Klassifikator (MLP) schreiben, 2185
OCR-Klassifikator (MLP) wiederverwenden, 2165, 2174, 2185
OCR-Klassifikator (SVM) einlesen, 2211, 2220
OCR-Klassifikator (SVM) löschen, 2207
OCR-Klassifikator (SVM) reduzieren, 2221
OCR-Klassifikator (SVM) schreiben, 2225, 2228
OCR-Klassifikator (SVM) wiederverwenden, 2211, 2220, 2225, 2228
ocr_change_char, 1718
ocr_get_features, 1719
open_compute_device, 2497
open_file, 534
open_framegrabber, 383
open_io_channel, 2516
open_io_device, 2517
open_serial, 2601
open_socket_accept, 2607
open_socket_connect, 2610
open_textwindow, 1639
open_window, 1056
opening, 2119

Opening Operator, 1700, 1702, 2084–2086, 2119–2121
Opening und Closing Operator, 1689
opening_circle, 2120
opening_golay, 1700
opening_rectangle1, 2121
opening_seg, 1702
Operatoren auswählen, 2527, 2533
Operatoren eines Kapitels, 2524
optical_flow_mg, 931
Optimierungsdaten für schnelle Fourier-Transformation (FFT) schreiben, 780, 781
optimize_aop, 2567
optimize_dl_model_for_inference, 548
optimize_fft_speed, 770
optimize_rft_speed, 771
optische Schriftzeichenerkennung (OCR) trainieren, 1139
optische Schriftzeichenerkennung (OCR) trainieren (Hyperbox), 1712, 1721, 1722
optische Schriftzeichenerkennung (OCR) trainieren (kNN), 2142, 2155
optische Schriftzeichenerkennung (OCR) trainieren (MLP), 2161, 2171, 2182, 2184, 2185
optische Schriftzeichenerkennung (OCR) trainieren (SVM), 2208, 2217, 2225–2228
optische Schriftzeichenkontrolle (OCV) durchführen, 1298
optischer Fluss, 931
orientation_points_xld, 3044
orientation_region, 2327
orientation_xld, 3044
Orientierung einer Textzeile, 2205
Orientierung und Radius der äquivalenten Ellipse einer Region, 452, 2310
Orientierung und Radius der äquivalenten Ellipse einer XLD-Kontur, 3014, 3015
orthogonal_decompose_matrix, 2053
overpaint_gray, 441
overpaint_region, 442

paint_gray, 443
paint_region, 444
paint_xld, 445
par_join, 2482
parallele Ausführung in HDevelop, 2482
Parallele einer XLD-Kontur, 2987
parallele Projektion, 1423
parallele XLD-Linien, 3090
Parameter der Rückweisungsklasse eines MLP, 1547
Parameter der Rückweisungsklasse eines MLP setzen, 1559
Parameter der Rückweisungsklasse für einen OCR-Klassifikator (MLP), 2174
Parameter der Rückweisungsklasse für einen OCR-Klassifikator setzen (MLP), 2182
Parameter der XLD-Distanztransformation, 2662
Parameter des Hintergrundschätzers, 2688
Parameter des Kleberaupeninspektionsmodells, 1293
Parameter des Zeichenobjekts, 1091
Parameter einer I/O-Geräteinstanz, 2516
Parameter einer Nachricht (Inter-Thread Kommunikation), 2548
Parameter einer Nachricht setzen (Inter-Thread Kommunikation), 2556
Parameter einer Nachschlagetabelle (Look-Up-Tabelle, LUT) eines Grafikfensters setzen, 1648
Parameter einer seriellen Schnittstelle, 2600
Parameter einer seriellen Schnittstelle setzen, 2603
Parameter einer Warteschlange für Nachrichten, 2550
Parameter einer Warteschlange für Nachrichten setzen, 2557
Parameter eines 3D-Matching-Modells (oberflächenbasiert) setzen, 144
Parameter eines Barcode-Modells setzen, 1200, 1209
Parameter eines Compute Devices, 2496
Parameter eines Compute Devices setzen, 2500
Parameter eines Datacode-Modells, 1229
Parameter eines Datacode-Modells setzen, 1253
Parameter eines Hintergrundschätzers setzen, 2692
Parameter eines Kameraaufbaumodells, 1438
Parameter eines Kameraaufbaumodells setzen, 1453, 1454
Parameter eines Lichtschnittmodells, 351
Parameter eines Lichtschnittmodells setzen, 358, 360
Parameter eines Matching-Modells (deskriptorbasiert), 1829
Parameter eines Matching-Modells (formbasiert), 1913, 1915
Parameter eines Matching-Modells (formbasiert) setzen, 1930, 1936
Parameter eines Matching-Modells (komponentenbasiert), 1959
Parameter eines Matching-Modells (korrelationsbasiert), 1994
Parameter eines Matching-Modells (korrelationsbasiert) setzen, 1997
Parameter eines Matching-Modells (lokal deformierbar), 1809
Parameter eines Matching-Modells (perspektivisch deformierbar), 1809
Parameter eines Modells für strukturiertes Licht, 1311, 1316
Parameter eines OCR-Klassifikators (CNN), 2130
Parameter eines OCR-Klassifikators (Hyperbox), 1717
Parameter eines OCR-Klassifikators (kNN), 2151
Parameter eines OCR-Klassifikators (MLP), 2171, 2173
Parameter eines OCR-Klassifikators (MLP) setzen, 2179
Parameter eines OCR-Klassifikators (SVM), 2216
Parameter eines Texturinspektionsmodells, 1332, 1339

Parameter für 3D-Szene setzen, 1015
Parameter für Bildeinzug setzen, 388
Parameter für das Kleberaupeninspektionsmodell setzen, 1294
Parameter für I/O-Übertragungskanal lesen, 2521
Parameter für I/O-Übertragungskanal setzen, 2521
Parameter für I/O-Gerät setzen, 2522
Parameter für Lichtquelle in 3D-Szene setzen, 1015
Parameter für Nachschlagetabelle (Look-Up-Tabelle, LUT) des Grafikfensters, 1630
Parameter für Objektinstanz in 3D-Szene setzen, 1010
Parameter für Textlabel in 3D-Szene setzen, 1013
Parameter für XLD-Distanztransformation setzen, 2677
Parameter für Zeichenobjekt setzen, 1095
Parameter oder Informationen eines Dictionaries, 2886
Parameter von Barcode-Modell, 1186, 1188, 1197
Parameter von I/O-Übertragungskanälen, 2515
Parameter-Anzeigefenster anzeigen, 1619
Parameterfenster verbergen, 1621
Parameternamen, 2529
Parametertypen, 2531
partition_dynamic, 2366
partition_lines, 1766
partition_rectangle, 2367
passende Linien für Region, 2702
perspektivische Projektion, 1423
Pfad zum Bildrand, 962
Pfeil anzeigen, 1019
Phase eines komplexen Bildes, 773, 774
phase_correlation_fft, 772
phase_deg, 773
phase_rad, 774
photometric_stereo, 329
Pixel der Regionenkontur, 2375
Pixel einer Region, 2377
Pixelkanten verdünnen, 2395, 2396
Pixelklassifizierung, 2401, 2403
Pixelkoordinaten in 3D-Koordinaten transformieren, 1472
Pixelkoordinaten rektifizieren, 1467
pixelweises Maximum von Bildern, 732
pixelweises Minimum von Bildern, 733
plane_deviation, 473
plateaus, 2456
plateaus_center, 2458
Plotbild, 1636
Plotfunktion, 1635
point_line_to_hom_mat2d, 2767
points_foerstner, 944
points_harris, 947
points_harris_binomial, 949
points_lepetit, 950
points_sojka, 951
polar_trans_contour_xld, 2988
polar_trans_contour_xld_inv, 2990
polar_trans_image, 1623
polar_trans_image_ext, 811
polar_trans_image_inv, 813
polar_trans_region, 2281
polar_trans_region_inv, 2284
polare Transformation einer Region, 2281
polare Transformation einer XLD-Kontur, 2988
polare Transformation eines Bildes, 811
Polygon anzeigen, 1033
Polygon-Region erzeugen, 2273, 2274
Polygonenapproximation von Regionenkontur, 2378
Polygonzug zeichnen, 1165
Polynomialmodell der Objektivverzeichnung, 1423
Pose eines 3D-Matching-Modells (oberflächenbasiert) verfeinern, 140, 142
Pose für 3D-Szene im Weltkoordinatensystem festlegen, 1017
Pose für Objektinstanz in 3D-Szene setzen, 1013
Pose invertieren, 2830
pose_average, 2828
pose_compose, 2829
pose_invert, 2830
pose_to_dual_quat, 2831
pose_to_hom_mat3d, 2800
pose_to_quat, 2832
Position des Grafikfensters setzen, 712
Potenzfunktion eines Bildes, 735
Potenzfunktion von Matrix, 2018–2021
Potenzfunktion von Matrizen, 2016, 2017
Potenzfunktion von Tupeln, 2873
pouring, 2459
pow_element_matrix, 2016
pow_element_matrix_mod, 2017
pow_image, 735
pow_matrix, 2018
pow_matrix_mod, 2019
pow_scalar_element_matrix, 2020
pow_scalar_element_matrix_mod, 2021
power_byte, 775
power_ln, 776
power_real, 777
Powerspektrum eines komplexen Bildes, 775–777
prepare_direct_variation_model, 1353
prepare_object_model_3d, 210
prepare_sample_identifier, 1278
prepare_variation_model, 1355
Prewitt-Filter, 906, 907
prewitt_amp, 906
prewitt_dir, 907
principal_comp, 788
Programm abbrechen, 2475
Programmfenster anzeigen, 1620
Programmfenster verbergen, 1621
proj_hom_mat2d_to_pose, 2832
proj_match_points_distortion_ransac, 2724
proj_match_points_distortion_ransac_guided, 2727
proj_match_points_ransac, 2731

proj_match_points_ransac_guided, 2734
project_3d_point, 1460
project_hom_point_hom_mat3d, 1461
project_object_model_3d, 213
project_point_hom_mat3d, 1462
project_shape_model_3d, 115
projection_pl, 2674
projective_trans_contour_xld, 2992
projective_trans_hom_point_3d, 2801
projective_trans_image, 815
projective_trans_image_size, 816
projective_trans_object_model_3d, 215
projective_trans_pixel, 2770
projective_trans_point_2d, 2771
projective_trans_point_3d, 2802
projective_trans_region, 2286
projektive 2D-Matrix aus Punktkorrespondenzen, 2765
projektive 2D-Matrix aus RANSAC-Punktmatching, 2724, 2727, 2731, 2734
projektive 2D-Matrix in 3D-Pose umrechnen, 2832
projektive 2D-Transformation einer Region, 2286
projektive 2D-Transformation einer XLD-Kontur, 2992
projektive 2D-Transformation eines Bildes, 815, 816
projektive 2D-Transformation eines Pixels, 2770
projektive 2D-Transformation eines Punktes, 2771
projektive 2D-Transformation von Punktkorrespondenzen, 2778, 2780
projektive 3D-Information mittels binoklularem Stereo rekonstruieren, 287
projektive 3D-Transformation eines Punktes, 2801, 2802
projektive 3D-Transformation von Punktkorrespondenzen, 2804
projektive Kameraparameter selbstkalibrieren, 1482
projektiver 3D-Punkt aus Fundamental-Matrix, 287
protect_ocr_trainf, 2231
Prozedur abbrechen, 2483
Prozedur in HDevelop importieren, 2480
Prozeduren auswählen, 2527, 2533
Prozeduren eines Kapitels, 2524
pruning, 2122
Punkt ändern, 1164
Punkt zeichnen, 1163
Punkte einer XLD-Kontur, 3088
Punkte einer XLD-Kontur zählen, 3005
Punkte eines Entzerrungsgitters verbinden, 2679
Punkte eines Matching-Modells (deskriptorbasiert), 1830
Punkte verbessern, 943
Punktextraktion, 944, 947, 949, 950
Punktkorrespondenzen für Kamerakalibrierung, 1451
punktweiser Abstand zweier Konturen, 2641
punktweiser Abstand zwischen Konturen, 2649
Quadratwurzel eines Bildes, 738
Quadratwurzel von Matrixelementen, 2026, 2027
Quadratwurzel von Tupel-Elementen, 2876
quat_compose, 2841
quat_conjugate, 2842
quat_interpolate, 2843
quat_normalize, 2843
quat_rotate_point_3d, 2844
quat_to_hom_mat3d, 2845
quat_to_pose, 2833
Quaternion in 3D-Pose umrechnen, 2833
Quaternion in homogene 3D-Matrix umrechnen, 2845
Quaternion interpolieren, 2843
Quaternion konjugieren, 2842
Quaternion normalisieren, 2843
Quaternionen multiplizieren, 2841
query_all_colors, 1108
query_aop_info, 2570
query_available_compute_devices, 2498
query_available_dl_devices, 549
query_bar_code_params, 1197
query_calib_data_observ_indices, 1439
query_color, 1109
query_colored, 1110
query_contour_attribs_xld, 3045
query_contour_global_attribs_xld, 3046
query_data_code_2d_params, 1250
query_font, 1138
query_gray, 1110
query_insert, 1642
query_io_device, 2519
query_io_interface, 2520
query_line_width, 1111
query_lut, 1039
query_mshape, 1071
query_operator_info, 2532
query_paint, 1112
query_param_info, 2532
query_params_ocr_class_cnn, 2131
query_shape, 1112
query_sheet_of_light_params, 356
query_spy, 2508
query_tshape, 1643
query_window_type, 1060

radial_distortion_self_calibration, 1476
radiometric_self_calibration, 1479
Radiometrische Kalibrierung, 1479
rank_image, 856
rank_n, 858
rank_rect, 859
rank_region, 2367
RANSAC-Algorithmus, 1824, 1826
Rauschen aus Region entfernen, 2369

Rauschen schätzen, 454
Rauschen zur XLD-Kontur hinzufügen, 3057
Rauschverteilung hinzufügen, 953
read_aop_knowledge, 2571
read_bar_code_model, 1199
read_calib_data, 1441
read_cam_par, 1417
read_camera_setup_model, 1441
read_char, 1139
read_class_box, 1660
read_class_gmm, 1503
read_class_knn, 1519
read_class_mlp, 1549
read_class_svm, 1590
read_class_train_data, 1569
read_component_model, 1972
read_contour_xld_arc_info, 517
read_contour_xld_dxf, 517
read_data_code_2d_model, 1251
read_deep_ocr, 2139
read_deformable_model, 1810
read_deformable_surface_model, 94
read_descriptor_model, 1832
read_dict, 2889
read_distance_transform_xld, 2676
read_dl_classifier, 1610
read_dl_model, 647
read_fft_optimization_data, 778
read_funct_1d, 2632
read_gray_se, 2089
read_image, 491
read_image_metadata, 493
read_io_channel, 2521
read_kalman, 1767
read_matrix, 2032
read_measure, 21
read_message, 2554
read_metrology_model, 65
read_ncc_model, 1996
read_object, 501
read_object_model_3d, 158
read_ocr, 1719
read_ocr_class_cnn, 2131
read_ocr_class_knn, 2152
read_ocr_class_mlp, 2174
read_ocr_class_svm, 2220
read_ocr_trainf, 2232
read_ocr_trainf_names, 2232
read_ocr_trainf_names_protected, 2233
read_ocr_trainf_select, 2234
read_ocv, 1299
read_polygon_xld_arc_info, 519
read_polygon_xld_dxf, 520
read_pose, 2834
read_region, 504
read_sample_identifier, 1279
read_samples_class_gmm, 1504
read_samples_class_mlp, 1550
read_samples_class_svm, 1590
read_sampset, 1661
read_sequence, 494
read_serial, 2602
read_shape_model, 1918
read_shape_model_3d, 117
read_sheet_of_light_model, 357
read_string, 1139
read_structured_light_model, 1314
read_surface_model, 139
read_template, 1679
read_texture_inspection_model, 1335
read_training_components, 1972
read_tuple, 512
read_variation_model, 1356
read_world_file, 509
real_to_complex, 479
real_to_vector_field, 480
realwertige schnelle Fourier-Transformation (FFT), 779
receive_data, 2611
receive_image, 2612
receive_region, 2612
receive_serialized_item, 2613
receive_tuple, 2614
receive_xld, 2614
Rechteck ändern, 1167
Rechteck als Zeichenobjekt erzeugen, 1086
Rechteck anzeigen, 1034, 1036
Rechteck zeichnen, 1166, 1168, 1169
rechteckige Region erzeugen, 2266, 2267
Rechteckigkeit einer Region, 2328
Rechteckigkeit einer XLD-Kontur, 3047
reconst3d_from_fundamental_matrix, 287
reconstruct_height_field_from_gradient, 333
reconstruct_points_stereo, 309
reconstruct_surface_stereo, 311
rectangle1_domain, 392
rectangularity, 2328
rectangularity_xld, 3047
reduce_class_svm, 1591
reduce_domain, 393
reduce_object_model_3d_by_view, 192
reduce_ocr_class_svm, 2221
reduzierte Support-Vektor-Maschine (SVM), 1591
reell-wertige schnelle Fourier-Transformation (FFT) optimieren, 771
Referenz für Modell (grauwertbasiert) setzen, 1682
Referenzkontur der XLD-Distanztransformation, 2662
refine_deformable_surface_model, 95
refine_surface_model_pose, 140
refine_surface_model_pose_image, 142
Reflexionskoeffizient (Albedo) schätzen, 326
Region abschneiden, 2352, 2353
Region an Socket senden, 2617
Region anzeigen, 686, 1028, 1037

Region aus Geraden erzeugen, 2261
Region aus Hessescher Normalform erzeugen, 2270
Region aus Histogramm erzeugen, 2269
Region aus Lauflängen erzeugen, 2276
Region aus Pixeln erzeugen, 2261, 2272
Region aus XLD-Kontur erzeugen, 2268
Region aus XLD-Polygon erzeugen, 2275
Region einlesen, 504
Region einzeichnen, 442, 444
Region enthält Pixel, 2350
Region enthält Region, 2350
Region erzeugen, 2104
Region für Blob-Analyse bearbeiten, 463, 474, 2096, 2097, 2099, 2119–2121
Region interaktiv verschieben, 1144–1146
Region of Interest, 391
Region schreiben, 505
Region spiegeln, 2280
Region transformieren um Hamming-Abstand zu erhalten, 1725
Region transponieren, 2287
Region um Punkt spiegeln, 2287
Region verdünnen, 1699, 1707–1709
Region vergrößern, 1730, 2360
Region vergrößern (Thickening), 1703, 1705, 1706
Region verschieben, 2281
Region von Socket einlesen, 2612
Region zeichnen, 1147, 1148, 1170–1172
Region zerlegen, 2366, 2367
region_features, 2329
region_to_bin, 424
region_to_label, 425
region_to_mean, 426
Regionen aus Zeilenkamera-Bild verbinden, 2364
Regionen auswählen, 463, 474, 2312, 2313, 2329, 2335–2337, 2340, 2342, 2347
Regionen für Blob-Analyse bearbeiten, 463, 474, 2096, 2097, 2099, 2119–2121, 2290–2293, 2352, 2357, 2362, 2366, 2367, 2370–2372
Regionen mit Rangmaske glätten, 2367
Regionen sind gleich, 2349
Regionen sortieren, 2372
Regionen verbinden, 2292, 2293
Regionengröße ändern (skalieren, Zoom), 2288
regiongrowing, 2385
regiongrowing_mean, 2387
regiongrowing_n, 2388
register_object_model_3d_global, 216
register_object_model_3d_pair, 218
Registrierung von 3D-Objektmodellen, 216, 218
regress_contours_xld, 3063
Regressionsparameter einer XLD-Kontur, 3063
Regressionsparameter von XLD-Kontur, 3033
Rektifizierung, 1463, 1471
rel_pose_to_fundamental_matrix, 288
relative Kameralage aus Punktkorrespondenzen, 297
relative Kameralage aus RANSAC-Punktmatching, 284
release_all_compute_devices, 2499
release_compute_device, 2500
remove_calib_data, 1442
remove_calib_data_observ, 1442
remove_dict_key, 2890
remove_dir, 510
remove_noise_region, 2369
remove_obj, 2249
remove_object_model_3d_attrib, 162
remove_object_model_3d_attrib_mod, 163
remove_sample_identifier_preparation_data, 1280
remove_sample_identifier_training_data, 1281
remove_scene_3d_camera, 1007
remove_scene_3d_instance, 1007
remove_scene_3d_label, 1008
remove_scene_3d_light, 1008
remove_texture_inspection_model_image, 1336
render_object_model_3d, 220
render_scene_3d, 1009
repeat, 2483
repeat_matrix, 2042
replace_obj, 2249
reset_fuzzy_measure, 22
reset_metrology_object_fuzzy_param, 66
reset_metrology_object_param, 66
reset_obj_db, 2504
reset_sheet_of_light_model, 358
return, 2483
rft_generic, 779
RGB-Bild, 782
RGB-Farbraum, 1127, 1128
rgb1_to_gray, 788
rgb3_to_gray, 789
rigid_trans_object_model_3d, 221
roberts, 908
Roberts-Filter, 908
Robinson-Filter, 909, 911
robinson_amp, 909
robinson_dir, 911
rotate_image, 818
Rotationsquaternion erzeugen, 2840
roundness, 2332
run_bg_esti, 2691
Rundheit einer Region, 2302, 2332
Rundheit einer XLD-Kontur, 3004
runlength_distribution, 2333
runlength_features, 2334

saddle_points_sub_pix, 2461
Salz-und-Pfeffer-Rauschen, 836, 847
Salz-und-Pfeffer-Rauschen erzeugen, 956
sample_funct_1d, 2633
sample_object_model_3d, 222
Sattelpunkte extrahieren (subpixelgenau), 2448, 2461

scale_image, 736
scale_image_max, 986
scale_matrix, 2022
scale_matrix_mod, 2023
scale_y_funct_1d, 2634
scene_flow_calib, 967
scene_flow_uncalib, 969
schachbrettartige Region erzeugen, 2253
Schleife abbrechen, 2468
Schleife vom Start fortsetzen, 2471
schnelle Fourier-Transformation (FFT), 752, 754
schnelle Fourier-Transformation (FFT) optimieren, 770
schnelle Fourier-Transformations (FFT)-Optimierungsdaten einlesen, 750, 778
Schnitt eines 3D-Objektmodells mit einer Ebene, 207
Schraubungsparameter aus einer dualen Quaternion berechnen, 2812
Schriftart (Font), 1134
Schriftart (Font) setzen, 1140
Schwellwert aus Histogramm, 2436
Schwellwertbilder eines Variationsmodells, 1352
Schwerpunkt einer Region, 448, 2300
Schwerpunkt einer XLD-Kontur, 3002, 3003
Schwerpunkt von Grauwert-Hochebenen, 2458
Schwerpunkt von Grauwert-Tiefebenen, 2455
screw_to_dual_quat, 2814
search_operator, 2533
segment_characters, 2194
segment_contour_attrib_xld, 3065
segment_contours_xld, 3066
segment_image_mser, 2414
segment_object_model_3d, 194
Segmentierungsparameter für Blob-Analyse extrahieren, 464–466, 470, 2436
Selbstüberschneidung einer XLD-Kontur, 3055
select_characters, 2196
select_contours_xld, 3047
select_feature_set_gmm, 1505
select_feature_set_knn, 1520
select_feature_set_mlp, 1551
select_feature_set_svm, 1593
select_feature_set_trainf_knn, 2152
select_feature_set_trainf_mlp, 2175
select_feature_set_trainf_mlp_protected, 2177
select_feature_set_trainf_svm, 2222
select_feature_set_trainf_svm_protected, 2223
select_gray, 474
select_grayvalues_from_channels, 301
select_lines, 1770
select_lines_longest, 1771
select_matching_lines, 2702
select_obj, 2250
select_object_model_3d, 184
select_points_object_model_3d, 195
select_region_point, 2335
select_region_spatial, 2336
select_shape, 2337
select_shape_proto, 2340
select_shape_std, 2342
select_shape_xld, 3049
select_sub_feature_class_train_data, 1570
select_xld_point, 3051
Semantischer Tupeltyp, 2958
send_data, 2615
send_image, 2616
send_mouse_double_click_event, 1072
send_mouse_down_event, 1073
send_mouse_drag_event, 1074
send_mouse_up_event, 1074
send_region, 2617
send_serialized_item, 2617
send_tuple, 2618
send_xld, 2619
Serialisierung, 3, 22, 53, 67, 88, 96, 107, 117, 124, 143, 152, 164, 350, 359, 491, 497, 501–503, 505, 511–513, 516, 521, 631, 652, 1181, 1199, 1220, 1252, 1297, 1300, 1306, 1315, 1330, 1338, 1351, 1357, 1416, 1418, 1428, 1443, 1444, 1496, 1507, 1516, 1522, 1542, 1553, 1567, 1571, 1582, 1595, 1607, 1612, 1655, 1662, 1677, 1680, 1715, 1720, 1794, 1811, 1824, 1833, 1949, 1973, 1974, 1983, 1996, 2031, 2033, 2146, 2154, 2165, 2179, 2595–2598, 2613, 2617, 2744, 2772, 2786, 2803, 2807, 2816, 2822, 2835, 2841, 2845
serialize_bar_code_model, 1199
serialize_calib_data, 1443
serialize_cam_par, 1418
serialize_camera_setup_model, 1444
serialize_class_box, 1662
serialize_class_gmm, 1507
serialize_class_knn, 1522
serialize_class_mlp, 1553
serialize_class_svm, 1595
serialize_class_train_data, 1571
serialize_component_model, 1973
serialize_data_code_2d_model, 1252
serialize_deformable_model, 1811
serialize_deformable_surface_model, 96
serialize_descriptor_model, 1833
serialize_distance_transform_xld, 2676
serialize_dl_classifier, 1612
serialize_dl_model, 652
serialize_dual_quat, 2816
serialize_fft_optimization_data, 780
serialize_handle, 513
serialize_hom_mat2d, 2772
serialize_hom_mat3d, 2803
serialize_image, 497
serialize_matrix, 2033

serialize_measure, 22
serialize_metrology_model, 67
serialize_ncc_model, 1996
serialize_object, 502
serialize_object_model_3d, 164
serialize_ocr, 1720
serialize_ocr_class_cnn, 2132
serialize_ocr_class_knn, 2154
serialize_ocr_class_mlp, 2179
serialize_ocr_class_svm, 2225
serialize_ocv, 1300
serialize_pose, 2835
serialize_quat, 2845
serialize_region, 505
serialize_sample_identifier, 1282
serialize_shape_model, 1918
serialize_shape_model_3d, 117
serialize_sheet_of_light_model, 359
serialize_structured_light_model, 1315
serialize_surface_model, 143
serialize_template, 1680
serialize_texture_inspection_model, 1338
serialize_training_components, 1974
serialize_tuple, 513
serialize_variation_model, 1357
serialize_xld, 521
serielle Schnittstelle öffnen, 2601
serielle Schnittstelle schließen, 2600
set_aop_info, 2572
set_bar_code_param, 1200
set_bar_code_param_specific, 1209
set_bead_inspection_param, 1294
set_bg_esti_params, 2692
set_calib_data, 1445
set_calib_data_calib_object, 1448
set_calib_data_cam_param, 1449
set_calib_data_observ_points, 1451
set_calib_data_observ_pose, 1391
set_camera_setup_cam_param, 1453
set_camera_setup_param, 1454
set_check, 2508
set_class_box_param, 1663
set_color, 1113
set_colored, 1115
set_comprise, 1644
set_compute_device_param, 2500
set_content_update_callback, 1093
set_contour_style, 1116
set_current_dir, 510
set_data_code_2d_param, 1253
set_deep_ocr_param, 2140
set_deformable_model_origin, 1812
set_deformable_model_param, 1813
set_descriptor_model_origin, 1834
set_diagonal_matrix, 2062
set_dict_object, 2891
set_dict_tuple, 2892
set_distance_transform_xld_param, 2677
set_dl_classifier_param, 1613
set_dl_device_param, 551
set_dl_model_layer_param, 613
set_dl_model_layer_weights, 614
set_dl_model_param, 653
set_dl_pruning_param, 656
set_draw, 1117
set_drawing_object_callback, 1094
set_drawing_object_params, 1095
set_drawing_object_xld, 1097
set_feature_lengths_class_train_data, 1571
set_fix, 1645
set_fixed_lut, 1645
set_font, 1140
set_framegrabber_callback, 386
set_framegrabber_lut, 387
set_framegrabber_param, 388
set_full_matrix, 2064
set_fuzzy_measure, 23
set_fuzzy_measure_norm_pair, 25
set_generic_shape_model_object, 1919
set_generic_shape_model_param, 1920
set_gray, 1117
set_grayval, 447
set_hsi, 1118
set_icon, 1120
set_insert, 1646
set_io_channel_param, 2521
set_io_device_param, 2522
set_line_approx, 1647
set_line_style, 1120
set_line_width, 1121
set_local_deformable_model_metric, 1814
set_lut, 1040
set_lut_style, 1648
set_message_obj, 2555
set_message_param, 2556
set_message_queue_param, 2557
set_message_tuple, 2559
set_metrology_model_image_size, 68
set_metrology_model_param, 69
set_metrology_object_fuzzy_param, 71
set_metrology_object_param, 73
set_mshape, 1075
set_ncc_model_origin, 1997
set_ncc_model_param, 1997
set_object_model_3d_attrib, 165
set_object_model_3d_attrib_mod, 168
set_offset_template, 1681
set_operator_timeout, 2579
set_origin_pose, 2835
set_paint, 1122
set_params_class_knn, 1523
set_part, 1125
set_part_style, 1126

set_pixel, 1649
set_planar_calib_deformable_model_metric, 1815
set_planar_uncalib_deformable_model_metric, 1816
set_profile_sheet_of_light, 359
set_reference_template, 1682
set_regularization_params_class_mlp, 1554
set_regularization_params_ocr_class_mlp, 2179
set_rejection_params_class_mlp, 1559
set_rejection_params_ocr_class_mlp, 2182
set_rgb, 1127
set_rgba, 1128
set_sample_identifier_object_info, 1283
set_sample_identifier_param, 1283
set_scene_3d_camera_pose, 1009
set_scene_3d_instance_param, 1010
set_scene_3d_instance_pose, 1013
set_scene_3d_label_param, 1013
set_scene_3d_light_param, 1015
set_scene_3d_param, 1015
set_scene_3d_to_world_pose, 1017
set_serial_param, 2603
set_shape, 1129
set_shape_model_clutter, 1930
set_shape_model_metric, 1933
set_shape_model_origin, 1935
set_shape_model_param, 1936
set_sheet_of_light_param, 360
set_socket_param, 2620
set_spy, 2510
set_stereo_model_image_pairs, 318
set_stereo_model_param, 319
set_structured_light_model_param, 1316
set_sub_matrix, 2065
set_surface_model_param, 144
set_system, 2580
set_text_model_param, 2200
set_texture_inspection_model_param, 1339
set_tposition, 1141
set_tshape, 1650
set_value_matrix, 2066
set_window_attr, 1061
set_window_dc, 1061
set_window_extents, 1062
set_window_param, 1130
set_window_type, 1063
sfs_mod_lr, 334
sfs_orig_lr, 336
sfs_pentland, 337
shade_height_field, 338
shape_histo_all, 476
shape_histo_point, 477
shape_trans, 2370
shape_trans_xld, 3068
Shen-Filter, 884, 887, 889, 892
Shock-Filter, 987
shock_filter, 987
sich überlagernde Regionen teilen, 2381, 2383
Sichtstrahl, 1392
sigma_image, 861
signal_condition, 2560
signal_event, 2561
sim_caltab, 1412
simplify_object_model_3d, 224
simulate_defocus, 991
simulate_motion, 992
sin_image, 737
Singulärwertzerlegung einer Matrix, 2056
Sinus eines Bildes, 737
Sinus hyperbolicus von Tupel-Elementen, 2875
Sinus von Tupel-Elementen, 2875
skeleton, 2371
Skelett beschneiden, 2122
Skelett einer Region, 1698, 2371
Skelett teilen, 2373, 2374
Slant der Lichtquelle schätzen, 327
Slant einer Textzeile, 2206
slide_image, 1651
smallest_bounding_box_object_model_3d, 186
smallest_circle, 2343
smallest_circle_xld, 3052
smallest_rectangle1, 2345
smallest_rectangle1_xld, 3053
smallest_rectangle2, 2346
smallest_rectangle2_xld, 3054
smallest_sphere_object_model_3d, 187
smooth_contours_xld, 3069
smooth_funct_1d_gauss, 2634
smooth_funct_1d_mean, 2635
smooth_image, 863
smooth_object_model_3d, 226
Sobel-Filter, 912, 914
sobel_amp, 912
sobel_dir, 914
Socket öffnen, 2607, 2610
Socket schließen, 2605
Socket-Datentyp, 2606
Socket-Deskriptor, 2606
Socket-Parameter, 2607
Socket-Parameter setzen, 2620
socket_accept_connect, 2620
Sojka-Operator, 951
solve_matrix, 2024
sort_contours_xld, 3070
sort_region, 2372
sp_distribution, 956
spatial_relation, 2347
sphärisches Mosaikbild erzeugen, 2717, 2721
split_contours_xld, 3071
split_skeleton_lines, 2373

split_skeleton_region, 2374
sqrt_image, 738
sqrt_matrix, 2026
sqrt_matrix_mod, 2027
Standardabweichung berechnen, 959
starre 2D-Transformation aus Punkt und Winkel, 2773
starre 2D-Transformation aus Punkt-zu-Gerade-Korrespondenzen, 2767
starre 2D-Transformation aus Punktkorrespondenzen, 2782
starre 3D-Transformation von Punktkorrespondenzen, 2804
Startwerte für interne Kameraparameter, 1449
stationary_camera_self_calibration, 1482
Stereo-Modell erzeugen, 305
Stereo-Modell löschen, 304
Stereomodell-Parameter erhalten, 307, 308
Stereomodell-Parameter setzen, 319
stop, 2483
Streuung von Tupel-Elementen, 2929
String aus Textdatei einlesen, 532
String aus Textfenster einlesen, 1139
String in Textdatei schreiben, 534
String teilen, 2941
Struktur der automatischen Operatorenparallelisierungsdaten, 2570
strukturierendes Element einlesen, 2089
strukturierendes Element erzeugen, 1690, 1692, 2073
sub_image, 738
sub_matrix, 2028
sub_matrix_mod, 2029
Submatrix, 2060
Submatrix setzen, 2065
Subpixel-Nulldurchgang extrahieren, 2447
subpixelgenaue Mausposition, 1068, 1069
Substring auswählen, 2942, 2943, 2949
Substring durch regulären Ausdruck ersetzen, 2938
Substring mit regulärem Ausdruck, 2935
Substring mit regulärem Ausdruck auswählen, 2939
Substring mit regulärem Ausdruck matchen, 2940
Substring suchen, 2947, 2948
Suchbaum für Komponentenmodell, 1960
suggest_lexicon, 2160
sum_matrix, 2050
Summe von Matrixelementen, 2050
Summe von Tupel-Elementen, 2933
Support-Vektor eines OCR-Klassifikators (SVM), 2219
surface_normals_object_model_3d, 228
svd_matrix, 2056
SVM-Klassifikator einlesen, 1590
SVM-Klassifikator erzeugen, 1577, 1593
SVM-Klassifikator löschen, 1576
SVM-Klassifikator schreiben, 1597
SVM-Klassifikator trainieren, 1574, 1577, 1586, 1587, 1591, 1595, 2400
SVM-Klassifikator wiederverwenden, 1545, 1584, 1585, 1597
switch, 2484
Symbol für Regionenausgabe, 1102
Symbol für Regionenausgabe setzen, 1120
Symbol lesen, 2096, 2097, 2104, 2157–2160, 2366, 2367
Symbol lesen (CNN), 2125–2128, 2131
Symbol lesen (Hyperbox), 1715, 1716
Symbol lesen (kNN), 2146–2148
Symbol lesen (MLP), 2165–2168, 2174
Symbol lesen (SVM), 2211–2213, 2220
symm_difference, 2291
symm_difference_closed_contours_xld, 2997
symm_difference_closed_polygons_xld, 2999
symmetrische Differenz von zwei Tupeln, 2924
symmetry, 964
system_call, 2492
Systemaufruf ausführen, 2492
Systemparameter (Hyperbox) setzen, 1663
Systemzeit, 2491
Systemzustand mit Kalman-Filter schätzen, 1760
Szenenfluss, 967, 969

tan_image, 740
Tangens eines Bildes, 740
Tangens eines Tupel-Elements, 2877
Tangens hyperbolicus von Tupel-Elementen, 2878
Teil des Grafikfensters kopieren, 1044
Teil des Grafikfensters verschieben, 1637
telezentrische Kamera, 1423
test_closed_xld, 3055
test_equal_obj, 2241
test_equal_region, 2349
test_region_point, 2350
test_sampset_box, 1664
test_self_intersection_xld, 3055
test_subset_region, 2350
test_xld_point, 3056
testd_ocr_class_box, 1721
Text als Zeichenobjekt erzeugen, 1087
Text einfügen in exportierter HDevelop-Prozedur, 2476
Text finden (Automatic Text Reader), 2189
Text finden (Manual Text Finder), 2189
Text im Fenster anzeigen, 684, 1132, 1142
Text-Ergebnis erhalten (Automatic Text Reader), 2192
Text-Ergebnis erhalten (Manual Text Finder), 2192
Text-Objekt erhalten (Automatic Text Reader), 2191
Text-Objekt erhalten (Manual Text Finder), 2191
text_line_orientation, 2205
text_line_slant, 2206
Textanzeige, 1136
Textdatei öffnen, 534
Textdatei schließen, 527
Textfenster öffnen, 1639

Textlabel aus 3D-Szene entfernen, 1008
Textlabel zu 3D-Szene hinzufügen, 1000
Textposition, 1136
Textur fortsetzen, 875
texture_laws, 976
Texturinspektion, 1324, 1325, 1327–1332, 1334–1336, 1338, 1339, 1343, 1345
Texturinspektionsmodell, 1324, 1325, 1327, 1329–1331, 1335, 1336, 1338, 1343, 1345
thickening, 1703
thickening_golay, 1704
thickening_seq, 1706
thinning, 1707
thinning_golay, 1708
thinning_seq, 1709
threshold, 2439
threshold_sub_pix, 2440
throw, 2485
Tiefpassfilter, 765
tile_channels, 399
tile_images, 400
tile_images_offset, 402
Tilt der Lichtquelle schätzen, 328
timed_wait_condition, 2561
Timeout für Barcode-Leser setzen, 1200, 1209
Timeout für Matching setzen, 1936, 1997
Tool öffnen, 693
Tool anzeigen, 714
Tool für optische Schriftzeichenkontrolle (OCV) einlesen, 1299
Tool für optische Schriftzeichenkontrolle (OCV) erzeugen, 1296
Tool für optische Schriftzeichenkontrolle (OCV) löschen, 1295
Tool für optische Schriftzeichenkontrolle (OCV) schreiben, 1302
Tool für optische Schriftzeichenkontrolle (OCV) trainieren, 1301
Tool schließen, 683
Tool-Position bestimmen, 711
Top-Hat-Operator, 2089, 2123
top_hat, 2123
topographic_sketch, 966
topographische Skizze, 966
train_class_gmm, 1508
train_class_knn, 1524
train_class_mlp, 1561
train_class_svm, 1595
train_dl_classifier_batch, 1616
train_dl_model_anomaly_dataset, 556
train_dl_model_batch, 657
train_generic_shape_model, 1937
train_model_components, 1974
train_sample_identifier, 1285
train_texture_inspection_model, 1343
train_variation_model, 1358
traind_ocr_class_box, 1721
traind_ocv_proj, 1301
trainf_ocr_class_box, 1722
trainf_ocr_class_knn, 2155
trainf_ocr_class_mlp, 2182
trainf_ocr_class_mlp_protected, 2184
trainf_ocr_class_svm, 2226
trainf_ocr_class_svm_protected, 2227
trainierte Support-Vektoren, 1588
trainierten OCR-Klassifikator (SVM) approximieren, 2221
trainierten SVM-Klassifikator approximieren, 1591
Trainingsdatei für optische Schriftzeichenerkennung (OCR) einlesen, 2232, 2234
Trainingsdatei für optische Schriftzeichenerkennung (OCR) schreiben, 2231, 2235
Trainingsdateien für optische Schriftzeichenerkennung (OCR) verknüpfen, 2230
Trainingsdaten (GMM) einlesen, 1504
Trainingsdaten (GMM) löschen, 1492
Trainingsdaten (GMM) schreiben, 1511
Trainingsdaten (Hyperbox) einlesen, 1661
Trainingsdaten (komponentenbasiert) einlesen, 1949, 1972
Trainingsdaten (komponentenbasiert) löschen, 1939
Trainingsdaten (komponentenbasiert) schreiben, 1980
Trainingsdaten (MLP) einlesen, 1550
Trainingsdaten (MLP) löschen, 1536
Trainingsdaten (MLP) schreiben, 1564
Trainingsdaten (SVM) einlesen, 1590
Trainingsdaten (SVM) löschen, 1577
Trainingsdaten (SVM) schreiben, 1598
Trainingsdaten eines Variationsmodells löschen, 1346, 1353
Trainingsdaten für Klassifikation einlesen, 1569
Trainingsdaten für Klassifikation schreiben, 1573
Trainingsmuster (GMM), 1498, 1501
Trainingsmuster (kNN), 1516, 1518
Trainingsmuster (MLP), 1543, 1548
Trainingsmuster (SVM), 1583, 1586
Trainingsmuster für GMM-Klassifikator wiederverwenden, 1504, 1511
Trainingsmuster für Hyperbox-Klassifikator wiederverwenden, 1661
Trainingsmuster für Klassifikator wiederverwenden, 1550
Trainingsmuster für MLP-Klassifikator wiederverwenden, 1564
Trainingsmuster für SVM-Klassifikator wiederverwenden, 1590, 1598
Trainingsmuster hinzufügen (GMM), 1489, 1490, 1564, 2397
Trainingsmuster hinzufügen (kNN), 1511, 1512, 1564, 2398
Trainingsmuster hinzufügen (MLP), 1533, 1534, 1564, 2399
Trainingsmuster hinzufügen (SVM), 1564, 1573, 1574, 2400
Trainingsparameter für Matching (formbasiert) bestimmen, 1837, 1847, 1856, 1865, 1915, 1916

Trainingsparameter für Matching (korrelationsbasiert) bestimmen, 1983
Trainingsparameter für Matching (lokal deformierbar) bestimmen, 1795
Trainingsparameter für Matching (perspektivisch deformierbar) bestimmen, 1795
Trainingsparameter für Matching bestimmen, 1107
Trainingsresultate der Klassifikation löschen (Deep Learning), 1606
trans_from_rgb, 790
trans_pose_shape_model_3d, 118
trans_to_rgb, 797
transform_funct_1d, 2636
transform_metrology_object, 1600
Transformationsmatrix, 785
translate_measure, 27
transpose_matrix, 2030
transpose_matrix_mod, 2031
transpose_region, 2287
triangulate_object_model_3d, 230
trimmed_mean, 864
try, 2486
try_lock_mutex, 2562
try_wait_event, 2562
Tupel addieren, 2853
Tupel als Dictionary-Eintrag setzen, 2892
Tupel an Socket senden, 2618
Tupel bitweise nach rechts verschieben (Rechts-Shift), 2883
Tupel dividieren, 2860
Tupel einlesen, 512
Tupel erzeugen, 2905–2907
Tupel ist größer, 2963, 2964
Tupel ist größer oder gleich, 2964, 2965
Tupel ist kleiner, 2966, 2967
Tupel ist kleiner oder gleich, 2967, 2968
Tupel mit Linksverschiebung, 2882
Tupel multiplizieren, 2872
Tupel schreiben, 515
Tupel sind gleich, 2961, 2962
Tupel sind nicht gleich, 2969, 2970
Tupel sortieren, 2903
Tupel subtrahieren, 2877
Tupel teilen, 2941
Tupel verknüpfen, 2905
Tupel von Socket einlesen, 2614
Tupel-Elemente auswählen, 2896–2901, 2921, 2922
Tupel-Elemente entfernen, 2921
Tupel-Elemente finden, 2894
Tupel-Elemente negieren, 2872
Tupel-Elemente sind größer, 2963, 2964
Tupel-Elemente sind größer oder gleich, 2964, 2965
Tupel-Elemente sind kleiner, 2966, 2967
Tupel-Elemente sind kleiner oder gleich, 2967, 2968
Tupel-Elemente sind Zahlen, 2911
Tupel-Elemente umrechnen, 2909, 2910, 2912–2915
Tupellänge, 2930
Tupeltyp, 2959, 2961
Tupeltyp abfragen, 2950–2956
Tupelwert setzen, 2468, 2481
Tuple in Vektor konvertieren, 2471
tuple_abs, 2851
tuple_acos, 2851
tuple_acosh, 2852
tuple_add, 2853
tuple_and, 2918
tuple_asin, 2853
tuple_asinh, 2854
tuple_atan, 2855
tuple_atan2, 2855
tuple_atanh, 2856
tuple_band, 2879
tuple_bnot, 2880
tuple_bor, 2881
tuple_bxor, 2881
tuple_cbrt, 2857
tuple_ceil, 2857
tuple_chr, 2909
tuple_chrt, 2910
tuple_concat, 2905
tuple_cos, 2858
tuple_cosh, 2858
tuple_cumul, 2859
tuple_deg, 2860
tuple_deviation, 2929
tuple_difference, 2923
tuple_div, 2860
tuple_environment, 2935
tuple_equal, 2961
tuple_equal_elem, 2962
tuple_erf, 2861
tuple_erfc, 2861
tuple_exp, 2862
tuple_exp10, 2863
tuple_exp2, 2863
tuple_fabs, 2864
tuple_find, 2894
tuple_find_first, 2895
tuple_find_last, 2895
tuple_first_n, 2896
tuple_floor, 2865
tuple_fmod, 2865
tuple_gen_const, 2905
tuple_gen_sequence, 2906
tuple_greater, 2963
tuple_greater_elem, 2964
tuple_greater_equal, 2964
tuple_greater_equal_elem, 2965
tuple_histo_range, 2929
tuple_hypot, 2866
tuple_insert, 2921
tuple_int, 2910
tuple_intersection, 2924
tuple_inverse, 2902
tuple_is_handle, 2950
tuple_is_handle_elem, 2950
tuple_is_int, 2951
tuple_is_int_elem, 2952

tuple_is_mixed, 2953
tuple_is_number, 2911
tuple_is_real, 2954
tuple_is_real_elem, 2955
tuple_is_serializable, 514
tuple_is_serializable_elem, 515
tuple_is_string, 2955
tuple_is_string_elem, 2956
tuple_is_valid_handle, 2957
tuple_last_n, 2897
tuple_ldexp, 2867
tuple_length, 2930
tuple_less, 2966
tuple_less_elem, 2967
tuple_less_equal, 2967
tuple_less_equal_elem, 2968
tuple_lgamma, 2867
tuple_log, 2868
tuple_log10, 2868
tuple_log2, 2869
tuple_lsh, 2882
tuple_max, 2931
tuple_max2, 2870
tuple_mean, 2931
tuple_median, 2932
tuple_min, 2933
tuple_min2, 2870
tuple_mod, 2871
tuple_mult, 2872
tuple_neg, 2872
tuple_not, 2919
tuple_not_equal, 2969
tuple_not_equal_elem, 2970
tuple_number, 2912
tuple_or, 2919
tuple_ord, 2912
tuple_ords, 2913
tuple_pow, 2873
tuple_rad, 2874
tuple_rand, 2907
tuple_real, 2914
tuple_regexp_match, 2935
tuple_regexp_replace, 2938
tuple_regexp_select, 2939
tuple_regexp_test, 2940
tuple_remove, 2921
tuple_replace, 2922
tuple_round, 2914
tuple_rsh, 2883
tuple_select, 2897
tuple_select_mask, 2898
tuple_select_range, 2899
tuple_select_rank, 2900
tuple_sem_type, 2958
tuple_sem_type_elem, 2958
tuple_sgn, 2874
tuple_sin, 2875
tuple_sinh, 2875
tuple_sort, 2903
tuple_sort_index, 2903
tuple_split, 2941
tuple_sqrt, 2876
tuple_str_bit_select, 2900
tuple_str_first_n, 2942
tuple_str_last_n, 2943
tuple_strchr, 2944
tuple_string, 2915
tuple_strlen, 2945
tuple_strrchr, 2946
tuple_strrstr, 2947
tuple_strstr, 2948
tuple_sub, 2877
tuple_substr, 2949
tuple_sum, 2933
tuple_symmdiff, 2924
tuple_tan, 2877
tuple_tanh, 2878
tuple_tgamma, 2879
tuple_type, 2959
tuple_type_elem, 2961
tuple_union, 2925
tuple_uniq, 2901
tuple_xor, 2920

Umgang mit Grafikfenster, 484, 488, 489, 1100, 1116, 1117, 1125
Umgebungsvariablen, 2935
uncalibrated_photometric_stereo, 340
union1, 2292
union2, 2293
union2_closed_contours_xld, 3000
union2_closed_polygons_xld, 3001
union_adjacent_contours_xld, 3071
union_cocircular_contours_xld, 3073
union_collinear_contours_ext_xld, 3076
union_collinear_contours_xld, 3078
union_cotangential_contours_xld, 3082
union_object_model_3d, 169
union_straight_contours_histo_xld, 1774
union_straight_contours_xld, 3086
unkalibriertes photometrisches Stereo, 340
unlock_mutex, 2563
unproject_coordinates, 1064
Untermerkmal wählen, 1570
Untermerkmale gruppieren, 1571
until, 2487
until-Schleife starten, 2483
unwarp_image_vector_field, 940
update_bg_esti, 2694
update_kalman, 1772
update_window_pose, 1065
Ursprung (Referenzpunkt) eines Matching-Modells (deskriptorbasiert) setzen, 1834
Ursprung (Referenzpunkt) eines Matching-Modells (formbasiert) setzen, 1935

Ursprung (Referenzpunkt) eines Matching-Modells (grauwertbasiert) setzen, 1681
Ursprung (Referenzpunkt) eines Matching-Modells (korrelationsbasiert) setzen, 1997
Ursprung (Referenzpunkt) eines Matching-Modells (lokal deformierbar) setzen, 1812
Ursprung (Referenzpunkt) eines Matching-Modells (perspektivisch deformierbar) setzen, 1812
Ursprung eines Matching-Modells (deskriptorbasiert), 1829, 1831
Ursprung eines Matching-Modells (formbasiert), 1914
Ursprung eines Matching-Modells (korrelationsbasiert), 1994
Ursprung eines Matching-Modells (perspektivisch deformierbar), 1808

var_threshold, 2441
Variablenfenster anzeigen, 1620
Variablenfenster verbergen, 1621
Variablentyp konvertieren, 2471, 2472
Variablenwert setzen, 2467
Variationsmodell einlesen, 1356
Variationsmodell erzeugen, 1350
Variationsmodell löschen, 1346
Variationsmodell schreiben, 1359
Variationsmodell trainieren, 1358
Variationsmodell vorbereiten, 889, 912, 1352, 1353, 1355, 1358, 2088
Variationsmodell-Qualität prüfen, 1353
vector_angle_to_rigid, 2773
vector_field_length, 941
vector_field_to_hom_mat2d, 2774
vector_field_to_real, 480
vector_to_aniso, 2775
vector_to_essential_matrix, 290
vector_to_fundamental_matrix, 292
vector_to_fundamental_matrix_distortion, 294
vector_to_hom_mat2d, 2776
vector_to_hom_mat3d, 2804
vector_to_pose, 2836
vector_to_proj_hom_mat2d, 2778
vector_to_proj_hom_mat2d_distortion, 2780
vector_to_rel_pose, 297
vector_to_rigid, 2782
vector_to_similarity, 2783
Vektor in Tuple konvertieren, 2472
Vektorfeld, 930, 940, 941
Vektorfeld-Bild, 480
Vektorlänge, 941
Verarbeitungszeit messen, 2491
Verbindungsanfrage für Socket annehmen, 2620
Vereinigung von 3D-Objektmodellen, 169
Vereinigungsmente von zwei Tupeln, 2925
verfügbare Anzeigemodi, 1112
verfügbare Compute Devices, 2498
verfügbare Cursorformen, 1643
verfügbare Datacode-Parameter, 1250
verfügbare Farbanzahl, 1110
verfügbare Farbnamen, 1108, 1109
verfügbare Fehlersuchmodi, 2508
verfügbare Fenstertypen, 1060
verfügbare Grafikmodi, 1642
verfügbare Linienbreiten, 1111
verfügbare Mauszeigerformen, 1071
verfügbare Modi zur Anzeige von Regionen, 1112
verfügbare Nachschlagetabellen (Look-Up-Tabellen, LUTs) des Grafikfensters erhalten, 1039
verfügbare Parameter des Lichtschnitt-Modells, 356
verfügbare Schriftarten (Fonts), 1138
verfügbare Slots der Operatorinformation, 2532
verfügbare Slots der Parameterinformation, 2532
verfügbare Slots der Prozedureninformation, 2532
verfügbare XLD-Kontur-Attribute, 3045, 3046
Vermessung von Kantenpaaren (1D Fuzzy-Vermessung), 4, 6
Vermessung von Kantenpositionen (1D Fuzzy-Vermessung), 8
verschachteltes Bild aus einem Mehrkanalbild erzeugen, 422
Verteilung anzeigen, 1626
Verteilung des Rauschens bestimmen, 956
verwendete HALCON-Module, 2504
volume_object_model_3d_relative_to_plane, 188
Volumen eines 3D-Objektmodells, 188
von serieller Schnittstelle einlesen, 2602
Vordergrundregion extrahieren, 2691
Vorhandensein einer Datei prüfen, 507
Vorverarbeitete Daten einziehen, 375
Vorzeichen von Tupel-Elementen, 2874
vorzeichenbehaftetes Bild mit Schwellwert segmentieren, 2431

wait_barrier, 2563
wait_condition, 2564
wait_event, 2564
wait_seconds, 2493
warten, 2493
Warteschlange für Nachrichtenobjekte löschen (Inter-Thread Kommunikation), 2536
Warteschlangenobjekt für Nachrichten erzeugen (Inter-Thread Kommunikation), 2542
Wasserscheiden extrahieren, 2462
watersheds, 2462
watersheds_marker, 2463
watersheds_threshold, 2465
weißes Rauschen hinzufügen, 954
Wert einer 1D-Funktion, 2627
Wert von 1D-Funktion interpolieren, 2628
Werte einer 1D-Funktion, 2627
Werte einer 1D-Funktion multiplizieren, 2634
Werte einer 1D-Funktion negieren, 2632
Werte von 1D-Funktion addieren, 2634
while, 2488
while-Schleife starten, 2488

wiener_filter, 993
wiener_filter_ni, 995
Window-Handle des Betriebssystems, 1050
Winkel zwischen 2D-Linie und der Reihenachse, 2640
Winkel zwischen 2D-Linien, 2638
Wort lesen (MLP), 2168
Wort lesen (SVM), 2213
Wort mittels Lexikon prüfen, 2159
Worte aus Lexikon, 2159
write_aop_knowledge, 2573
write_bar_code_model, 1212
write_calib_data, 1455
write_cam_par, 1418
write_camera_setup_model, 1455
write_class_box, 1665
write_class_gmm, 1510
write_class_knn, 1525
write_class_mlp, 1563
write_class_svm, 1597
write_class_train_data, 1573
write_component_model, 1979
write_contour_xld_arc_info, 522
write_contour_xld_dxf, 523
write_data_code_2d_model, 1262
write_deep_ocr, 2141
write_deformable_model, 1818
write_deformable_surface_model, 97
write_descriptor_model, 1834
write_dict, 2893
write_distance_transform_xld, 2678
write_dl_classifier, 1618
write_dl_model, 659
write_fft_optimization_data, 781
write_funct_1d, 2637
write_image, 497
write_image_metadata, 500
write_io_channel, 2523
write_lut, 1652
write_matrix, 2033
write_measure, 28
write_message, 2565
write_metrology_model, 76
write_ncc_model, 1998
write_object, 502
write_object_model_3d, 170
write_ocr, 1723
write_ocr_class_knn, 2156
write_ocr_class_mlp, 2185
write_ocr_class_svm, 2228
write_ocr_trainf, 2235
write_ocr_trainf_image, 2235
write_ocv, 1302
write_polygon_xld_arc_info, 525
write_polygon_xld_dxf, 526
write_pose, 2839
write_region, 505
write_sample_identifier, 1286
write_samples_class_gmm, 1511
write_samples_class_mlp, 1564
write_samples_class_svm, 1598
write_serial, 2604
write_shape_model, 1938
write_shape_model_3d, 119
write_sheet_of_light_model, 363
write_string, 1142
write_structured_light_model, 1319
write_surface_model, 148
write_template, 1682
write_texture_inspection_model, 1345
write_training_components, 1980
write_tuple, 515
write_variation_model, 1359

x_range_funct_1d, 2637
XLD als Zeichenobjekt erzeugen, 1088
XLD auf Geschlossenheit testen, 3055
XLD für Zeichenobjekt setzen, 1097
XLD-Distanztransformation deserialisieren, 2645
XLD-Distanztransformation erzeugen, 2643
XLD-Distanztransformation löschen, 2643
XLD-Distanztransformation lesen, 2676
XLD-Distanztransformation schreiben, 2678
XLD-Distanztransformation serialisieren, 2676
XLD-Kontur, 2268
XLD-Kontur abschneiden, 3058
XLD-Kontur an Socket senden, 2619
XLD-Kontur anzeigen, 686, 1028, 1038
XLD-Kontur ausschneiden, 3061
XLD-Kontur einer Ellipse erzeugen, 2978
XLD-Kontur einer NURBS-Kurve erzeugen, 2972
XLD-Kontur einer Region erzeugen, 2975
XLD-Kontur eines gerundeten Polygons erzeugen, 2974
XLD-Kontur eines Kreises erzeugen, 2971
XLD-Kontur eines Kreuzes erzeugen, 2978
XLD-Kontur eines Polygons erzeugen, 2975
XLD-Kontur eines Rechtecks erzeugen, 2983
XLD-Kontur einzeichnen, 445
XLD-Kontur enthält Punkt, 3056
XLD-Kontur glätten, 3069
XLD-Kontur in 3D-Koordinaten transformieren, 1467
XLD-Kontur rektifizieren, 1465
XLD-Kontur schließen, 3059
XLD-Kontur segmentieren, 3065, 3066
XLD-Kontur teilen, 3071
XLD-Kontur von Skelett erzeugen, 2977
XLD-Kontur von Socket einlesen, 2614
XLD-Kontur zeichnen, 1171
XLD-Konturen an Ellipsen anpassen, 3018
XLD-Konturen an Kreise anpassen, 3016
XLD-Konturen an Linien anpassen, 3021
XLD-Konturen an Rechtecke anpassen, 3023
XLD-Konturen auswählen, 3047, 3049, 3051
XLD-Konturen bearbeiten, 2291, 2982, 2993–2997, 2999–3002, 3008, 3012, 3014, 3030, 3039, 3040, 3042, 3044, 3047, 3049, 3051, 3065,

3066, 3068, 3071, 3073, 3076, 3078, 3082, 3086
XLD-Konturen einlesen, 517
XLD-Konturen erzeugen, 884, 887, 889, 892, 912, 916, 917, 919, 921, 1171, 1172, 2440, 2446, 2447, 2974, 2975, 2977, 2978, 2983
XLD-Konturen mit lokalem Grauwertmaximum auswählen, 3037
XLD-Konturen schreiben, 522, 523
XLD-Konturen sortieren, 3070
XLD-Konturen zusammenfügen, 3000, 3001, 3062, 3071, 3073, 3076, 3078, 3082, 3086
XLD-Parallelen erzeugen, 2981
XLD-Parallelen extrahieren, 2984
XLD-Parallelen verbinden, 3038
XLD-Polygone einlesen, 519, 520
XLD-Polygone erzeugen, 2982
XLD-Polygone schreiben, 525, 526
XLD-Polygone und XLD-Parallelen verbinden, 3060
XLD-Polygonlinie, 3089
XLD-Polygonpunkte und -linien, 3091
xyz_to_object_model_3d, 239

y_range_funct_1d, 2638

Zeichen aus Region extrahieren, 2196
Zeichen aus Textdatei einlesen, 529
Zeichen aus Textfenster einlesen, 1139
Zeichen in String (Zeichenkette) suchen, 2944, 2946
Zeichen zu Trainingsdatei hinzufügen für optische Schriftzeichenerkennung (OCR), 2229
Zeichen-Nachschlagetabelle (Look-Up-Tabelle, LUT) für optische Schriftzeichenerkennung (OCR) erzeugen, 1718
Zeichenkettengröße (String), 1136
Zeichenkettenlänge (String), 2945
Zeichenmodus für Regionen, 1100
Zeichennamen aus der Trainingsdatei der optischen Schriftzeichenerkennung (OCR), 2232, 2233
Zeichenobjekt löschen, 1078
Zeichenobjekt mit HALCON-Fenster verknüpfen, 1077
Zeichenobjekt von einem graphischen HALCON Fenster entfernen, 1090
Zeichenregionen extrahieren, 2194
Zeiger (pointer) auf Bild, 484, 486, 487
Zeiger (pointer) auf Fensterdaten, 1053
Zeile aus Textdatei einlesen, 530
Zeilenkamera für Konturverarbeitung verwenden, 3062
Zeilenregion erzeugen, 2271
Zeilenvorschub im Text erzeugen, 528
zeilenweises Bildhistogramm, 981
zero_crossing, 2446
zero_crossing_sub_pix, 2447
zero_crossings_funct_1d, 2638
zoom_image_factor, 819
zoom_image_size, 820
zoom_region, 2288
zufällige Region erzeugen, 2263, 2264
Zufallszahlen, 2907
zur seriellen Schnittstelle schreiben, 2604
zusammengesetzte Symbole einlesen, 2096, 2099, 2291, 2357
Zusammenhänge zwischen den Komponenten eines Matching-Modells ändern, 1970
Zusammenhänge zwischen den Komponenten von Matching-Modell, 1962
Zusammenhangskomponenten, 2357
Zusammenhangskomponenten im Hintergrund, 2351
Zusammenhangskomponenten und Löcher einer Region zählen, 2305
Zwischenspeicher (Puffer) der seriellen Schnittstelle löschen, 2599